Metaheuristics: Foundations

Fernando Lobo

Universidade do Algarve

Foundations

- Combinatorial Problems
- Problems and Problem Instances
- Decision vs. Optimization problems
- Candidate Solutions, Search Space

Foundations

- Objective function
- Systematic vs. Local Search
- Neigbourhoods
- Local vs. Global Minimum
- Feasible and unfeasible solutions

Combinatorial Problems

- Solving a combinatorial problem means finding an ordering, or assignment, or grouping, of a finite set of objects that satisfy a number of conditions
- TSP and SAT are examples of combinatorial problems
  - TSP involves ordering a finite set of cities
  - SAT involves the assignment of boolean values to variables

Problems and Instances

- It is important to distinguish between problem and problem instance
- TSP is a problem
- If we consider a specific set of cities and their distances, we have a TSP problem instance

Decision vs. Optimization problems

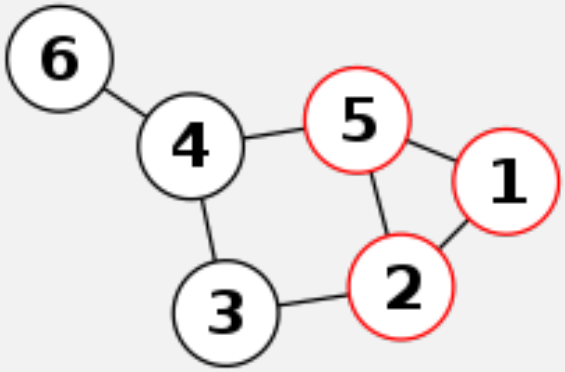
- A decision problem is one that has a yes/no answer
- Example: SAT is a decision problem
  - given a boolean formula F, is there a variable assignment that satisfies F?
- Many decision problems have a related optimization problem
  - MAXSAT is the optimization version of SAT

Decision vs. Optimization problems

- Another example: Clique Problem
- Given a graph $G = (V, E)$ and an integer k, does G contains a clique of size k?
  - A click of size k is a complete subgraph of G with k vertices. That is, it is a subset $V' \subseteq V$ such that $|V'| = k$, and every two vertices in V' are connected by an edge.

Decision vs. Optimization problems

- Optimization version of the Clique Problem: Maximum Clique Problem
- Given a graph $G = (V, E)$ find the size of its maximum clique



- What's the maximum clique size of the Facebook graph?

Optimization problems

- An optimization problem involves optimizing (i.e. maximizing or minimizing) a function
- This function is traditionally called an *objective function* or *fitness function*
- This function can be subject to several constraints

Candidate solutions

- A *candidate solution* is a member of the set of all possible solutions for the problem
  - A tour that visits every city exactly once and returns back to the original city is a candidate solution for a TSP problem instance defined over those cities.
    - ⋆ In graph theory, such a tour is called a *Hamiltonian cycle*
    - ⋆ The TSP can be modeled as a weighted graph. Each city corresponds to a vertex of the graph, and the distance between city A and city B corresponds to the weight of edge A–B
    - ⋆ A Hamiltonian cycle of a graph is a cycle that visits each vertex exactly once
  - A specific assignment of boolean values to the variables $x_1, \ldots, x_n$, is a candidate solution for a SAT problem instance defined over those variables

Search space

- The set of all candidate solutions to a problem defines the *Search Space* for that problem
- SAT example:
  - For an instance with n variables, the search space is the set of all possible boolean assignments to those n variables
  - There is a one-to-one mapping to the set of all binary strings of length n
  - The size of such a search space is 2^n

Search space

- TSP example:
  - For an instance with n cities, the search space is the set of all Hamiltonian cycles of the corresponding TSP graph
  - The size of the search space is $n!/n = (n-1)!$
    - ★ we divide by n because it doesn't matter where we start
    - ★ another way of thinking: starting from an arbitrary city we need an ordering of the remaining $n-1$ cities
    - ★ example for $n = 4$: 1-2-3-4 represents the Hamiltonian cycle 1-2-3-4-1
    - ★ these 4 permutations of the cities 1,2,3,4 correspond to the same Hamiltonian cycle: 1-2-3-4, 2-3-4-1, 3-4-1-2, 4-1-2-3
  - In the case of a Symmetric TSP ($\text{dist}(x, y) = \text{dist}(y, x)$, for every pair of cities x and y) the size of the search space is halved: $(n-1)!/2$

Generating candidate solutions

- We can solve problems by generating and evaluating candidate solutions
- How to generate candidate solutions?
  - uniformly at random from the search space
  - incrementally using heuristic information
    - ★ Example: Nearest Neighbour Heuristic for TSP: Start with a city at random. Call that the *current city*. Then choose among the remaining cities the one whose distance to the current city is minimal.
  - by doing a modification to an existing solution

Evaluating candidate solutions

- For many problems, it is relatively easy to evaluate a solution
  - Side note: easy doesn't mean cheap. It can still take a considerable amount of time.
- Evaluating a solution can be done by computing the objective function value of that solution
- . . . or in some cases we may evaluate the solution in an incremental manner
  - Can be much more efficient
  - More on this later. . .

Systematic search

- We can solve a search/optimization problem instance by systematically visiting the entire search space
- It's only practical for small search spaces
- SAT example: generate and evaluate all possible 2^n boolean assignments to the n variables

Local search

- Start with some candidate solution, and evaluate it
- Make a modification to that solution, obtaining a new candidate solution
- Either keep the original or the new candidate solution, according to some criteria
- Iterate the previous two steps

Local search and neighbourhoods

- Local search often relies on the notion of *neighbourhood*
- The neighbourhood $\mathcal{N}(x)$ of a solution x is the set of solutions that can be obtained from x in a single search step
- Example: 1-bit neighbourhood for SAT
  - Set of solutions that can be obtaining by choosing a variable and flipping its value
  - Formally: $\mathcal{N}(x) = \{y \in S : d(x, y) = 1\}$
    - S is the search space
    - $d(x, y)$ is the Hamming distance between x and y
    - That is, $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$
    - Example: $d(1000, 1111) = 3$

k-bit neighbourhood

- The previous 1-bit neighbourhood is quite common for problems whose solutions can be mapped to bitstrings
- Can be extended to a k-bit neighbourhood
- That is, the set of solutions that can be obtaining by choosing k variables and flipping their values
- The size of the neighbourhood becomes $\binom{n}{k} = \frac{n!}{(n-k)!k!}$

Other kind of neighbourhoods

- A k-bit neighbourhood makes no sense for a problem like the TSP
- Neighbourhoods depend on the candidate solution representation
- And for a given representation, different neighbourhoods can be specified

k-exchange neighbourhood for TSP

- A common neighbourhood for the TSP
- Edges of the graph are viewed as components
- Two candidate solutions are neighbours if and only if one solution can be obtained from the other by removing at most k edges and reconnecting the resulting partial tours

TSP: example of 2-exchange move

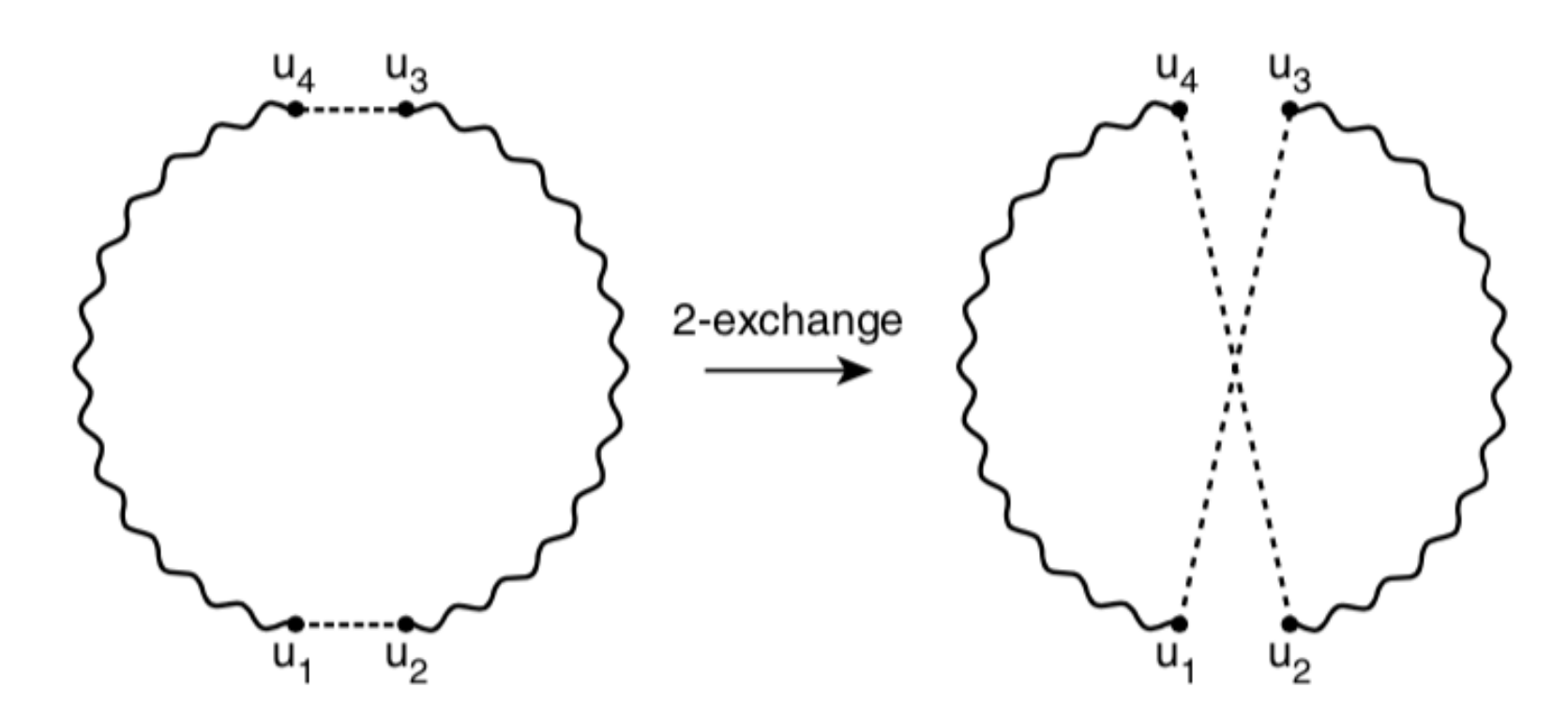


Figure taken from Hoos and Stützle's book

Incremental evaluation in a 2-exchange TSP move

- After a 2-exchange move there's no need to recompute the total tour length from scratch
- We just need to add the contribution of the 2 new edges and subtract the contribution of the 2 removed edges
- Evaluation only needs 4 arithmetic operations instead of n
- For large TSP instances this can give significant time savings, especially because the evaluation of candidate solutions is done many times (sometimes thousands or even millions) over the course of the execution of a search algorithm

Local Minimum

- Solution x is a *local minimum* if and only if $f(x) \leq f(x')$ for all $x' \in \mathcal{N}(x)$
  - That is, no neighbour of x is better than x with respect to a minimization problem
- Solution x is a *strict local minimum* if and only if $f(x) < f(x')$ for all $x' \in \mathcal{N}(x)$
- For maximization problems, we get similar definitions for *local maximum* and *strict local maximum*

Global Minimum

- Solution x is a *global minimum* if and only if $f(x) \leq f(x')$ for all $x' \in S$
  - That is, no solution in the entire search space S is better than x with respect to a minimization problem
- For maximization problems, we get a similar definition for *global maximum*

Feasible and unfeasible solutions

- For some problems there are a number of constraints that must be satisfied
- If a candidate solution satisfies all constraints, we say the solution is *feasible*
- Otherwise the solution is *unfeasible*
- Example: Knapsack problem
  - Given a set of items, each with an associated weight and profit value, find a subset of items such that the total profit is maximized, and such that the total weight does not exceed a maximum capacity C
  - Any subset of items is a candidate solution
  - But if the capacity C is exceeded, that solution becomes unfeasible