**COP - Combinatorial Optimization Problem**

problem is typically to find values for variables to optimize some objective function

optimizing over some discrete structure gives a **Combinatorial Optimization Problem**

**Exact algorithms:**

set of instructions that result in solution to problem when followed correctly

assumed to give optimal solution

hard to implement for large problems

large running times

**Heuristic algorithms:**

do not guarantee optimal solution

can return a solution that is arbitrarily bad compared to the optimal solution

successful in solving large real-world problems

**Local search**

small modification to solution gives “neighbour solution”

certain set of operations on a solution gives set of neighbour solutions, a “neighborhood”

**Neighborhood**

mapping from a solution to the set of possible solutions, reached by a move:

* N: S -> 2s

most often defined by a given operation on a solution

Often simple operations:

– Remove an element of the solution

– Add an element to the solution

– Interchange two or more elements of a solution

**Global and Local Optima**

assume we want to solve “max f(x)”

let x be our current solution in a local search

if f(x) >= f(y) for all y in feasible solutions, they x is a global optimum (of f)

now assume N(x) is the set of neighbors of x

if f(x) >= f(y) for all y in N(x), then x is a local optimum (of f, respecting the neighborhood)

**all global optima are also local optima (with respect to any neighborhood)**

**Local Search/Neighborhood Search**

start with an initial solution

search through neighborhood of that solution for better ones

define a strategy for which solution to pick from neighborhood to be new solution

define a stopping criteria

define what happens when the neighbourhood does not contain a better solution (local optimum):

* if solution “x” is better than all solutions in neighborhood, x is **local optimum**

options:

* heuristic method
* iterative method
* small changes to given solution
* alternative search strategies:
  + accept first improving solution (”First Accept”)
  + search the full neighbourhood and go to the best improving solution (“Best Accept”)
* Strategies with randomization
  + Random neighborhood search (”Random Walk”)
  + ”Random Descent”

A move is the process of selecting a given solution in the neighborhood of the current solution to be the current solution for the next iteration

**“Best Accept” and “First Accept”**

Both stop at local optimum

“Best accept” – selects best neighbor

“First accept” – selects first better neighbor found

A picture containing diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**Metaheuristics**

can be used to guide a Local Search, and help it escape a local optimum

classification:

x/y/z:

* x = A (adaptive memory) or M (memoryless)
* y = N (systematic neighborhood search) or S (random sampling)
* z = 1 (one current solution) or P (population of solutions)

**Simulated Annealing (Arrefecimento Simulado)**

usually only achieves an approximate solution to the global optima

the goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy

algorithm explanation:

temperature decreases over time

initially, temperature is high, accepting almost any alternative solution given, even if it’s worse than the current solution

as time goes on, the temperature decreases, making the algorithm more and more greedy, having a lower tolerance got accepted solutions and only accepting solutions which actually improve the system

example:

assuming maximization problem

A = Eval(random neighbor) – Eval(current solution)

if A > 0, accept (improving neighbor)

else accept it with probability eA/T (T = temperature)

if move not accepted, try another neighbor

if temperature reduced too fast -> stop in some local optimum too early

if temperature reduced too slow -> too slow convergence

**Iterated Local Search**

pick random starting solution

perform local search

repeat (record best local optimum encountered)

Local Search algorithm defines a set of locally optimal solutions

Iterated Local Search metaheuristic searches among these solutions, rather than in the complete solution space

* The search space of the ILS is the set of local optima
* The search space of the LS is the solution space

**Variable Neighborhood Search**

basic principle: change the neighborhood during the search

* local minimum in a neighborhood is not necessarily local minimum in another neighborhood
* global minimum is locally optimal in all neighborhoods
* normally, local minima in one neighborhood is close to others

**Guided Local Search**

metaheuristic used to guide a neighborhood search

tries to overcome local optima by “removing” them:

uses extended move evaluation function:

ev. function = original objective function + penalties

assumes we can find features of solution we can penalize

example of features: city A next to city B

features have a cost:

represents influence of solution on evaluation function

GLS tries to avoid costly features

penalties start at 0

when search reaches local optimum:

penalty increased for some of features of current solution

makes current solution worse compared to neighbor solutions

increase penalty of feature that has highest utility value

penalties only adjusted when reached local optimum, and only features included in local optimum

**Tabu Search**

allows moving to worse position: can exit local optima

local search with “best improvement” strategy (always select best move)

some neighbors are “tabu” and cannot be selected:

defined by **tabu criterion**

tabu neighbors might be selected anyways if deemed good enough, **aspiration criterion**

tabu criterion:

used to avoid cycling between solutions

avoid visiting again same solution

solutions are tabu for certain amount of time/iterations (**tabu tenure)**

how:

find some way to represent recent solutions and avoid returning to them

tabu tenure (TT):

dynamic/reactive tenure recomended:

Dependent on the Tabu attributes

Detect stagnation → increase TT

When escaped → reduce TT

aspiration criterion:

possible problem:

One of the neighbors is very good, but we cannot go there because some attribute is tabu

solution:

If we somehow know that the solution is not visited before, we can allow ourselves to move there anyway

simplest:

allow new best solutions, otherwise keep tabu status

criteria based:

degree of feasability

degree of change

etc

if all moves are tabu:

choose best move, or randomly

**Frequency based memory**

frequency counters:

residency-based (ex. how often has an edge been in the solution)

transition-based (ex. how often has the edge status been changed)

**Genetic Algorithms**

Diagram

Description automatically generated

Evaluation of individuals:

adaptability: “fitness”

Relates to the objective function value for the problem

fitness is maximized

Used in selection (”Survival of the fittest”)

often between [0, 1]

**Memetic Algorithms**

Genetic Algorithm with Local Search as improvement mechanism

In a sense this elevates the population search to a search among locally optimal solutions, rather than among any solution in the solution space