Computational Techniques

Metaheuristics

Stefan Rath

Syllabus

- Design and Implementation of a Metaheuristic (group of 2 students)
  - Problems: p-median problem, Travelling Salesman Problem (TSP), Quadratic Assignment Problem (QAP)
  - Metaheuristics
    - (Iterated) Local Search ((I)LS): 1 point (for each group member)
    - Simulated Annealing (SA), Variable Neighborhood Search (VNS): 2 points
    - Tests and evaluation of
      - different parameters
      - different neighborhood structures/operators
      - runtime analysis
      - additional 1(-2 or even more) point(s)

Syllabus

- Only one metaheuristic per group
- Same number of points for each group member (exception: part of group decides to do additional work)
- Grading scale

>= 6 points	sehr gut
>= 5 points	gut
>= 4 points	befriedigend
>= 3 points	genügend

Syllabus

- Programming language of your choice, e.g. Python, Java, C++
- Frameworks can be used but need to be documented
- Minimum requirements:
  - Report best found solution per run
  - Report total run time
  - Runtime when best solution was found
- Documents to be submitted by email: source code, presentation slides, optional additional documentation, short description of how work was shared beteween group members

Syllabus

- Register group by 8.12. per email
  - names, metaheuristic, presentation date
- Concept and design discussion on 15.12.
  - Design details of your metaheuristic (neighborhood structures, start heuristic, termination criterion,..)
- Presentation in January
  - Choose one of three possible dates (12.1, 19.1. or 26.1)

Schedule

- 24.11.
- 01.12.
- (8.12. No class, deadline: registration)
- 15.12. Concept and design discussion
- 12.1. Final presentations
- 19.1. Final presentations
- 26.1. Final presentations

Problems

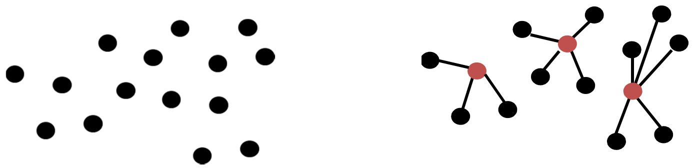
- Traveling Salesman Problem (TSP)
  - http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
  - http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/
  - Instances (mandatory for exercise): berlin52, ch130, …
- Quadratic Assignment Problem (QAP)
  - http://www.seas.upenn.edu/qaplib/
  - http://www.seas.upenn.edu/qaplib/inst.html#BO
  - Instances (mandatory for exercise): Tai50a,Bur26a, …

P-median problem

- A Median is a point that minimizes the total distance between it and the users
- p-Median Problem
  - Given is a set of clients/users and a set of potential facilities. These sets may also be identical. (i.e. each client serves as potential facility)
  - Locate exactly p facilities to minimize the total cost (distances) between users and their closest facility.

P-median problem

- n number of candidate locations on the network (possibly all vertices)
- p number of facilities to be located
- Number of feasible solutions: $\binom{n}{p} = \frac{n!}{p!(n-p)!}$



P-median Problem

- Minimize the sum of the distances
- y_k is a binary variable indicating whether we choose facility *k* or not
- x_{ki} indicates whether client *i* is served by facility *k*
- $\text{minimize}[\sum_{i=1}^{n} \sum_{k=1}^{n} d_{ki} x_{ki} \quad]$
- S.t.
- We have to choose p locations as a service facility
- $\sum_{k=1}^{n} y_k = p$
- Every client has to be assigned to a service facility
- $\sum_{i=1}^{n} x_{ki} = 1$,*k*=*1*…*n*
- Client i can only be assigned to chosen facilities
- $y_k \geq x_{ki}, k, i = 1 \ldots n$

Benchmark instances

- Lorena instances:
  - http://www.lac.inpe.br/~lorena/instancias.html
- Instances from TSPLIB
  - http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
  - (symmetric TSP instances)
  - Use fl1400, pcp3038 and rl 5934 with different values for p
- Mandatory instances for the exercise:
  - pmedian324.txt p=5 and p=16 (Lorena)
  - Fl1400 p=50 and p=200 (TSPLIB)

Heuristics

- „Heuristic“ Greek for find/discover
- Want to find a „good“ solution
- Not necessarily the optimum

- Recommended Literature: Handbook of Metaheuristics, Gendreau, Potvin, 2010, Springer

Heuristics vs. exact methods

- Exact methods:
  + Find optimal solution
  - Runtime
  - Many problem instances too large to solve to optimality
- Heuristics
  + Faster
  + Can tackle hard problems
  - No information about solution quality

Heuristics

- Constructive methods: construct a solution from scratch
- Iterative improvement methods: improve a given starting solution
- Metaheuristics: framework to guide problem-specific heuristics
- Approximation Methods: worst case gap to the optimal solution (provable solution quality)

Heuristics

- Greedy: execute best decision at each stage
- Look-ahead: use some kind of look-ahead information

Historical background

- Difficult combinatorial problems have been around for a long time (e.g., TSP)
- Complexity theory was developed in the seventies (NP-hardness)
- => Solving many important problems is not possible in reasonable time!
- What can be done in practical settings where solutions are needed?!?

Classical local improvement heuristics

- Exchange methods
- Principle:
  - Start with a feasible, initial solution.
  - Apply a sequence of local modifications to the current solution as long as an improvement of the objective function is possible
- Properties:
  - Solution never deteriorates
  - Solution is always feasible

Local optima

- Local Optimum: Local improvement methods stop when they encounter a ***local optimum*** (w.r.t. to the allowed modifications).
- Solution quality (and CPU times) depends on the “richness” of the set of transformations allowed at each iteration of the heuristic.

Metaheuristics

- Generic heuristic solution approaches designed to control and guide specific problem-oriented heuristics
- Deteriorating and infeasible solutions can be allowed

Classification of Metaheuristics

- Nature-inspired vs. non-nature inspired
- Population-based vs. single point search
- One vs. various neighborhood structures
- Memory usage vs. memory-less methods

Nature inspired Metaheuristics

- Genetic/Memetic Algorithms
- Simulated Annealing
- Ant Colony Optimization

Local Search Based Metaheuristics

- Tabu Search
- Variable Neighborhood Search
- Adaptive Large Neighborhood Search
- …

Neighborhood

- *N(s)* is the neighborhood of *s*
- *N(S)* = {*solutions obtained by applying a single local transformation to S*}.
- The neighborhood is the set of all solutions reachable from *s* by performing a special operation.
- A *move* is the choice of a solution *s'* from the neighborhood *N(s)*

Concepts

- Intensification: explore more thoroughly the portions of the search space that seem "promising"
- Diversification: forcing the search into unexplored areas of the search space

Simulated Annealing

- Statistical Mechanics
- Kirkpatrick, Gelatt and Vecchi (1983)

Simulated Annealing

– Improving solutions are always accepted

– Inferior solutions are accepted with probability

$$\exp\left(\frac{-(f(\omega') - f(\omega))}{T}\right)$$

– Temperature is decreased during search process

Simulated Annealing

- It is like a controlled random walk in the space of solutions:
  – Improving moves are always accepted;
  – Deteriorating moves are accepted with a probability that depends on
    - the amount of the deterioration
    - the temperature, which is decreases with time.

Simulated Annealing

- Construct an initial solution *s*
- Initialize the temperature *T*
- **While** termination conditions not met **do**
  – s‘← PickAtRandom(N(s))
  – **If***(f(s‘) < f(s))*
    s ← s’
  – **Else**
    - Accept *s’* with probability *p(T, s’, s)*

  endIf
- Update *T*
- **endwhile**

Tabu Search

- Best improvement local search
- Tabu List (short term memory): keeps track of the most recently visited solutions
- Tabu List avoids visiting recently visited solutions
- Prevents cycling
- Deteriorating moves are accepted

Tabu Search

- Construct an initial solution *s*
- **While** termination conditions not met do
  - *s* ← Choose best solution of *N(s)* that is not tabu
  - Update best found solution
  - Update tabu list
- **endwhile**

Tabu Search

- Tabu List:
  - Tabu tenure: length of the tabu list
  - Keeping track of complete solutions is inefficient
  - Solution attributes are stored
- Aspiration Criteria: A solution that is tabu can be accepted if it improves the best found solution

Variable Neighborhood Search

- Based on dynamically changing neighborhood structures
- Neighborhoods are usually used in ascending order
- Three phases: shaking, local search and acceptance decision

Variable Neighborhood Search

Initialization: Select a set of neighborhood structures N_k (k=1,…,k_{max})

Repeat until a stopping condition is met:

- 1. Set k $\leftarrow$1;
- 2. Repeat the following steps until k = $k_{max:}$
  - *Shaking*. Generate a point *x'* at random from k^{th} neighborhood
  - *Local search*. Local search with *x'* as initial solution;
  - *Acceptance Decision*. If this local optimum is better than the incumbent, move there (x$\leftarrow$x''), and continue the search with N_1 (k$\leftarrow$1); otherwise, set k$\leftarrow$k+1;

Adaptive Large Neighborhood Search

- Destroy and repair principle
- Destroy operator: removes *k* elements
- Repair operator: reinserts these *k* elements
- Operators have a score based on past success
- Operators with a higher score are more likely to be chosen

Adaptive Large Neighborhood Search

- s $\leftarrow$ InitialSolution
- **While** termination conditions not met do
  - select a destroy and a repair operator based on their scores
  - perform destroy and repair
  - Acceptance decision
  - Update scores of operators
- **endwhile**

Genetic Algorithms

- Population based method
- Natural selection
- In each iteration: generate the next generation of individuals from the current population

Genetic Algorithms

- Three phases:
  - Recombination/crossover
  - Mutation/modifications
  - Selection
- *Fitness* of an individual: objective function
- Survival of the fittest

Genetic Algorithms

- Example Crossover: (one-point)

  10011|111

  01010|001

  gives

  10011|001

  01010|111

Genetic Algorithms

- Example: the order-crossover
- Two cutting sites are randomly selected and copied in the offspring

P1: 1-3-2-|6-4-5|-9-7-8
P2: 3-7-8-|1-4-9|-2-5-6

C1: x-x-x|1-4-9|x-x-x
C2: x-x-x|6-4-5|x-x-x

- Then the missing nodes are copied from the parents

C1: 3-2-6|1-4-9|5-7-8
C2: 3-7-8|6-4-5|1-9-2

Genetic Algorithms

- P← GenerateInitialPopulation()
- Evaluate (P)
- **While** termination conditions not met do
  - P' ← Recombine (P)
  - P'' ← Mutate (P')
  - Evaluate (P'')
  - P ← Select (P''∪ P)
- **endwhile**

Genetic Algorithms

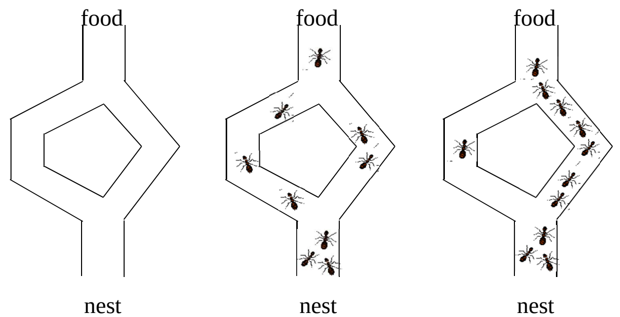
- Individuals:
  - Solutions or partial solutions
  - Encoded as bit-strings or permutations of integer numbers
- Infeasibility after recombination
  - Reject
  - Repair
  - Penalize
- Intensification strategy: apply local search (Memetic Algorithms)
- Diversification strategy: random mutation

Ant Colony Optimization

- Inspired by the behavior of real ants
- Ants find the shortest path between food source and their nest
- Ants lay pheromone
- As many ants follow a trail, the concentration of pheromone is intensified
- Trails with a high concentration of pheromone are reinforced
- Pheromone of unused trails evaporates

ANT COLONY OPTIMIZATION

- Ants use both trails
- Ants using the shorter trail are back faster
- Therefore, the pheromone concentration on the shorter trail is stronger



Ant Colony Optimization

- Each ant constructs a solution by adding solution components to a partial solution
- Probabilistic choice of next solution based on pheromone values and heuristic information
- Pheromone Evaporation

Ant Colony Optimization

- Initialize Pheromone Values
- **While** termination conditions not met do
  - AntBasedSolutionConstruction()
  - UpdatePheromone()
- **endwhile**

Other developements

- Hybridizations
- Parallel implementations (cooperative search)
- Matheuristics