PhD course on Metaheuristics

Seminar 1: Chapter 1.5-1.9 in E.-G. Talbi (2009).

Johan Högdahl

October 13, 2015

Outline

1.5 Constraint handling

- The constraints can be of any kind; nonlinear, linear, equality or inequality
- Strategies:
  - Reject strategies
  - Penalizing strategies
  - Repairing strategies
  - Decoding strategies
  - Preserving strategies

1.5 Constraint handling

Reject strategies

- Infeasible solutions is rejected
- A good strategy when the portion of infeasible solutions are small compared to the hole search space
- Not so good if the search path goes through a infeasible region

Penalizing strategies

- The approach is to add a penalizing term to the unconstrained objective:

$$f_p(s) = f(s) + \lambda c(s)$$

where λ is a weight and $c(s)$ is the cost of constraint violation.

1.5 Constraint handling

Penalizing strategies

- **Violated constraints:** count the number of violated constraints

$$f_p(x) = f(x) + \sum_{i=1}^{m} w_i \alpha_i$$

- **Amount of infeasibility:** Penalize the distance to the feasible region

$$f_p(x) = f(x) + \sum_{i}^{m} w_i d_i^k$$

where d_i^k is a distance metric for the constraint i and k is usually 0 or 1.

1.5 Constraint handling

Penalizing strategies

- The strategies in the previous slide used **static** penalizing factors, w_i.
- **Dynamic:** Change the factors w_i during the time. The aim could be to allow infeasibility early in the search and penalize more as the search progresses.

$$f_p(x,t) = f(x) + \sum_{i=1}^{m} w_i(t) d_i^k$$

- **Adaptive:** Exploit information of the search process to update the factors w_i. For example: increasing the factors if many infeasible solutions are generated and decrease the factors when many feasible solutions are generated.

1.5 Constraint handling

Repairing strategies

- Repairing strategies is heuristic algorithms to transform an infeasible solution into a feasible one.
- They are specific to the optimization problem.
- Example: If a infeasible solution to the knapsack problem is obtained (that is, the chosen objects does not fit in the knapsack). A repairing strategy is to remove the object that maximizes the ratio u_i/w_i until the solution is feasible.

1.5 Constraint handling

Decoding strategies

- Indirect encoding can be used to reduce the number of constraints and the size of the genotype
- To generate a complete solution a decoding function is needed
- The decoding function associates a representation $r \in R$ to a feasible solution $s \in S$ in the search space.

1.5 Constraint handling

For the decoding function the following properties must hold:

- For each solution $r \in R$, corresponds a feasible solution $s \in S$.
- For each feasible solution $s \in S$, there is a representation $r \in R$ that corresponds to it.
- The feasible solutions in S must have the same number of corresponding solutions in R.
- The computational complexity of the decoder must be reduced.
- The representation space must have the locality property in the sense that distance between solutions in R must be positively correlated with the distance between feasible solutions in S.

1.5 Constraint handling

Preserving strategies

- This type of strategies is problem specific
- Specific representation and operators to ensure feasible solutions
- Is claimed to be a efficient strategy, but for some problems it is difficult even to find a feasible initial solution to start the search.

1.6 Parameter tuning

- When using metaheuristics many parameters need to be tuned
- The parameters can be numerical coefficient and different search components
- This allows larger flexibility and robustness
- The choice of parameters usually have huge impact in solving efficiency
- Two different strategies; off-line parameter initialization and online parameter tuning

1.6 Parameter tuning

Off-line parameter initialization

- Experimental design
  - A simple strategy to tune the parameters is to tune them one-by-one
  - No guarantee to find optimal parameters
  - Suppose n parameters with k level each $\Rightarrow n^k$ combinations $\Rightarrow$ many experiments is necessary!
  - Statistical methods, such as Latin hypercube or fractional design can be applied to reduce the number of experiments. Also methods used in machine learning (racing algorithms) can be applied.
- Meta-optimization
  - The problem of finding the best parameters may be formulated as an optimization problem
  - This problem can then be solved using metaheuristic $\Rightarrow$ meta-metaheuristics

1.6 Parameter tuning

Off-line parameter initialization

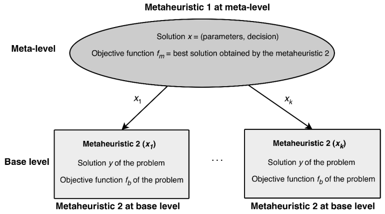

Metaheuristic 1 at meta-level
Meta-level
Solution x = (parameters, decision)
Objective function f_m = best solution obtained by the metaheuristic 2
x_1
x_k
Base level
Metaheuristic 2 (x_1)
Solution y of the problem
Objective function f_b of the problem
Metaheuristic 2 at base level
...
Metaheuristic 2 (x_k)
Solution y of the problem
Objective function f_b of the problem
Metaheuristic 2 at base level

FIGURE 1.25 Meta-optimization using a meta-metaheuristic.

Figure: Meta-optimization

$$f_m(x) = f_b(Meta(x))$$

1.6 Parameter tuning

Online parameter tuning

- Drawbacks with off-line parameter tuning
  - Computational expensive and it may be necessary to tune the parameters for each instance
  - The effectiveness of the parameters may change during the search
- Online approaches may be classified as
  - Dynamic: A random or deterministic update is carried out without taking in account the search progress
  - Adaptive update: Updates the parameters according to the search progress. Uses the search memory. A subclass is self-adaptive update
- Off-line and online parameter tuning will be dealt with later in the book.

1.7 Performance Analysis of Metaheuristics

- Performance analysis of metaheuristics need to be done on a fair basis
- The following figure illustrates how this analysis should be carried out

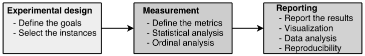


FIGURE 1.26 Different steps in the performance analysis of a metaheuristic: experimental design, measurement, and reporting.

1.7 Performance Analysis

Experimental design

- Defining the goals of the computational experiments
- Selecting problem instances
  - Real-life or Constructed (randomized or synthetic)
  - The set of instances must be diverse in terms of size, difficulties and structure
- Selection of parameter values
  - The problem instances is divided into a calibration and validation instances
  - The parameters must be the same for all instances, they are chosen according to the principles of section 1.6

1.7 Performance Analysis

Measurements

- Quality of solution
  - Global optimal solution
  - Lower bound
  - Best known solution
  - Requirements or actual implemented solution
- Computational effort
  - Theoretical analysis: worst-case complexity or average-case complexity
  - Empirical analysis:
    - Computation time
    - Counting the number of objective function evaluations
    - Stopping criteria must be chosen

1.7 Performance Analysis

Measurements

- Robustness
  - Insensitivity against small deviations in input instances or parameters. Lower variability of the solutions is means better robustness.
  - The parameters may be overfitted which gives poor robustness. The metaheuristics should be able to perform well on different instances or problems using the same parameters
- Statistical analysis
  - When the experiments have been conducted, a statistical analysis on the performance indicators should be carried out.
  - Under normality conditions, a t-test or ANOVA test may be carried out. Otherwise a nonparameteric analysis is performed.
  - Checking the normality conditions can be done using for instance the Kolmogorov-Smirnov test.

1.7 Performance Analysis

Measurements

- Ordinal analysis
  - We want to compare n metaheuristics from m experiments
  - For each method l a set of ordinal values o_{lk} $(1 \leq k \leq m)$ are attached
  - The ordinal values o_{lk} denotes the rank of the metaheuristic l from experiment k $(1 \leq o_{lk} \leq n)$
  - To obtain an aggregated rank different voting methods can be used. For example:
    - Borda count voting method
    - Copeland's method

1.7 Performance Analysis

Reporting

- The results must be interpreted with respect to the defined goals and performance measurements
- The results should be visualized to make the interpretation of the numerical results easier.
- The metaheuristic must be well documented to be reproduced
- If possible, the use of a common software framework is to prefer.

1.7 Performance Analysis

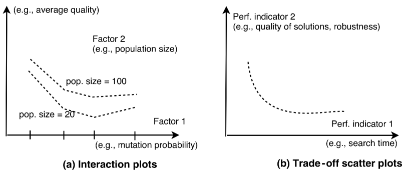


FIGURE 1.29 (a) Interaction plots analyze the effect of two factors (parameters, e.g., mutation probability, population size in evolutionary algorithms) on the obtained results (e.g., solution quality, time). (b) Scatter plots analyze the trade-off between the different performance indicators (e.g., quality of solutions, search time, robustness).

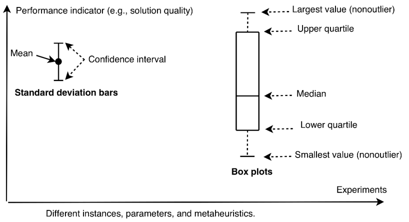
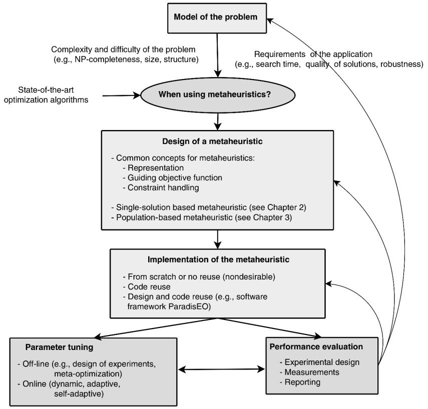

Performance indicator (e.g., solution quality)
Mean
Confidence interval
Standard deviation bars
Largest value (nonoutlier)
Upper quartile
Median
Lower quartile
Smallest value (nonoutlier)
Box plots
Experiments
Different instances, parameters, and metaheuristics.

FIGURE 1.30 Some well-known visualization tools to report results: deviation bars, confidence intervals.

1.9 Conclusions



Model of the problem
Complexity and difficulty of the problem
(e.g., NP-completeness, size, structure)
Requirements of the application
(e.g., search time, quality of solutions, robustness)
State-of-the-art
optimization algorithms
When using metaheuristics?
Design of a metaheuristic
- Common concepts for metaheuristics:
- Representation
- Guiding objective function
- Constraint handling
- Single-solution based metaheuristic (see Chapter 2)
- Population-based metaheuristic (see Chapter 3)
Implementation of the metaheuristic
- From scratch or no reuse (nondesirable)
- Code reuse
- Design and code reuse (e.g., software framework ParadisEO)
Parameter tuning
- Off-line (e.g., design of experiments, meta-optimization)
- Online (dynamic, adaptive, self-adaptive)
Performance evaluation
- Experimental design
- Measurements
- Reporting

FIGURE 1.33 Guidelines for solving a given optimization problem.