Selected Topics in Nature-inspired Algorithms

Seminar

Winter 2017

Nico Potyka

Today

- Introduction
- Organization
- Optimization and Metaheuristics
- Local Search

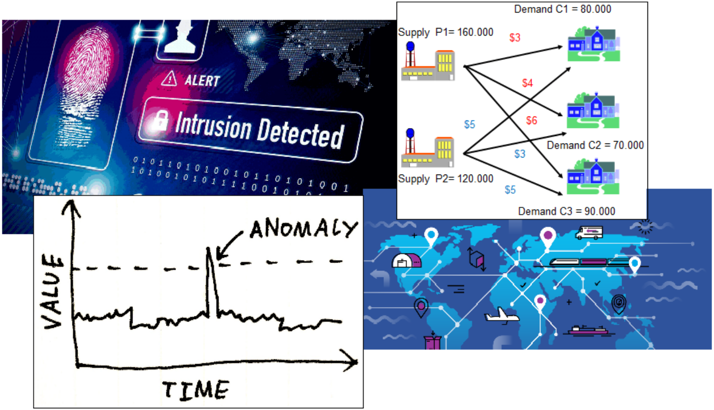
Introduction

Nature-inspired Algorithms

- nature-inspired algorithms take inspiration from problem solving methods inherent to biological systems
- they are often easy to understand conceptually, but can be difficult to tune
- some classical approaches are
  - Genetic Algorithms
  - Differential Evolution
  - Ant Colony Optimization
  - Particle Swarm Optimization

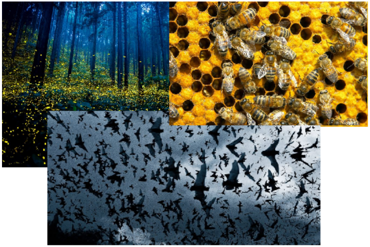
Applications

- successful applications include
  - Discrete and Numerical Optimization (logistics, routing)
  - Constraint Satisfaction (frequency assignment, scheduling)
  - Machine Learning (intrusion detection, anomaly detection)



Development of the Field

- the field keeps on developing new nature-inspired algorithms
- including successful ones
  - Firefly Algorithm
  - Bee Colony Optimization
  - Bat-inspired Algorithms



- and rather exotic ones
  - Reincarnation Algorithm
  - Zombie Survival Optimization
  - Community of Scientists Opt.

A Critical View

- most algorithms come down to a few principles like
  - adaptation
  - natural selection
  - swarm behavior
- lack of novelty is often hidden beneath biological metaphors
- as put in the Evolutionary Computation Bestiary[1], the field is

  [...] as The island of Doctor Moreau: a place with a few good creatures, but which are vastly outnumbered by mindless beasts.

[1] `https://github.com/fcampelo/EC-Bestiary`

On the bright Side

- many nature-inspired algorithms have proven useful for solving diverse problems heuristically
- no free lunch theorems[2] justify diversity of the field
- intuitively, these theorem show that there can be no algorithm that outperforms all other algorithms for every problem

[2] Wolpert, D. H., & Macready, W. G. No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1(1), 67-82: 1997.

Goals of the Course

- our goal is to understand a selection of well established nature-inspired algorithms properly
- the algorithms are usually simple to understand conceptually
- however, they can usually be configured by a significant number of parameters
- tuning these parameters is often art rather than science
- we will therefore have programming tasks where you
    1. implement algorithms
    2. and tune the parameters for a given problem

Organization

Prerequisites

- basic mathematical concepts
- basic computer science concepts
- basic programming skills (can be developed during course)

Organization

Course Structure

- we will alternate theory and practice sessions
- Theory Session: introduction to one framework and subsequent discussion
- Practice Session: presentation and discussion of experiences with implementation and parameter tuning

Theory Session

Preparation for Theory Session (presenting group)

- familiarize yourself with topic
- find additional citable literature if necessary
  (use Google Scholar for instance)
- prepare well-structured, correct and reader-friendly slides
- back up claims with references to the literature
- prepare a few questions for discussion
- presentation length based on group size

Practice Session

Preparation for Practice Session (every group)

- read slides/ basic literature for framework
- implement algorithm for given problem
- make experiments with different parameters
- report your findings in short presentation
- every group member has to present findings for at least one implementation
- every group member must be able to answer questions

Grading

Grading based on

1. presentations
   - reader-friendly slides
   - self-contained, comprehensible presentation
   - well prepared for questions and discussion
2. participation in discussion
   - raise critical questions
   - argue objectively
3. programming exercises
   - systematic parameter tuning
   - clean and well documented code

   (first few programming exercises will have less weight)

Topics and Basic Literature:

- 26/10: Introduction and Basic Local Search Algorithms
- 09/11: Genetic Algorithms

  Sastry, K., Goldberg, D. E., & Kendall, G. Genetic algorithms. In Search methodologies (pp. 93-117). Springer US: 2014.
- 23/11: Ant Colony Optimization

  Merkle, D., & Middendorf, M. Swarm intelligence. In Search methodologies (pp. 213-242). Springer US: 2014.
- 07/12: Differential Evolution

  Das, S., & Suganthan, P. N. Differential evolution: A survey of the state-of-the-art. IEEE transactions on evolutionary computation, 15(1), 4-31: 2011.
- 21/12: Bee Colony Optimization

  Teodorović, D. Bee colony optimization (BCO). Innovations in swarm intelligence, 39-60: 2009.
- 18/01: Particle Swarm Optimization

  Poli, R., Kennedy, J., & Blackwell, T. Particle swarm optimization. Swarm intelligence, 1(1), 33-57: 2007.
- 01/02: Artificial Immune Systems

  Aickelin, U., Dasgupta, D., & Gu, F. Artificial immune systems. In Search Methodologies (pp. 187-211). Springer US: 2014.

Groups and Topics:

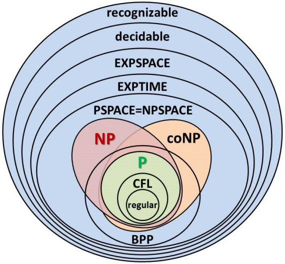
- 6 topics: 6 or 12 groups
  (dependent on number of participants)
- send me your preferred group members and topics until Monday morning
- if you have little programming experience, you can focus on parameter tuning intially and learn during the course

Optimization Problems and Meta-Heuristics

Some interesting Classes of Algorithms

- during the next weeks, we will in particular look at optimization problems
- in this context, nature-inspired algorithms can be regarded as meta-heuristics
- meta-heuristics can be seen as a special class of algorithms
  - Optimal Algorithms: solve problem optimally
  - Approximation Algorithms: solve problem with guaranteed error bound (e.g., at most 20% worse than optimal)
  - Heuristics: solve problem often well in practice, but there are no general guarantees (error can be arbitrarily large)
  - Meta-Heuristics: template heuristics that can be applied to many problems

Why Heuristics?



- many interesting problems cannot be solved efficiently under the usual complexity-theoretical assumptions
- in fact, many cannot even be approximated efficiently
- for many of these problems, heuristics have been developed that 'often' perform well
- meta-heuristics abstract from a concrete problem and can be applied to many problems

Optimization Problems

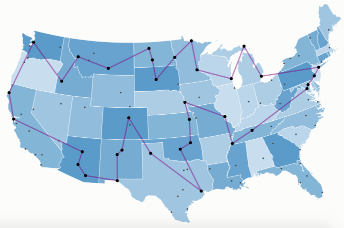
- optimization problems have the following ingredients
  - variables with corresponding domains
  - constraints on values that variables can take (optional)
  - objective function (cost or utility of variable assignments)

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i\in F,\, j\in C} c_{ij}x_{ij} + \sum_{i\in F} f_i y_i & \\
\text{subject to} & \displaystyle\sum_{i\in F} x_{ij} \geq 1, & j \in C \\
 & y_i - x_{ij} \geq 0, & i \in F,\ j \in C \\
 & x_{ij} \in \{0,1\}, & i \in F,\ j \in C \\
 & y_i \in \{0,1\}, & i \in F
\end{array}
$$

$$
\begin{array}{ll}
\min & \displaystyle\sum_{t\in T} \left(\sum_{i\in I} f_i^t y_i^t + \sum_{i\in I}\sum_{j\in J} c_{ij}^t x_{ij} + \sum_{j\in J} h_j^t z_j^t \right), \\
\text{subject to} & \displaystyle\sum_{i\in I} x_{ij}^t + z_j^t = d_j^t, \quad t \in T,\ j \in J, \\
 & \displaystyle\sum_{j\in J} x_{ij}^t \leq q_i y_i^t, \quad t \in T,\ i \in I, \\
 & \displaystyle\sum_{i\in I} y_i^t \leq p^t, \quad t \in T, \\
 & y_i^t \leq y_i^{t+1}, \quad t \in T\backslash\{k\},\ i \in I, \\
 & y_i^t \in \{0,1\}, \quad t \in T,\ i \in I, \\
 & x_{ij}^t \geq 0, \quad t \in T,\ i \in I,\ j \in J \\
 & z_j^t \geq 0, \quad t \in T,\ j \in J.
\end{array}
$$

- examples include
  - loss minimization in ML (unconstrained)
  - routing problems (discrete)
  - logistic problems (discrete or numerical)

Traveling Salesman Problem



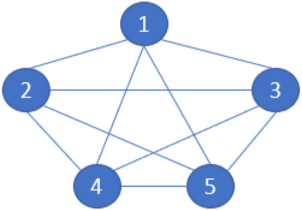
- Problem: given complete weighted graph, compute cheapest cycle that contains all nodes
- Examples
  - find the shortest tour visiting all offices in the US that starts and ends in your own office (minimize travel cost)
  - find the shortest tour visiting all tourist attractions starting and ending in your hotel (minimize walking distance)
  - find the shortest tour visiting all customers starting and ending in your distribution center (minimize time)

Traveling Salesman Problem Formalized

- given completely connected weighted graph (V, E, c) with
  - n nodes $V = \{1, \dots, n\}$
  - edges $E = V \times V$
  - cost function $c : E \to \mathbb{R}_0^+$
- compute shortest cycle starting from 1 and visiting all nodes
- optimization problem
  - Variables: $n - 1$ variables with domain $\{2, \dots, n\}$
    ($x_i = j$ means that we visit node j at time i)
  - no constraints
  - Objective function: cost of tour
    $f(x_1, \dots, x_{n-1}) = c(1, x_1) + c(x_{n-1}, 1) + \sum_{i=1}^{n-2} c(x_i, x_{i+1})$

Example

c(row,col)	1	2	3	4	5
1	0	2	3	4	5
2	2	0	3	2	3
3	3	3	0	1	1
4	4	2	1	0	2
5	5	3	1	2	0



We abbreviate assignment $(x_1 = v_1, \ldots, x_5 = v_5)$ by $(v_1, \ldots, v_5)$

- $f(2,3,4,5) = 2+3+1+2+5 = 13$
- $f(3,4,5,2) = 3+1+2+3+2 = 11$
- $f(5,2,3,4) = 5+3+3+1+4 = 16$

Complexity of TSP

- TSP is NP-complete
- we cannot even find an efficient approximation algorithm if $P \neq NP$, see (Vazirani 2013)[3], Theorem 3.6 for a proof
- if the edge costs satisfy triangle inequality (Metric TSP), approximation guarantee is possible
- however, even for Metric TSP, heuristics 'may' perform better

[3] Vazirani, V. Approximation algorithms. Springer Science & Business Media: 2013.

Local Search Methods

Local Search Methods

- computational problems arise due to size of the search space
- the idea of local search methods is to narrow down the space
- to this end, we define a neighborhood for variable assignments
- local search algorithms basically work as follows:
    1. start with random assignment
    2. replace current assignment with a 'reasonable' neighbor
    3. go back to 2 until no improvement is possible

Choosing a good Neighborhood

- use the whole search space as neighborhood
  1. the found solution will be optimal
  2. however, we do not gain anything computationally
- use empty set as neighborhood
  1. our algorithm will run in constant time
  2. however, the solution is just a random guess
- a good neighborhood balances between runtime and solution quality
- however, usually local search algorithms will find only locally optimal solutions (optimal in local neighborhood)

Swap-Neighborhood for TSP

- Swap-Neighborhood: select one variable and swap value with left neighbor
- each assignment has $n - 1 = \Theta(n)$ neighbors

Example: $(2, 3, 4, 5)$ has neighbors

- $(3, 2, 4, 5)$
- $(2, 4, 3, 5)$
- $(2, 3, 5, 4)$
- $(5, 3, 4, 2)$

Transposition-Neighborhoods for TSP

- Transposition-Neighborhood: swap values of two arbitrary variables
- each assignment has $\binom{n-1}{2} = \frac{(n-1)(n-2)}{2} = \Theta(n^2)$ neighbors

Example: $(2, 3, 4, 5)$ has neighbors

- $(3, 2, 4, 5)$
- $(4, 3, 2, 5)$
- $(5, 3, 4, 2)$
- $(2, 4, 3, 5)$
- $(2, 5, 4, 3)$
- $(2, 3, 5, 4)$

Hill-Climbing

- the easiest local search method is Hill-Climbing
- it chooses the steepest descent neighbor (thus hill-climbing) until no improvement is possible anymore

Hill-Climbing

$A \leftarrow$ *random assignment*

repeat

if neighborhood of A contains better assignment A'

$A \leftarrow$ *best assignment in neighborhood*

else

return A

Demo

Results for Random TSP with 15 nodes $\{0, 1, \ldots, 14\}$

Initial assignment: (1,2,3,4,5,6,7,8,9,10,11,12,13,14)

Hill Climbing with Swap Neighborhood
Found local optimum after 5 iterations.
Cost: 52.0

Hill Climbing with Transposition Neighborhood
Found local optimum after 6 iterations.
Cost: 38.0

Initial assignment: (6,3,4,1,8,5,2,11,14,10,13,7,9,12)

Hill Climbing with Swap Neighborhood
Found local optimum after 6 iterations.
Cost: 47.0

Hill Climbing with Transposition Neighborhood
Found local optimum after 5 iterations.
Cost: 33.0

Observations

- Transposition Neighborhood gives us often better solutions than Swap Neighborhood
- this makes intuitively sense, since it allows to explore the search space better
- on the other hand, each iteration can take significantly more time (quadratic vs. linear)
- indeed, exploring the whole neighborhood can be expensive in large domains

First-Choice Hill-Climbing

- Hill-Climbing can take a lot of time if neighborhoods are large
- First-Choice Hill-Climbing stops iterating neighbors as soon as a better neighbor is found

First-Choice Hill-Climbing

$A \leftarrow$ *random assignment*
repeat
  $A^* \leftarrow A$
  for neighbors A' of A^*
    if A' improves A^*
      $A^* \leftarrow A'$
      skip loop
  if A^* did not change
    return A

Demo

Run	Algorithm	Neighborhood	Iterations	Cost	Time (ms)	Time/Iteration
1	HC	Swap	26	331	15	0.58
	HC	Transp	42	135	123	2.93
	FCHC	Swap	40	337	0	0
	FCHC	Transp	132	140	78	0.59
2	HC	Swap	35	328	0	0
	HC	Transp	35	159	38	1.01
	FCHC	Swap	48	337	15	0.31
	FCHC	Transp	156	145	69	0.44
3	HC	Swap	25	347	0	0
	HC	Transp	39	138	31	0.79
	FCHC	Swap	33	344	0	0
	FCHC	Transp	120	145	54	0.45

Results for Random TSP with 100 nodes, runs started from different initial assignments

Observations

- FCHC can sometimes find better local optima than HC
- the reason is that locally suboptimal choices can lead to better regions in search space
- two further improve exploration of search space, we can
  1. allow suboptimal (or even worse) moves in search space *(e.g. Stochastic Hill-Climbing, Simulated Annealing)*
  2. consider multiple search paths in parallel *(e.g. Parallel Hill-Climbing, Local Beam Search)*

Conclusions

- Local Search methods can be seen as metaheuristics
- choice of neighborhood balances between exploration (large) and efficiency (small)
- one of the fastest local search method is Hill Climbing
- in terms of solution quality, it is often outperformed by other metaheuristics that provide better exploration of search space
- however, Hill Climbing is often used in hybrid algorithms because of its efficiency (e.g. memetic algorithms)

Programming Task

Knapsack Problem

- Problem: given items $I_1, \ldots, I_n$ with weights $w_1, \ldots, w_n$, values $v_1, \ldots, v_n$ and a weight limit W, maximize value
- Examples
  - invest budget in assets with maximum expected return
  - cut sheet metal into pieces of specified size, minimize waste
  - load truck with items of maximal value
  - subproblem (resource allocation) in many other problems

Knapsack Problem Formalized

- given n items $I_1, \dots, I_n$ with
  - weights $w_1, \dots, w_n$
  - values $v_1, \dots, v_n$
  - and a weight limit W
- select items to meet weight limit and maximize value
- optimization problem
  - Variables: n variables with domain $\{0, 1\}$ ($x_i = 1$ iff we select I_i)
  - Constraints: $w(x_1, \dots, x_n) = \sum_{i=1}^{n} w_i \cdot x_i \leq W$
  - Objective function: $f(x_1, \dots, x_n) = \sum_{i=1}^{n} v_i \cdot x_i$

Feasible Solutions

	Pattern 1		Pattern 2				Pattern 3					
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
w	40	40	25	25	25	25	15	15	15	15	15	15
v	20	20	10	10	10	10	5	5	5	5	5	5

- $v_1 = (1,1,1,0,0,0,0,0,0,0,0,0)$, $w(v_1) = 105 > 100$ (infeasible)
- $v_2 = (0,0,1,1,1,1,0,0,0,0,0,0)$, $w(v_2) = 100$, $f(v_2) = 40$ (feasible)
- $v_3 = (1,1,0,0,0,0,0,0,0,0,0,1)$, $w(v_3) = 95$, $f(v_3) = 45$ (feasible)

Knapsack Problem Programming Task

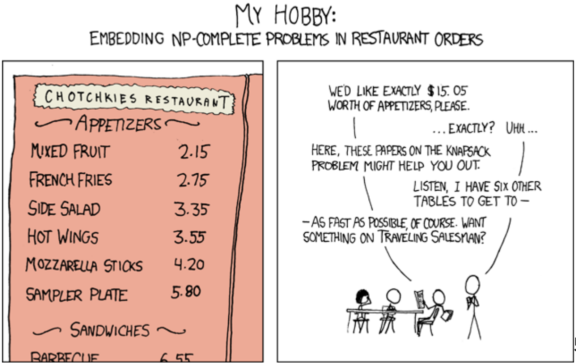
- define a 'small' and a 'large' neighborhood
  (don't forget to assure feasibility)
- implement Hill-Climbing and First-Choice Hill-Climbing
- make experiments with both implementations for some Knapsack instances and different initial assignments
- document your findings and prepare a small presentation
  (5-10 minutes)

Complexity of Knapsack Problem

- Knapsack Problem is NP-complete
- however, as opposed to TSP, it is only weakly NP-hard (there exists a 'pseudo-polynomial' algorithm)
- in particular, it can be approximated efficiently to an arbitrary degree (there exists an 'FPTAS')
- see (Vazirani 2013)[4], Chapter 8 for more details

[4] Vazirani, V. Approximation algorithms. Springer Science & Business Media: 2013.

Questions



[5]

[5] https://imgs.xkcd.com/comics/np_complete.png