Social Inspired Algorithms

Nature Inspired Computing

Introduction

Characteristics

- Inspired in social systems
- Borrows ideas from the organization of swarms, flocks, social psychology

Examples

- Particle Swarm Optimization
- Ant Colony Optimization
- Jaya Algorithm
- Grey Wolf Optimization
- Ant Colony Optimization

Concepts

- Optimization based on social interactions
- Uses population of individuals
- Most algorithms are using for real optimization
- They borrow ideas from the behavior of living systems

Particle Swarm Optimization

- Created in 1995 by James Kennedy and Russell Eberhart (Blum and Merkle 2008)
- Inspired in social psychology and bird flock simulation
- Uses a population of individuals
- Each individual has a position and a velocity
- Velocity is updated by:
  - Atraction to the best position it found in the past
  - Attraction to the best position found by the group

PSO General scheme

1. Initialize population
2. Evaluate individuals
3. For each individual
    1. Choose individuals from neighborhood
    2. Imitate these individuals
    3. Update best performance if a better position was found
4. Iterate to 3 until stopping criterion is found

PSO Algorithm

Individuals' state

Position	Current position $\vec{x}_i$
Velocity	Current velocity $\vec{v}_i$
Individualism	Previously best found position $\vec{p}_i$
Conformism	Previously best found position by the group $\vec{p}_g$

Algorithm

$$\begin{cases} \vec{v}_i = & \chi \left(\vec{v}_i + \vec{\mathrm{U}}[0, \varphi_1] \left(\vec{p}_i - \vec{x}_i\right) + \vec{\mathrm{U}}[0, \varphi_2] \left(\vec{p}_g - \vec{x}_i\right)\right) \\ \vec{x}_i = & \vec{x}_i + \vec{v}_i \end{cases}$$

Initial version

- Initially, the algorithm was proposed with these equations:

$$\begin{cases} \vec{v}_i = & \vec{v}_i + \vec{\mathrm{U}}[0, \varphi_1] \left(\vec{p}_i - \vec{x}_i\right) + \vec{\mathrm{U}}[0, \varphi_2] \left(\vec{p}_g - \vec{x}_i\right) \\ \vec{x}_i = & \vec{x}_i + \vec{v}_i \end{cases}$$

Maximum velocity

- The velocity often becomes very large
- To counter this effect, a new parameter was introduced: V_{max}
- This parameter prevents the velocity from becoming too large
- This parameter is usually coordinate-wise
- If it is too large, individuals fly past good solutions
- If it is too small, individuals explore too slowly and may become trapped in local optima
- Early experience showed that φ_1 and φ_2 could be set to 2 for almost all applications and only V_{max} needed to be adjusted

Innertial Weight

$$\begin{cases} \vec{v}_i = & \alpha\left(\vec{v}_i + \vec{\mathrm{U}}[0,\varphi_1]\left(\vec{p}_i - \vec{x}_i\right) + \vec{\mathrm{U}}[0,\varphi_2]\left(\vec{p}_g - \vec{x}_i\right)\right) \\ \vec{x}_i = & \vec{x}_i + \vec{v}_i \end{cases}$$

- φ_1 and φ_2 were usually set to 2.1
- The *alpha* parameter was uniformly varied between 0.9 and 0.4.

Constriction Coefficient

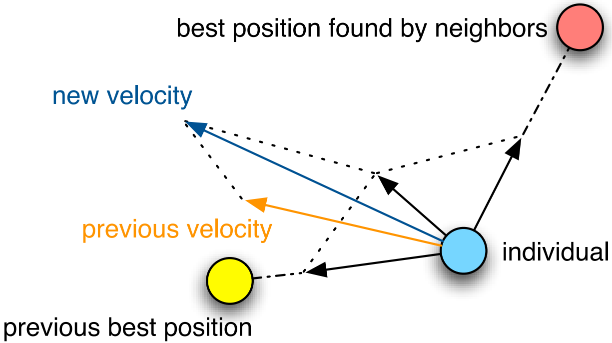
- Clerc proposed a version similar to the innertia weight
- The coefficient has a fixed value instead of a varying one
- The theoretical study seemed to indicate that a setting of all the parameters was enough to guarantee convergence without explosion or oscillation behaviors

Parameters $\varphi_1 = \varphi_2 = 2.05$, $\chi = 0.729$

Algorithm

$$\begin{cases} \vec{v}_i = & \chi \left(\vec{v}_i + \vec{\mathrm{U}}[0, \varphi_1] \left(\vec{p}_i - \vec{x}_i\right) + \vec{\mathrm{U}}[0, \varphi_2] \left(\vec{p}_g - \vec{x}_i\right)\right) \\ \vec{x}_i = & \vec{x}_i + \vec{v}_i \end{cases}$$

Solution Generation in PSO



Premature Convergence in PSO

- The global best individual in the population often degrades PSO performance
- Convergence is fast
- Diversity is lost fast
- This leads to premature convergence
- It is better to use strategies to decrease the information flow

What are the causes of premature convergence?

Optimization is a balance between two factors:

exploration The ability to explore the search space to find promising areas

exploitation The ability to concentrate on the promising areas of the search space

- An algorithm with too much exploration isn't *efficient*
- An algorithm with too much exploitation loses *diversity* fast
- If an algorithm doesn't have enough diversity, it will quickly stagnate

Neighborhood concept

- Individuals imitate their (most successful) neighbors
- His neighbors will only influence their neighbors once they become sufficiently successful
- This favours clustering: different social neighborhoods may explore different areas of the search space
- Immediacy may be based on:

Proximity Proximity in Cartesian space

Social To share social bonds

Example of Good Topologies

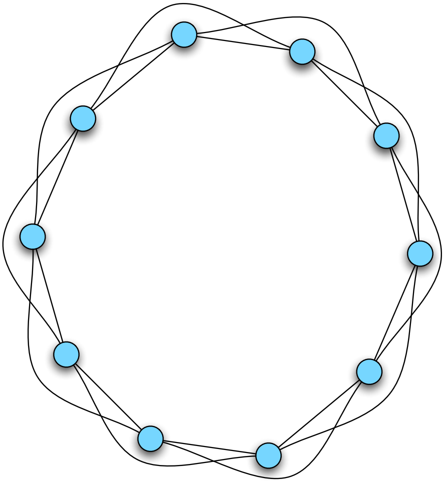


Figure 1: LBest 2

Example of Good Topologies

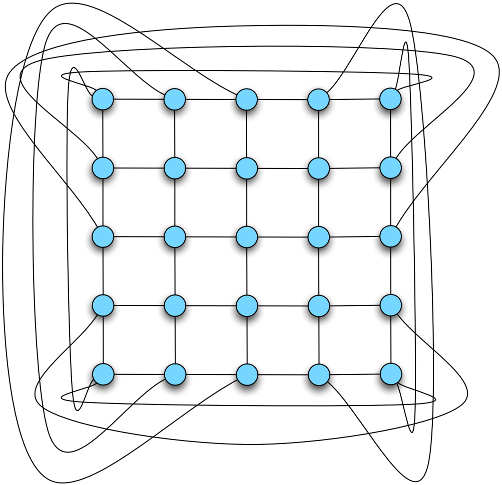


Figure 2: von Neumann or Square

Discussion

- PSO is easy to implement
- The canonical version has a setting for both φ_1 and φ_2 and *chi*
- Thus, the only parameters that need to be changed are the population size and the number of function evaluations
- Any of the population topologies given above works well

Fully Informed Particle Swarm

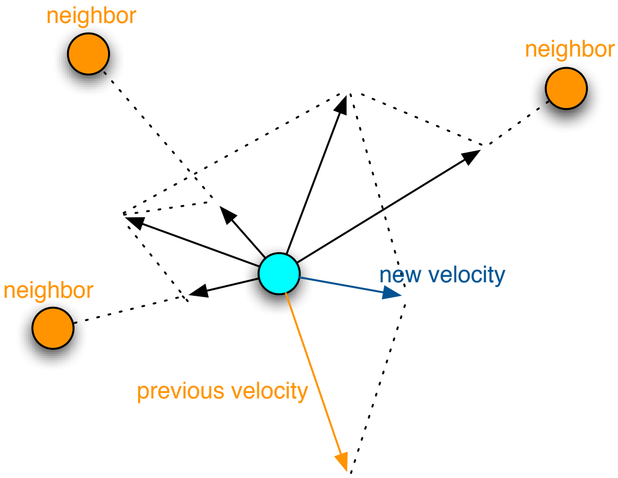
Characteristics

- All contributions of the neighborhood are used
- Individual imitates the social norm
- The social norm is the center of gravity
- $\vec{p}_k$ is the best position of neighbor k
- $\mathcal{N}$ is the set of neighbors

Algorithm

$$\begin{cases} \vec{v}_{t+1} = & \chi\left(\vec{v}_t + \frac{\sum_{k\in\mathcal{N}} \vec{\mathrm{U}}[0,\varphi_{max}](\vec{p}_k - \vec{x}_t)}{|\mathcal{N}|}\right) \\ \vec{x}_{t+1} = & \vec{x}_t + \vec{v}_{t+1} \end{cases}$$

Solution Generation in FIPS



Differences between Canonical PSO and FIPS

- There is no self contribution
- All individuals in the neighborhood contribute to the influence
- The number of individuals used is very important: A few contributions, typically between 2 and 4 are best
- Given a well chosen population topology, it outperforms the canonical model

Discussion

- FIPS is easy to implement
- It often has better performance than the canonical PSO
- Parameter settings are the same as for PSO, and thus are fixed
- Thus, the only parameters that need to be changed are the population size and the number of function evaluations
- Any of the population topologies given above works well

Jaya Optimization

Characteristics

- Similar to PSO
- Fewer control parameters
- Has a component towards the best point found
- Has a component away from the worst point found
- Update is elitist (i.e., new position is only used if it is better than the current one)

Algorithm

$x_i = x_i + \mathrm{U}[0,1]\,(p_i - |x_{best}|) - \mathrm{U}[0,1]\,(p_g - |x_{worst}|)$

Discussion

- Jaya is similar to PSO
- The best and worst performers influence the search
- The only parameters that need to be changed are the population size and the number of function evaluations
- It needs further study to ascertain its performance and if the formulas can be simplified

Grey Wolf Optimization

Characteristics

- Idea is quite similar to FIPS
- Best three solutions (α, β, δ) found guide the search
- New positions are generated by a random combination of the three positions
- The parameter a is linearly decreased from 2 to 0

Algorithm

$$\begin{aligned}
\vec{A}_k &= \vec{\mathrm{U}}[-a, a] && \text{where } k \in \{\alpha, \beta, \delta\} \\
\vec{C}_k &= \vec{\mathrm{U}}[0, 2] && \text{where } k \in \{\alpha, \beta, \delta\} \\
\vec{D}_k &= |\vec{C}_k \otimes \vec{X}_k - \vec{X}| && \text{where } k \in \{\alpha, \beta, \delta\} \\
\vec{P}_k &= \vec{X}_k - \vec{A}_k \otimes \vec{D}_k && \text{where } k \in \{\alpha, \beta, \delta\} \\
\vec{X}_{t+1} &= \frac{\vec{P}_\alpha + \vec{P}_\beta + \vec{P}_\delta}{3}
\end{aligned}$$

Solution Generation in GWO

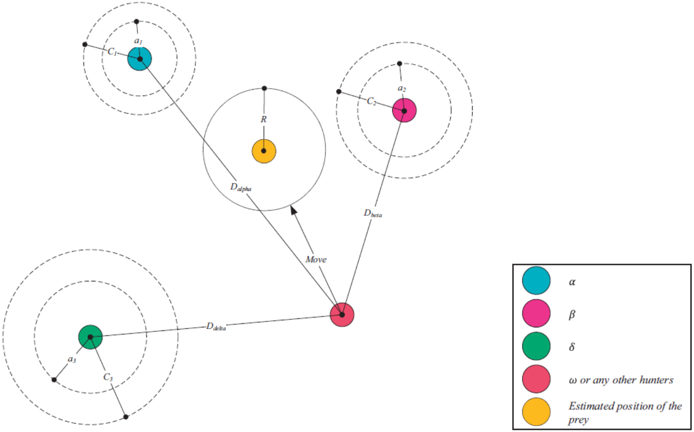


Figure taken from https://doi.org/10.1016/j.advengsoft.2013.12.007

Discussion

- GWO is similar to FIPS
- The 3 best individuals guide the search
- Like FIPS, new solutions are generated by a stochastic barycenter
- The only parameters that need to be changed are the population size and the number of function evaluations
- It needs further study to ascertain its performance and if the formulas can be simplified
- It seems to only work well when the best solution is zero
- There is ongoing research for solving this tendency

Ant Colony Optimization

- Inspired in ant foraging behavior
- Solutions are built by navigating in a graph
- A graph path represents a solution
- A path starts in the initial vertex
- In each vertex, the choice of next edge to visit is *probabilistic*
- The probability of choosing an edge depends on two criteria:
  - attractiveness
  - pheromone levels
- Initially, all graphs have the same pheromone level τ_0
- After each iteration, pheromone is deposited on each edge according to the quality of the solution
- Pheromone gradually evaporates from all edges

Edge choice

Attractiveness Heuristic that determines the *a priori* attractiveness of a given branch

Trail Pheromone quantity deposited in an edge depending on the contribution of that edge in good quality solutions (*a posteriori* contribution)

Probability of choosing an edge

$$p_{uv} = \frac{\tau_{uv}^{\alpha} \cdot \nu_{uv}^{\beta}}{\sum_{w \in \mathrm{viz}(u)} \tau_{uw}^{\alpha} \cdot \nu_{uw}^{\beta}}$$

where:

- τ is the pheromone value deposited in the edge and ν is the heuristic.
- α e β are parameters that weigh the importance of each component.

Pheromone update

Evaporation Each branch evaporates using an evaporation coefficient $0 \leq \rho < 1$

Deposit Each ant deposits pheromone on the edges used in a solution proportionally to the solution quality.

$$\tau_{uv} = (1 - \rho) \cdot \tau_{uv} + \sum_{k=1}^{m} \Delta_{\tau_{uv}}^{k}$$

The value $\Delta_{\tau_{uv}}^{k}$ corresponds to the contribution of edge uv to the solution found by ant k.

$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System

- There is a lower τ_{min} and upper τ_{max} bound for the pheromone
- Only the best ant deposits pheromone

$$\tau_{uv} = (1 - \rho) \cdot \tau_{uv} + \Delta_{\tau_{uv}^{best}}$$

$$\tau_{uv} = \begin{cases} \tau_{uv} & \text{if} \quad \tau_{min} \leq \tau_{uv} \leq \tau_{max} \\ \tau_{min} & \text{if} \quad \tau_{min} > \tau_{uv} \\ \tau_{max} & \text{if} \quad \tau_{max} < \tau_{uv} \end{cases}$$

In the TSP, if D_{best} is the total distance of the best solution:

$$\Delta_{\tau_{uv}^{best}} = \begin{cases} 1/D_{best} & \text{if } (u, v) \text{ belongs to the best path} \\ 0 & \text{otherwise} \end{cases}$$

Advantages of $\mathcal{MM}$AS

- Since the best ant deposits pheromone, there is a *bias* towards higher quality solutions
- In order to control the *bias* the τ_{min} and τ_{max} limits guarantee that:
  - there is always a minimum probability of choosing any edge in the graph
  - the probability of choosing a highly successful edge is never bigger than a given amount

Ant Colony System

- An *elitist* transition is chosen with probability q_0
- The pheromone update method has a *global* and *local* components
- In the *global* component, pheromone is deposited in the edges belonging to the best solution found thus far
- In the *local* component, the pheromone amount of the branches used by the ants is decreased

Ant Colony System: Transition rules

$$v = \begin{cases} \underset{w \in \mathrm{viz}(u)}{\arg\max}\ \tau_{uw}^{\alpha} \cdot \nu_{uw}^{\beta} & \text{if } q \leq q_0 \\ V & \text{if } q > q_0 \end{cases}$$

where $0 \leq q_0 \leq 1$ is a probability, $q = U[0, 1]$ and V is chosen probabilistically according to the rule:

$$p_{uv} = \frac{\tau_{uv}^{\alpha} \cdot \nu_{uv}^{\beta}}{\sum\limits_{w \in \mathrm{viz}(u)} \tau_{uw}^{\alpha} \cdot \nu_{uw}^{\beta}}$$

Ant Colony System: pheromone update

global only for the best solution found thus far

$$\tau_{uv} = (1 - \rho) \cdot \tau_{uv} + \rho \cdot \Delta_{\tau_{uv}^{best}}$$

local for all solutions visited in the current iteration

$$\tau_{uv} = (1 - \rho) \cdot \tau_{uv} + \rho \cdot \tau_0 \backslash [1\ \mathrm{em}]$$

τ_0 is the initial pheromone value of all the edges in the initial iteration

Ant Colony System: Advantages

- The transition method allows the introduction of some *elitism* depending on the value given to q_0
- As in $\mathcal{MM}$AS, pheromone is only increased in the edges belonging to the *best solution* found
- The local update *lowers the attractiveness* of the edges traversed by the ants
- This characteristics fosters exploration

Applications

Routing	Travelling salesman, vehicle routing, sequential ordering
Assignment	Quadratic assignment, timetables, graph coloring
Scheduling	Several scheduling problems
Subsets	Knapsacking, cliques
Bioinformatics	Shortest common supersequence problem, sequencing, protein folding, protein-ligand docking, haplotype inference
Others	Constraint satisfaction, data mining

Using ant colonies in a problem

- Which variant to use and the parameters' values
- How to represent the solution as a graph
- How to compute the solution quality
- What heuristic to use for each edge ν_{uv}
- Is it possible to use the information of the partial solution in the heuristic?
- Is it possible to use local search?

Example: Travelling Salesman Problem

- Each ant starts in the beginning city
- The fitness function is the inverse of the total distance
- The probability of revisiting a city is zero
- The heuristic is the inverse of the distance between both cities

Example: Constraint satisfaction

- There is a set of variables $X_1, \ldots, X_n$
- Each variable has several possible values (e.g., $X_1 \in \{x_{11}, \ldots, x_{1m}\}$)
- There are constraints $p_1, \ldots, p_r$ where each constraint is a predicate (e.g., $p_1(X_1, \ldots, X_n)$)

Example: Constraint satisfaction

- Establish an order for the variables
- There is an initial state S_0
- Each vertex of the graph is a variable assigment (e.g., $X_1 = x_{14}$)
- The objective function may be the number of valid constraints
- The heuristic may be the number of valid constraints after this assigment
- A solution to a problem with 3 variables could be $< S_0 > \rightarrow < X_1 = x_{14} > \rightarrow < X_2 = x_{22} > \rightarrow < X_3 = x_{31} >$

Example: Set covering problem

- We have a finite set of elements $A = \{a_1, \ldots, a_n\}$
- And a set of subsets $B = \{B_1, \ldots, B_l\}$ such as $B_i \subseteq A$ that covers A, i.e., $\bigcup_{i=1}^{l} B_i = A$
- Each set B_i has an associated cost c_i
- The objective of the SCP is to find a subset $C \in B$ such as:
  1. $\bigcup_{B_i \in C} B_i = A$
  2. minimize $\sum_{B_i \in C} c_i$

Example: Set covering problem

- Each ant starts with the empty set and adds a set B_i each time
- Only use edges that add at least one a_j that is not in the current solution
- The heuristic of choosing the set B_i may be:
    1. $\frac{1}{c_i}$
    2. $\frac{|B_i|}{c_i}$
    3. $\frac{d_i}{c_i}$ where d_i is the number of elements of B_i that are not yet covered by the current solution
- An ant finishes building the solution when it has covered the entire set A
- Pruning may be performed *a posteriori* by removing superfluos sets
- Best solutions may use local search

Links

- Software: https://github.com/thieunguyen5991/metaheuristics

Bibliography I

Blum, Christian, and Daniel Merkle. 2008. "Swarm Intelligence." *Swarm Intelligence in Optimization; Blum, C., Merkle, D., Eds*, 43–85.

Bonabeau, Eric, Marco Dorigo, Directeur de Recherches Du Fnrs Marco, Guy Theraulaz, Guy Théraulaz, and others. 1999. *Swarm Intelligence: From Natural to Artificial Systems*. 1. Oxford university press.

Eberhart, Russell C, Yuhui Shi, and James Kennedy. 2001. *Swarm Intelligence*. Elsevier.

Mendes, Rui, James Kennedy, and José Neves. 2004. "The Fully Informed Particle Swarm: Simpler, Maybe Better." *IEEE Transactions on Evolutionary Computation* 8 (3): 204–10.

Mirjalili, Seyedali, Seyed Mohammad Mirjalili, and Andrew Lewis. 2014. "Grey Wolf Optimizer." *Advances in Engineering Software* 69: 46–61.

Bibliography II

Rao, R. 2016. "Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems." *International Journal of Industrial Engineering Computations* 7 (1): 19–34.