Particle Swarm Optimization: A Universal Optimizer?

AP Engelbrecht

Computational Intelligence Research Group (CIRG)
Department of Computer Science
University of Pretoria
South Africa
engel@cs.up.ac.za
http://cirg.cs.up.ac.za

Instructor

Andries Engelbrecht received the Masters and PhD degrees in Computer Science from the University of Stellenbosch, South Africa, in 1994 and 1999 respectively. He is a Professor in Computer Science at the University of Pretoria, and serves as Head of the department. He also holds the position of South African Research Chair in Artificial Intelligence. His research interests include swarm intelligence, evolutionary computation, artificial neural networks, artificial immune systems, and the application of these Computational Intelligence paradigms to data mining, games, bioinformatics, finance, and difficult optimization problems. He is author of two books, Computational Intelligence: An Introduction and Fundamentals of Computational Swarm Intelligence. He holds an A-rating from the South African National Research Foundation.

Presentation Outline I

Introduction

Original PSO has been developed to solve optimization problems that are

- unconstrained/boundary constrained
- static
- single-objective
- continuous-valued

However:

- Can PSO be used to solve optimization problems of different classes, without significantly changing the principles of the basic PSO?
- If this is the case, we say that PSO is a universal optimizer
- For each problem class, what are the issues, and how can PSO be adapted to address these issues, while still maintaining the behavioral principles of PSO?

Goal:

- To show that PSO is a universal optimizer
- Not to present a review of the best possible approaches to solve optimization problems of the different problem classes, but to show that PSO can solve these problems
- Focus is on simple, efficient approaches

Optimization Problem Classes

A number of different optimization problem classes can be identified

- Unconstrained
- Boundary constrained
- Constrained
- Multi-objective, many-objective
- Multi-modal
- Dynamic and noisy
- Continuous-valued versus discrete-valued
- Large scale problems

What are the main components?

- a swarm of particles
- each particle represents a candidate solution
- elements of a particle represent parameters to be optimized

The search process:

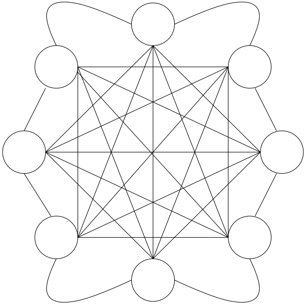
- Position updates

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1), \quad \mathbf{x}_{ij}(0) \sim U(x_{min,j}, x_{max,j})$$

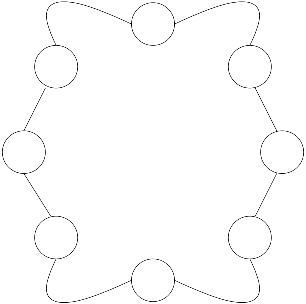
- Velocity (step size)
  - drives the optimization process
  - step size
  - reflects experiential knowledge and socially exchanged information

CIRG

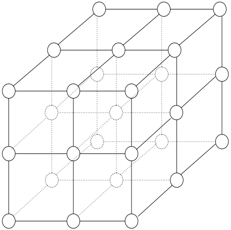
Social network structures are used to determine best positions/attractors



: Star Topology



: Ring Topology



: Von Neumann Topology

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- uses the star social network
- velocity update per dimension:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

- $v_{ij}(0) = 0$ (preferred)
- c_1, c_2 are positive acceleration coefficients
- $r_{1j}(t), r_{2j}(t) \sim U(0,1)$
- note that a random number is sampled for each dimension

- $\mathbf{y}_i(t)$ is the personal best position calculated as (assuming minimization)

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases}$$

- $\hat{\mathbf{y}}(t)$ is the global best position calculated as

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \ldots, \mathbf{y}_{n_s}(t)\} | f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), \ldots, f(\mathbf{y}_{n_s}(t))\}$$

or (removing memory of best positions)

$$\hat{\mathbf{y}}(t) = \min\{f(\mathbf{x}_0(t)), \ldots, f(\mathbf{x}_{n_s}(t))\}$$

where n_s is the number of particles in the swarm

- uses the ring social network

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)]$$

- $\hat{\mathbf{y}}_i$ is the neighborhood best, defined as

$$\hat{\mathbf{y}}_i(t+1) \in \{\mathcal{N}_i | f(\hat{\mathbf{y}}_i(t+1)) = \min\{f(\mathbf{x})\}, \ \forall \mathbf{x} \in \mathcal{N}_i\}$$

  with the neighborhood defined as

$$\mathcal{N}_i = \{\mathbf{y}_{i-n_{\mathcal{N}_i}}(t), \mathbf{y}_{i-n_{\mathcal{N}_i}+1}(t), \ldots, \mathbf{y}_{i-1}(t), \mathbf{y}_i(t), \mathbf{y}_{i+1}(t), \ldots, \mathbf{y}_{i+n_{\mathcal{N}_i}}(t)\}$$

  where $n_{\mathcal{N}_i}$ is the neighborhood size
- neighborhoods based on particle indices, not spatial information
- neighborhoods overlap to facilitate information exchange

- previous velocity, $\mathbf{v}_i(t)$
  - inertia component
  - memory of previous flight direction
  - prevents particle from drastically changing direction
- cognitive component, $c_1\mathbf{r}_1(\mathbf{y}_i - \mathbf{x}_i)$
  - quantifies performance relative to past performances
  - memory of previous best position
  - nostalgia
- social component, $c_2\mathbf{r}_2(\hat{\mathbf{y}}_i - \mathbf{x}_i)$
  - quantifies performance relative to neighbors
  - envy

Synchronous Iteration Strategy

```
Create and initialize the swarm;
repeat
    for each particle do
        Evaluate particle's fitness;
        Update particle's personal best position;
        Update particle's neighborhood best position;
    end
    for each particle do
        Update particle's velocity;
        Update particle's position;
    end
until stopping condition is true;
```

Asynchronous Iteration Strategy

```
Create and initialize the swarm;
repeat
    for each particle do
        Update the particle's velocity;
        Update the particle's position;
        Evaluate particle's fitness;
        Update the particle's personal best position;
        Update the particle's neighborhood best position;
    end
until stopping condition is true;
```

What is the problem?

- PSO was originally developed for optimizing continuous-valued variables
- That is $x_{ij} \in \mathbb{R}$
- Uses vector algebra on floating-point vectors to adjust search positions

How do we adapt PSO so that $x_{ij} \in \{0, 1\}$?

Adapting PSO for binary-valued variables: Binary PSO

- Velocity remains a floating-point vector, but meaning changes
- Velocity is no longer a step size, but is used to determine a probability of selecting bit 0 or bit 1
- Position is a bit vector, i.e. $x_{ij} \in \{0, 1\}$
- How to interpret velocity as a probability?

$$p_{ij}(t) = \frac{1}{1 + e^{-v_{ij}(t)}}$$

- Then, position update changes to

$$x_{ij}(t+1) = \begin{cases} 1 & \text{if } U(0,1) < p_{ij}(t+1)) \\ 0 & \text{otherwise} \end{cases}$$

Velocity clamping:

- sets the minimal probability for a bit change
- if $V_{max,j} = 4$, then $\text{sig}(V_{max,j}) = 0.982$ is the probability of x_{ij} to change to bit 1, and 0.018 the probability to change to bit 0
- small values for $V_{max,j}$ promotes exploration
- for $V_{max,j} = 0$, the search changes to a random search
- large values for $V_{max,j}$ promotes exploitation
- start with small $V_{max,j}$ that increases over time

Inertia weight:

- $w < 1$ works against convergence, as v_{ij} becomes zero over time, and each bit then has a 50% change of changing
- velocity should not become zero
- start with small w, increase over time

Velocity initialization: Initialize to zero.

- for $v_{ij} > 1$, $\lim_{t\to\infty} \text{sig}(v_{ij}(t)) \to 1$ and the probability that all bits change to 1 increases
- for $v_{ij} < -1$, $\lim_{t\to\infty} \text{sig}(v_{ij}(t)) \to 0$ and the probability that all bits change to 0 increases

Some issues with the binary PSO:

- Changes the meaning of the velocity update
  - No longer a step size
  - No longer a search trajectory
- Effect of control parameters change
- Theoretical analysis of standard PSO no longer applies

An approach to solve a $\mathbb{B}^{n_x}$-dimensional problem in $\mathbb{R}^4$

- Velocities and particle positions remain floating-point vectors
- Find a bitstring generating function, used to generate the bitstring solution
- The generating function:

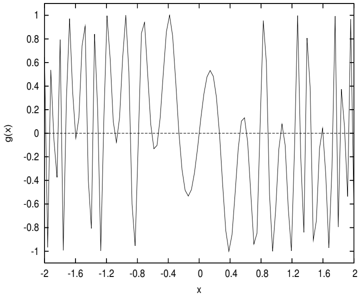
$$g(x) = \sin(2\pi(x - a) \times b \times \cos(2\pi(x - a) \times c)) + d$$

where x is a single element from a set of evenly separated intervals determined by the required number of bits that need to be generated

$$g(x) = \sin(2\pi(x - a) \times b \times \cos(2\pi(x - a) \times c)) + d$$

The coefficients determine the shape of the generating function:

- *a*: horizontal shift of generating function
- *b*: maximum frequency of the sin function
- *c*: frequency of the cos function
- *d*: vertical shift of generating function



Use a standard PSO to find the best values for these coefficients

Generate a swarm of 4-dimensional particles;
repeat
  Apply any PSO for one iteration;
  for *each particle* **do**
    Substitute values for coefficients a, b, c and d into generating function;
    Produce n_x bit-values to form a bit-vector solution;
    Calculate the fitness of the bit-vector solution in the original bit-valued space;
  end
until *a convergence criterion is satisfied*;

Assuming minimization,

Boundary constrained optimization problem:

$$\begin{array}{rll} \text{minimize} & f(\mathbf{x}), & \mathbf{x} = (x_1, x_2, \ldots, x_{n_x}) \\ \text{subject to} & & x_j \in \text{dom}(x_j) \end{array}$$

where $\mathbf{x} \in \mathcal{F} = \mathcal{S}$, and $\text{dom}(x_j)$ is the domain of variable x_j.

Multi-solution problem: Find a set of solutions,

$$\mathcal{X} = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \ldots, \mathbf{x}_{n_\mathcal{X}}^*\}$$

such that each $\mathbf{x}^* \in \mathcal{X}$ is a minimum of the general optimization problem

Niching capability of PSO:

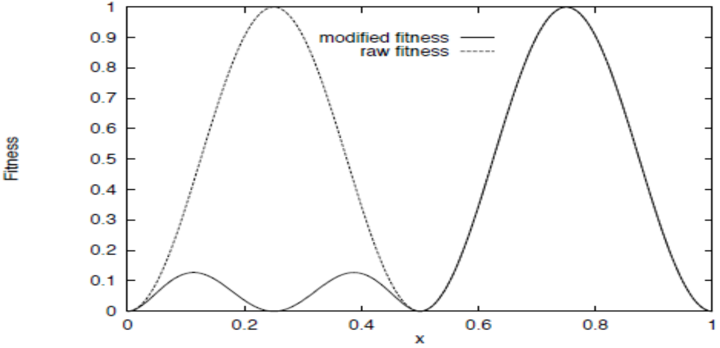
- Can the *gbest* PSO find more than one solution?
  - Formal proofs showed that all particles converge to a weighted average of their personal best and global best positions

$$\lim_{t\to\infty} \mathbf{x}_i(t) = \frac{c_1\mathbf{y}_i + c_2\hat{\mathbf{y}}}{c_1 + c_2}$$

  - Therefore, only one solution can be found
  - What if we re-run the algorithm? No guarantee to find different solutions
- What about *lbest* PSO?
  - Neighborhoods may converge to different solutions
  - However, due to overlapping neighborhoods, particles are still attracted to one solution
  - Formal proof exist to show that all particles converge in the limit

Sequential niching, stretching the function to remove found minima

Create and initialize a n_x-dimensional swarm, S, and $\mathcal{X} = \emptyset$;
repeat
  Perform a single PSO iteration;
  if $f(S.\hat{\mathbf{y}}) \leq \epsilon$ **then**
    Isolate $S.\hat{\mathbf{y}}$;
    Perform a local search around $S.\hat{\mathbf{y}}$;
    if *a minimizer* $\mathbf{x}^*_{\mathcal{N}}$ *is found by the local search* **then**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{x}^*_{\mathcal{N}}\}$;
      Let $f(\mathbf{x}) \leftarrow H(\mathbf{x})$;
    end
  end
  Reinitialize the swarm S;
until *stopping condition is true*;
Return $\mathcal{X}$ as the set of multiple solutions;



: Effect of Sequential Niching for One Dimension

Parallel niching PSO

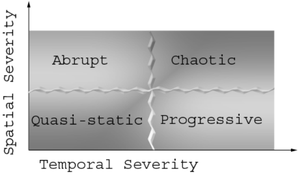
Create and initialize a n_x-dimensional *main* swarm, S;
repeat
  Train main swarm, S, for one iteration using *cognition-only* model;
  Update the fitness of each main swarm particle, $S.\mathbf{x}_i$;
  for *each sub-swarm S_k* **do**
    Train sub-swarm particles, $S_k.\mathbf{x}_i$, using a full model PSO;
    Update each particle's fitness;
    Update the swarm radius $S_k.R$;
  endFor
  If possible, merge sub-swarms;
  Allow sub-swarms to absorb any particles from the main swarm that moved into the sub-swarm;
  If possible, create new sub-swarms;
until *stopping condition is true*;
Return $S_k.\hat{\mathbf{y}}$ for each sub-swarm S_k as a solution;

- Objective: To find and track solutions in dynamically changing search spaces

$$\mathbf{x}^*(t) = \min_{\mathbf{x}} f(\mathbf{x}, \varpi(t))$$

where $\mathbf{x}^*(t)$ is the optimum found at time step t, and $\varpi(t)$ is a vector of time-dependent objective function control parameters

- Environment types:
  - Location of optima may change
  - Value of optima may change
  - Optima may disappear and new ones appear
  - Change frequencey
  - Change severity



: Environment Classes

Can PSO be applied to track an optimum?

- Only for quasi-static environments, to some success

What are the problems?

- Loss of diversity
- Memory
- Change detection

What can be done to address these problems?

- Diversity
  - Inject diversity into the swarm, but how much, and how?
  - Maintain diversity
- Memory
  - Re-evaluate personal best and neighborhood best positions
- Change detection
  - Use sentry particles

Maintains diversity throughout the search process

- Some particles attract one another, and others repell one another
- Velocity changes to

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_2(t)[\hat{y}_j(t) - x_{ij}(t)] + a_{ij}(t)$$

where $\mathbf{a}_i$ is the particle acceleration, determining the magnitude of inter-particle repulsion

$$\mathbf{a}_i(t) = \sum_{l=1, i \neq l}^{n_s} \mathbf{a}_{il}(t)$$

- The repulsion force between particles i and l is

$$\mathbf{a}_{il}(t) = \begin{cases} \left(\frac{Q_i Q_l}{d_{il}^3}\right)(\mathbf{x}_i(t) - \mathbf{x}_l(t)) & \text{if } R_c \leq d_{il} \leq R_p \\ 0 & \text{otherwise} \end{cases}$$

- Based on quantum model of an atom, where orbiting electrons are replaced by a quantum cloud which is a probability distribution governing the position of the electron
- Developed as a simplified and less expensive version of the charged PSO
- Swarm contains
  - neutral particles following standard PSO updates
  - charged, or quantum particles, randomly placed within a multi-dimensional sphere

$$\mathbf{x}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + \mathbf{v}_i(t+1) & \text{if } Q_i = 0 \\ \mathbf{B}_{\hat{\mathbf{y}}}(r_{cloud}) & \text{if } Q_i \neq 0 \end{cases}$$

  - charged particles uniformly sampled within the sphere

Can use different distributions:

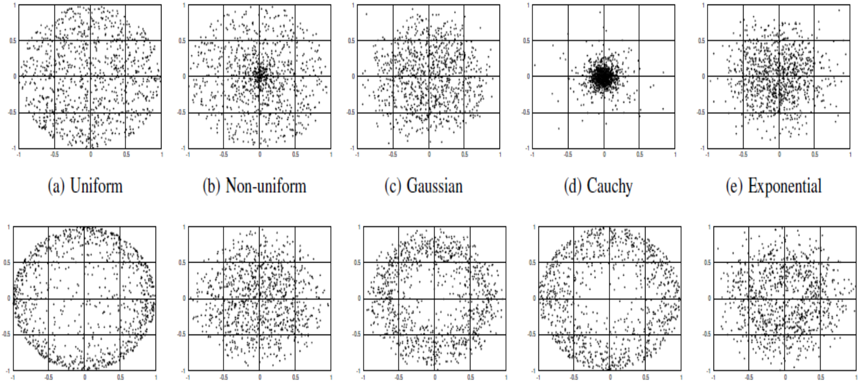
$$\mathbf{x}_i(t+1) \sim P(\hat{\mathbf{y}}(t), r_{cloud})$$

where P is some probability distribution and r_{cloud} is the quantum radius

Some alternative distributions to consider:

- Non-uniform (decreasing probability)
- Gaussian
- Cauchy
- Exponential
- Beta
- Triangular
- Weibull

Best distribution depends on type of dynamism



(a) Uniform (b) Non-uniform (c) Gaussian (d) Cauchy (e) Exponential

(f) Beta (g) Triangular-0 (h) Triangular-0.5 (i) Triangular-1 (j) Weibull

Constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}), \quad \mathbf{x} = (x_1, \ldots, x_{n_x}) \\ \text{subject to} \quad & g_m(\mathbf{x}) \leq 0, \; m = 1, \ldots, n_g \\ & h_m(\mathbf{x}) = 0, \; m = n_g + 1, \ldots, n_g + n_h \\ & x_j \in \text{dom}(x_j) \end{aligned}$$

where n_g and n_h are the number of inequality and equality constraints respectively

- How do we ensure that only feasible solutions are found?
- Boundary versus functional constraints
- For boundary constraints:
  - Do not allow particles that violate boundary constraints to become personal best positions
  - Reinitialize those elements that violate the boundary constraints within the bounds
- Reject infeasible solutions
  - Do not allow infeasible particles to become personal best or neighborhood best positions
  - Replace infeasible solutions with randomly generated, feasible solutions

Optimization problem is reformulated as

$$\text{minimize} \quad F(\mathbf{x}, t) = f(\mathbf{x}, t) + \lambda p(\mathbf{x}, t)$$

λ is the penalty coefficien, and $p(\mathbf{x}, t)$ is the (possibly) time-dependent penalty function

- How to find the best penalty coefficients?
- And the penalty?

$$p(\mathbf{x}_i, t) = \sum_{m=1}^{n_g+n_h} \lambda_m(t) p_m(\mathbf{x}_i)$$

where

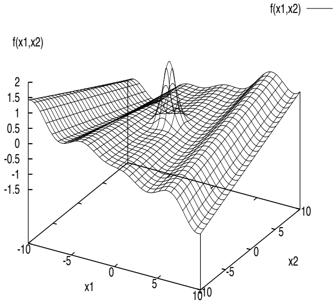
$$p_m(\mathbf{x}_i) = \begin{cases} \max\{0, g_m(\mathbf{x}_i)^\alpha\} & \text{if } m \in [1, \ldots, n_g] \\ |h_m(\mathbf{x}_i)|^\alpha & \text{if } m \in [n_g + 1, \ldots, n_g + n_h] \end{cases}$$

α is a positive constant, representing the power of the penalty

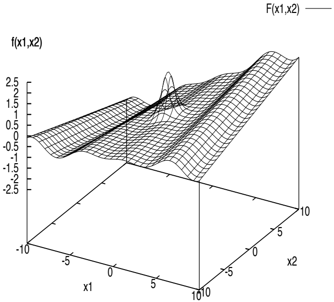
Constrained Optimization Problems

Penalty Methods (cont)

$$f(x_1, x_2) = \frac{x_1 \cos(x_1)}{20} + 2e^{-x_1^2-(x_2-1)^2} + 0.01x_1x_2$$



(a) Original function



(b) With penalty $p(x_1, x_2) = 3x_1$ and $\lambda = 0.05$

Constrained problem 1: Minimize the function

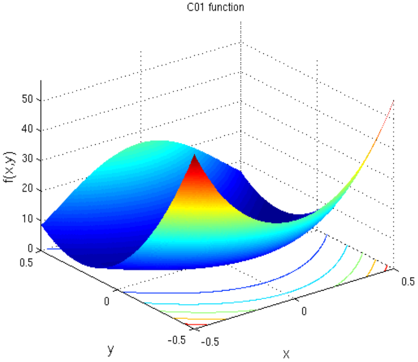
$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

subject to the nonlinear constraints,

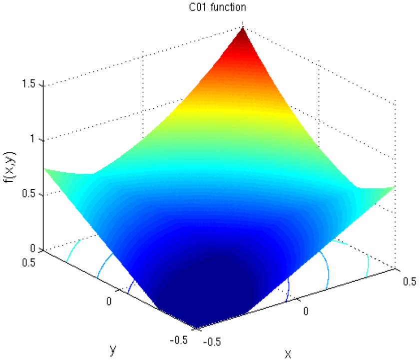
$$x_1 + x_2^2 \geq 0$$
$$x_1^2 + x_2 \geq 0$$

with $x_1 \in [-0.5, 0.5]$ and $x_2 \leq 1.0$.

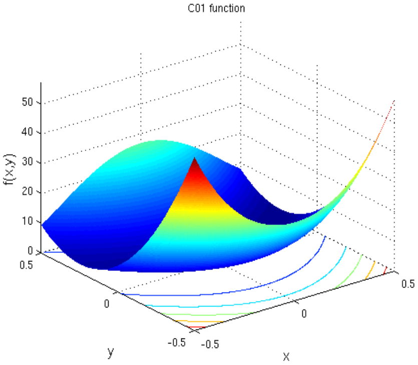
The global optimum is $\mathbf{x}^* = (0.5, 0.25)$, with $f(\mathbf{x}^*) = 0.25$



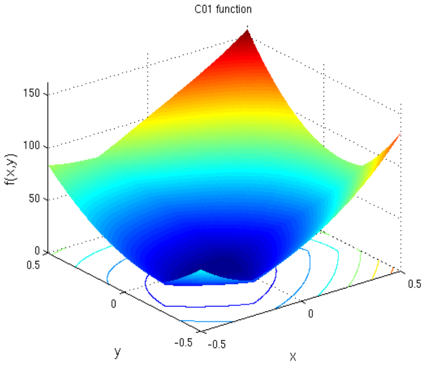
: Function Landscape



: Violation Space



: With $\lambda = 1$



: With $\lambda = 100$

Convert the constrained (primal) problem to an unconstrained problem by defining the Lagrangian for the constrained problem:

$$L(\mathbf{x}, \lambda_g, \lambda_h) = f(\mathbf{x}) + \sum_{m=1}^{n_g} \lambda_{gm} g_m(\mathbf{x}) + \sum_{m=n_g+1}^{n_g+n_h} \lambda_{hm} h_m(\mathbf{x})$$

Then maximize the Lagrangian (dual problem):

$$\begin{array}{ll} \text{maximize}_{\lambda_g, \lambda_h} & L(\mathbf{x}, \lambda_g, \lambda_h) \\ \text{subject to} & \lambda_{gm} \geq 0, \quad m = 1, \ldots, n_g + n_h \end{array}$$

The vector $\mathbf{x}^*$ that solves the primal problem, as well as the Lagrange multiplier vectors, λ_g^* and λ_h^*, can be found by solving the min-max problem,

$$\min_{\mathbf{x}} \max_{\lambda_g, \lambda_h} L(\mathbf{x}, \lambda_g, \lambda_h)$$

A coevolutionary PSO approach to solve the above min-max problem uses two swarms

- Swarm S_1 uses fitness function

$$f(\mathbf{x}) = \max_{\lambda_g, \lambda_h \in S_2} L(\mathbf{x}, \lambda_g, \lambda_h)$$

- Swarm S_2 uses fitness function

$$f(\lambda_g, \lambda_h) = \min_{\mathbf{x} \in S_1} L(\mathbf{x}, \lambda_g, \lambda_h)$$

Create and initialize two swarms, S_1 and S_2, where S_1 is n_x-dimensional and S_2 is $n_g + n_h$ dimensional;

repeat

  Run a PSO algorithm on swarm S_1 for $S_1.n_t$ iterations;

  Re-evaluate $S_2.\mathbf{y}_i(t), \forall i = 1, \ldots, S_2.n_s$;

  Run a PSO algorithm on swarm S_2 for $S_2.n_t$ iterations;

  Re-evaluate $S_1.\mathbf{y}_i(t), \forall i = 1, \ldots, S_1.n_s$;

until *stopping condition is true*;

Reformulate as a boundary constrained multi-objective optimization problem:

$$\mathbf{f}(\mathbf{x}) = (f(\mathbf{x}), p(\mathbf{x}))$$

Solve using any multi-objective PSO algorithm

Multi-objective problem:

$$\begin{array}{ll} \text{minimize} & \mathbf{f}(\mathbf{x}) \\ \text{subject to} & g_m(\mathbf{x}) \leq 0, \quad m = 1, \ldots, n_g \\ & h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \ldots, n_g + n_h \\ & \mathbf{x} \in [\mathbf{x}_{min}, \mathbf{x}_{max}]^{n_x} \end{array}$$

where $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_{n_k}(\mathbf{x})) \in \mathcal{O} \subseteq \mathbb{R}^{n_k}$

$\mathcal{O}$ is referred to as the *objective space*

The search space, $\mathcal{S}$, is also referred to as the *decision space*

Important things to note:

- Goals are in conflict with one another
- Need to achieve a balance between these objectives
- A balance is achieved when a solution cannot improve any objective without degrading one or more of the other objectives
- There is not just one solution
- Solutions are referred to as *non-dominated solutions*
- Set of solutions is referred to as the Pareto-optimal set, and the corresponding objective vectors are referred to as the Pareto front

Definition:

$$
\begin{array}{ll}
\text{minimize} & \sum_{k=1}^{n_k} \omega_k f_k(\mathbf{x}) \\
\text{subject to} & g_m(\mathbf{x}) \leq 0, \quad m = 1, \ldots, n_g \\
& h_m(\mathbf{x}) = 0, \quad m = n_g + 1, \ldots, n_g + n_h \\
& \mathbf{x} \in [\mathbf{x}_{min}, \mathbf{x}_{max}]^{n_x} \\
& \omega_k \geq 0, k = 1, \ldots, n_k
\end{array}
$$

It is also usually assumed that $\sum_{k=1}^{n_k} \omega_k = 1$

Aggregation methods have the following problems:

- The algorithm has to be applied repeatedly to find different solutions if a single-solution algorithm is used
- It is difficult to get the best weight values, ω_k, since these are problem-dependent
- Aggregation methods can only be applied to generate members of the Pareto-optimal set when the Pareto front is concave, regardless of the values of ω_k

Domination: *A decision vector,* $\mathbf{x}_1$ *dominates a decision vector,* $\mathbf{x}_2$ *(denoted by* $\mathbf{x}_1 \prec \mathbf{x}_2$*), if and only if*

- $\mathbf{x}_1$ *is not worse than* $\mathbf{x}_2$ *in all objectives, i.e.* $f_k(\mathbf{x}_1) \leq f_k(\mathbf{x}_2), \forall k = 1, \ldots, n_k$*, and*
- $\mathbf{x}_1$ *is strictly better than* $\mathbf{x}_2$ *in at least one objective, i.e.* $\exists k = 1, \ldots, n_k : f_k(\mathbf{x}_1) < f_k(\mathbf{x}_2)$*.*

So, solution $\mathbf{x}_1$ is better than solution $\mathbf{x}_2$ if $\mathbf{x}_1 \prec \mathbf{x}_2$ (i.e. $\mathbf{x}_1$ dominates $\mathbf{x}_2$), which happens when $\mathbf{f}_1 \prec \mathbf{f}_2$

Pareto-optimal: *A decision vector,* $\mathbf{x}^* \in \mathcal{F}$ *is Pareto-optimal if there does not exist a decision vector,* $\mathbf{x} \neq \mathbf{x}^* \in \mathcal{F}$ *that dominates it. That is,* $\nexists k : f_k(\mathbf{x}) < f_k(\mathbf{x}^*)$. *An objective vector,* $\mathbf{f}^*(\mathbf{x})$, *is Pareto-optimal if* $\mathbf{x}$ *is Pareto-optimal.*

Pareto-optimal set: *The set of all Pareto-optimal decision vectors form the Pareto-optimal set,* $\mathcal{P}^*$. *That is,*

$$\mathcal{P}^* = \{\mathbf{x}^* \in \mathcal{F} | \nexists \mathbf{x} \in \mathcal{F} : \mathbf{x} \prec \mathbf{x}^*\}$$

Pareto-optimal front: *Given the objective vector,* $\mathbf{f}(\mathbf{x})$, *and the Pareto-optimal solution set,* $\mathcal{P}^*$, *then the Pareto-optimal front,* $\mathcal{PF}^* \subseteq \mathcal{O}$, *is defined as*

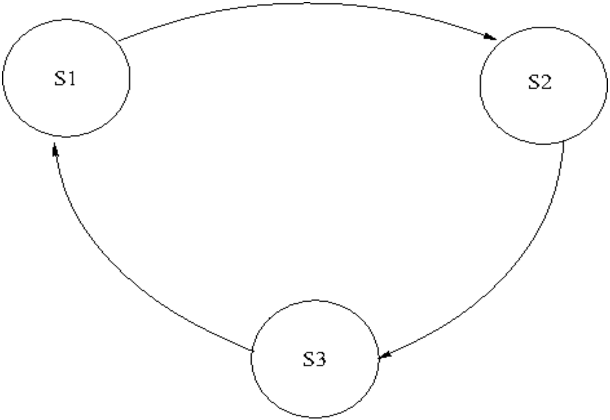
$$\mathcal{PF}^* = \{\mathbf{f} = (f_1(\mathbf{x}^*), f_2(\mathbf{x}^*), \ldots, f_k(\mathbf{x}^*)) | \mathbf{x}^* \in \mathcal{P}\}$$

A multi-swarm approach:

- Assume K sub-objectives
- K sub-swarms are used, where each optimizes one of the objectives
- Need a knowledge transfer strategy (KTS) to transfer information about best positions between sub-swarms
- Exchanged information are via selection of global guides, replacing the global best positions in the velocity updates
- Standard KTS: the ring KTS
  - Sub-swarms are arranged in a ring topology
  - Global guide of swarm S_k is swarm $S_{(k+1) \mod K}$

Multi-Objective Problems

VEPSO (cont)



Assume two objectives

$$\begin{aligned} S_1.v_{ij}(t+1) &= wS_1.v_{ij}(t) + c_1 r_{1j}(t)(S_1.y_{ij}(t) - S_1.x_{ij}(t)) \\ &+ c_2 r_{2j}(t)(S_2.\hat{y}_i(t) - S_1.x_{ij}(t)) \\ S_2.v_{ij}(t+1) &= wS_2.v_{ij}(t) + c_1 r_{1j}(t)(S_2.y_{ij}(t) - S_2.x_{ij}(t)) \\ &+ c_2 r_{ij}(t)(S_1.\hat{y}_j(t) - S.x_{2j}(t)) \end{aligned}$$

where sub-swarm S_1 evaluates individuals on the basis of objective $f_1(\mathbf{x})$, and sub-swarm S_2 uses objective $f_2(\mathbf{x})$

Local guide selection:

- Local guide replaces the personal best
- Update personal best position only if the new particle position dominates the previous personal best position

Global guide selection:

- Global guide replaces the neighborhood best
- Selection dictated by a knowledge transfer strategy (KTS):
    - Ring KTS
    - Random KTS

Using archives

- Objective of archive is to keep track of all non-dominated solutions
- Non-dominated solutions added to archive after each iteration
- Fixed-sized archives versus unlimited sizes
- Local versus global guides

Let $t = 0$;
Initialize the swarm, $S(t)$, and archive, $A(t)$;
repeat
  Evaluate $(S(t))$;
  $A(t+1) \leftarrow \text{Update}(S(t), A(t))$;
  $S(t+1) \leftarrow \text{Generate}(S(t), A(t))$;
  $t = t + 1$;
until *stopping condition is true*;

Curse of dimensionality:

- As dimensionality increases, performance deteriorates

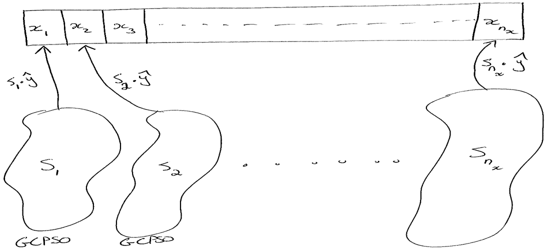
What to do?

- Increase number of particles
  - Increases computational complexity
  - Reduces step sizes, due to smaller difference vectors
- Reduce the complexity of the problem, using divide-and-conquer

Large Scale Optimization

Cooperative PSO (CPSO)

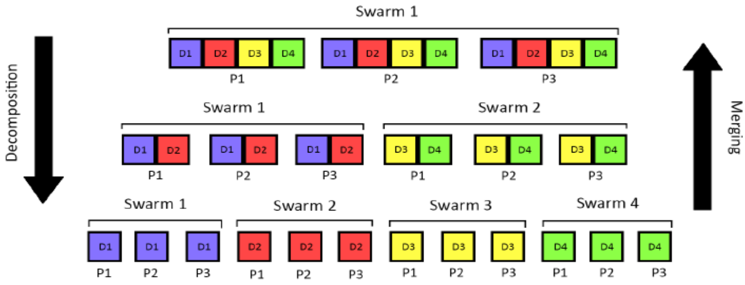
- Each particle is split into K separate parts of smaller dimension
- Each part is then optimized using a separate sub-swarm
- If $K = n_x$, each dimension is optimized by a separate sub-swarm
- What are the issues?
  - Problem if there are strong dependencies among variables
  - How should the fitness of sub-swarm particles be evaluated?



$K_1 = n_x \bmod K$ and $K_2 = K - (n_x \bmod K)$;
Initialize K_1 $\lceil n_x/K \rceil$-dimensional and K_2 $\lfloor n_x/K \rfloor$-dimensional swarms;
repeat
    for *each sub-swarm* $S_k, k = 1, \ldots, K$ **do**
        for *each particle* $i = 1, \ldots, S_k.n_s$ **do**
            if $f(\mathbf{b}(k, S_k.\mathbf{x}_i)) < f(\mathbf{b}(k, S_k.\mathbf{y}_i))$ **then**
                $S_k.\mathbf{y}_i = S_k.\mathbf{x}_i$;
            end
            if $f(\mathbf{b}(k, S_k.\mathbf{y}_i)) < f(\mathbf{b}(k, S_k.\hat{\mathbf{y}}))$ **then**
                $S_k.\hat{\mathbf{y}} = S_k.\mathbf{y}_i$;
            end
        end
        Apply velocity and position updates;
    end
until *stopping condition is true*;

How to cope with variable dependencies?

- Pre-processing to determine correlations and group correlated variables in same sub-swarm
- Random grouping
- Top-down versus bottom-up approaches



Summary

- Particle swarm optimization is an extremely simple, yet powerful optimization method
- Without changing the basic principles of PSO, minor modifications allow PSO to be applied to a wide range of problem classes, including:
    - Unconstrained
    - Constrained
    - Unimodal and multimodal
    - Continuous-valued and discrete-valued
    - Dynamically changing landscapes
    - Multi-objective
    - Multiple solutions
- Various combinations of the above problem types

Summary

- We can therefore safely say that PSO is a universal optimizer
- We do not say that PSO is the best for all classes of problems, and all landscape characteristics, only that it can be applied to solve a wide range of problem classes
- This tutorial is based on the content of the following reference: AP Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, Wiley, 2005.