Swarm Optimization

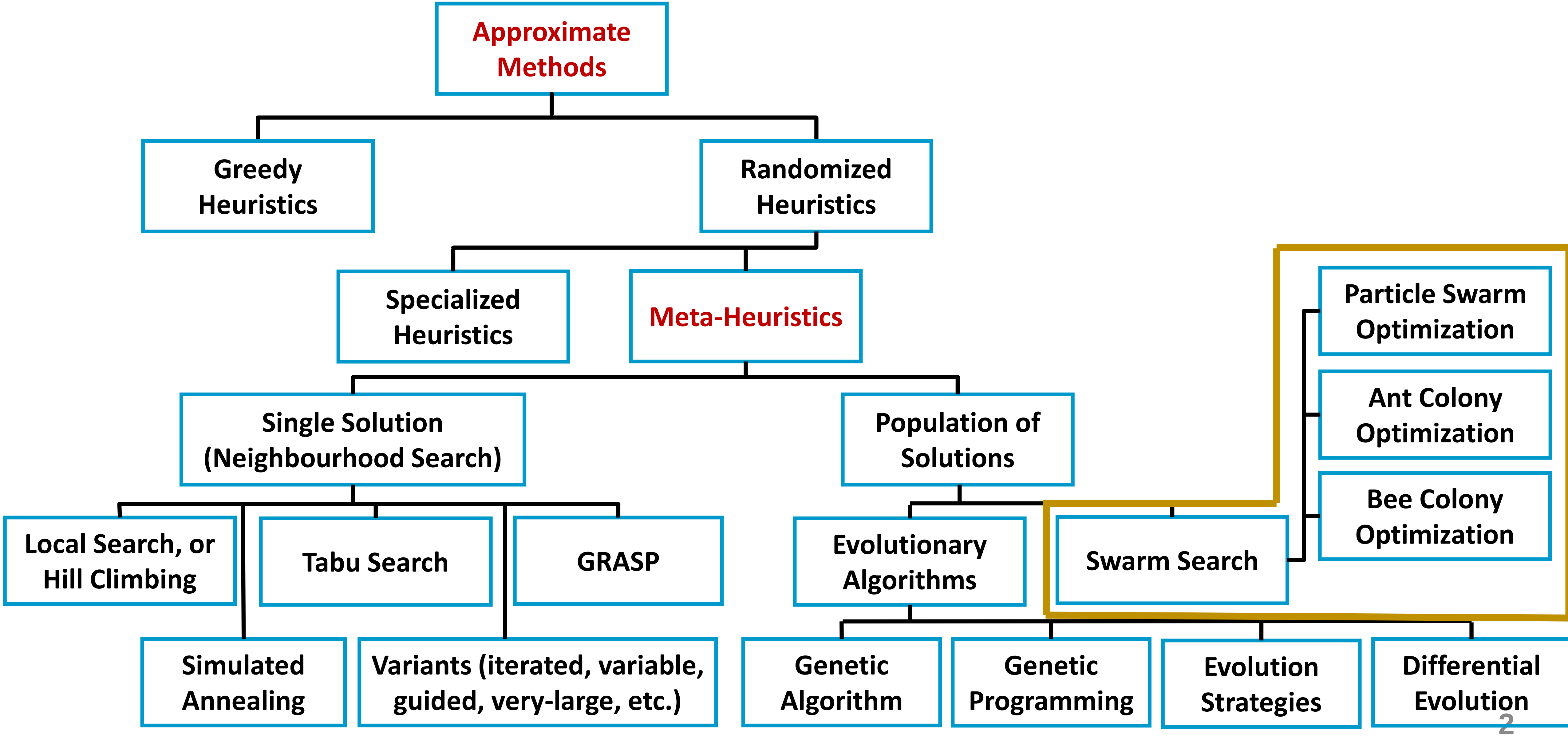
Nuno Antunes Ribeiro

Assistant Professor

Engineering Systems and Design

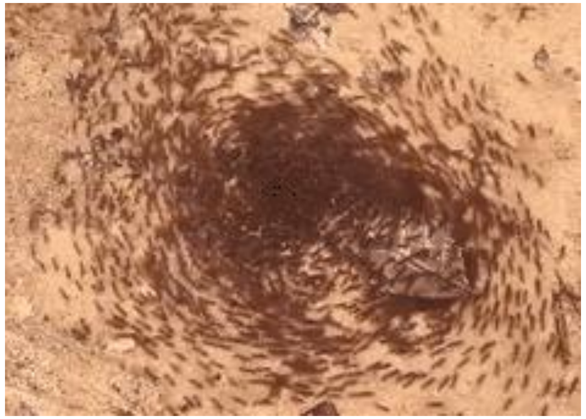
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

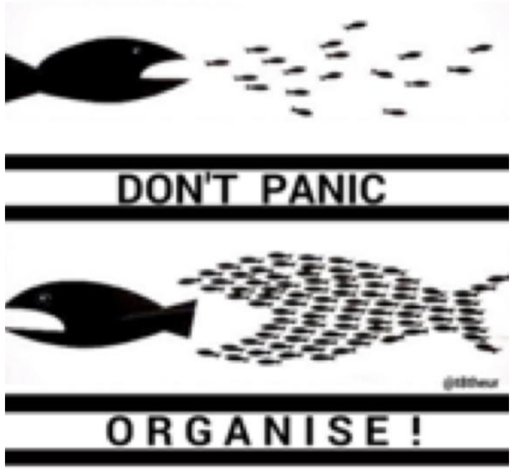
Approximate Optimization Methods



Swarm Behaviour

- **Swarm behavior** is the collective behavior of **decentralized, self-organized** systems. A typical swarm system consists of a **population of simple agents which can communicate** (either directly or indirectly) locally with each other by acting on their local environment.
- Examples in natural systems of swarm intelligence include **bird flocking, ant foraging, and fish schooling**
  - Swarm agents establish a social network
  - Swarm agents profit from the discoveries and previous experience of the other agents of the swarms
  - Swarm agents iteratively change their positions (i.e., decides how to move) using information from personal past experience and from its social neighborhood

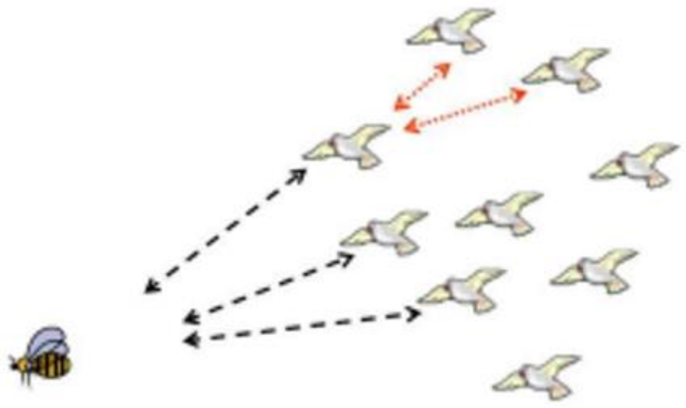




Communication and Cooperation

- Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behavior of birds. The name "boid" corresponds to a shortened version of "bird-oid object", which refers to a bird-like object

- Boids follow 3 fundamental rules are:
  - Birds are **attracted** to the location of the roost
  - Birds **remember** where it was closer to the roost
  - Birds **share information** with its neighbors about its closest location to the roost

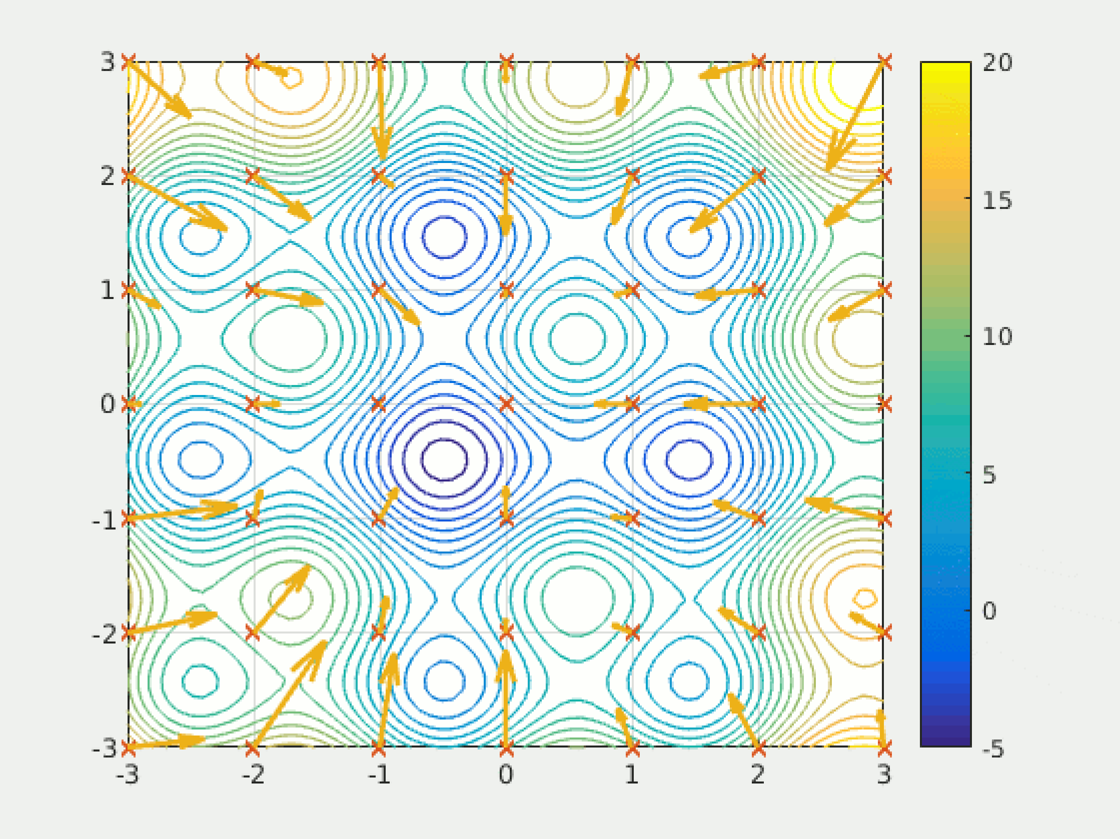


Eventually, all agents land on the roost

- **What if**
  - Roost = (unknown) local optima of a function
  - Distance to the roost = quality of current agent position on the optimization landscape

Particle Swarm Optimization

- Particle Swarm Optimization cosists of simulating the movement of swarm bird-like **particles** (solutions)
- At each iteration, each particle is found at a position in the **solution space**
- The fitness of each particle represents the **quality of its position** on the optimization landscape
- Particles move over the search space with a certain **velocity**
- At each iteration the velocity of a particles is influenced by:
  - pbest: its **own best positions** found so far
  - gbest: the **global best solution** so far
- Eventually the swarm of particles will converge to optimal; positions



Swarm Optimization Algorithms

Algorithm	Proposed by year	Short description
Particle Swarm Optimization (PSO)	**Kennedy et. al. (1995)**	**The algorithm simulates the movement of swarm bird-like particles**
Ant Colony Optimization (ACO)	**Dorigo et. Al. (1996)**	**The algorithm is inspired by the foraging behavior of some ant species.**
Harmony Search (HS)	Geem et. al. (2001)	Search works on the principle of a musical trying to identify a state of pleasing harmony
Honey-Bee Mating Optimization (HBMO)	Abbass (2001)	The algorithm is inspired by matting process of bees
Glowworm Swarm Optimization (GSO)	Krishnanand and Ghose (2006)	The search imitates the behaviour that a glowworm carries a luminescence quantity along with itself to exchange information
Firefly Algorithm (FFA)	Yang (2007)	The algorithm is inspired by the firecles and their ability to emit light through a biochemical process
Bat Algorithm (BA)	Yang (2010)	The algorithm is inspired by the echolocation of bats
Cuckoo Search (CS)	Yang and Deb (2010)	The algorithm is inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nest of host birds
Multi-colony Bacteria Foraging Optimization (MCBFO)	Chen et. al. (2010)	The algorithm integrates the cell-to-cell communication strategies of multi-colony bacterial communities

Particle Swarm Optimization

Nuno Antunes Ribeiro

Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

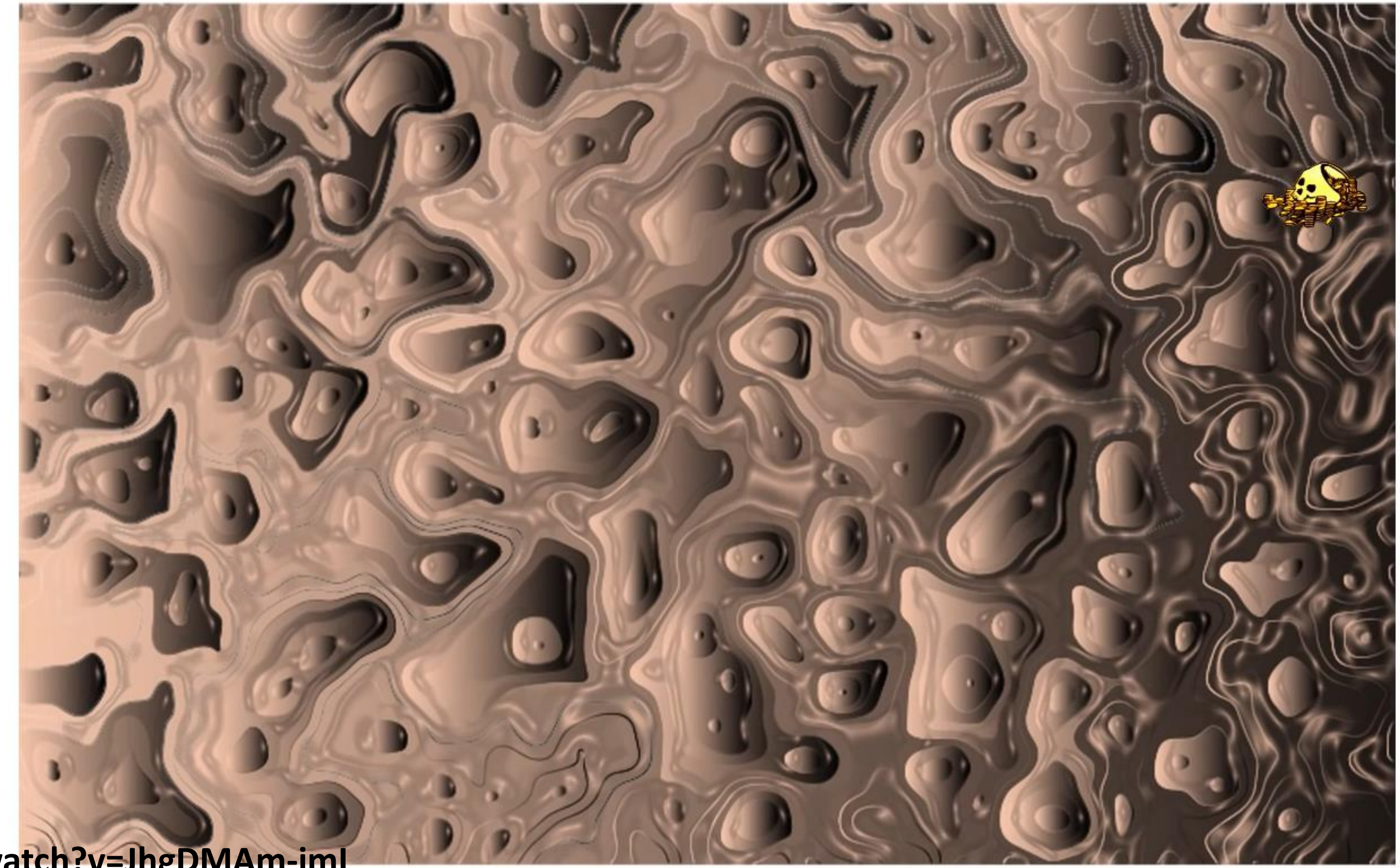
Engineering Systems and Design

PSO Search Procedure

Bob

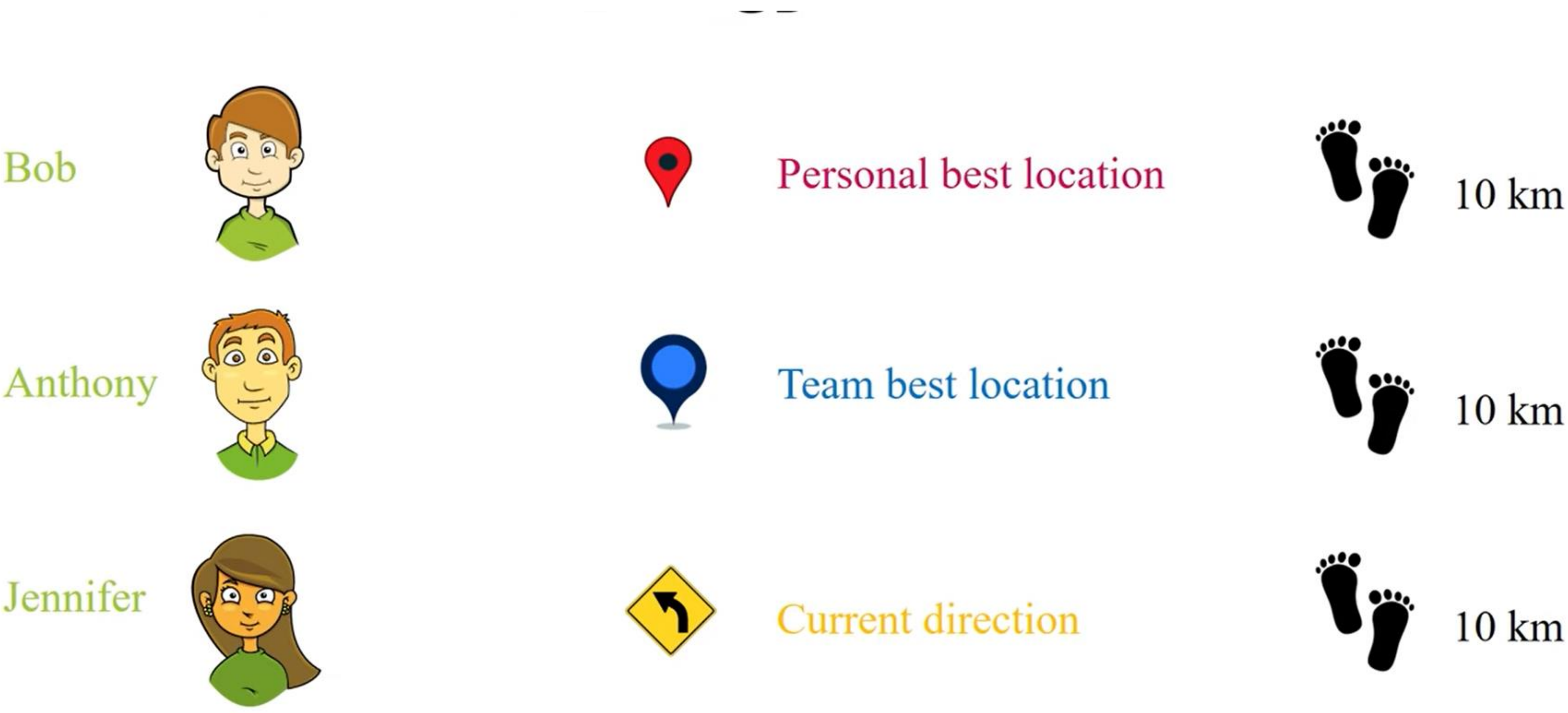
Anthony

Jennifer



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

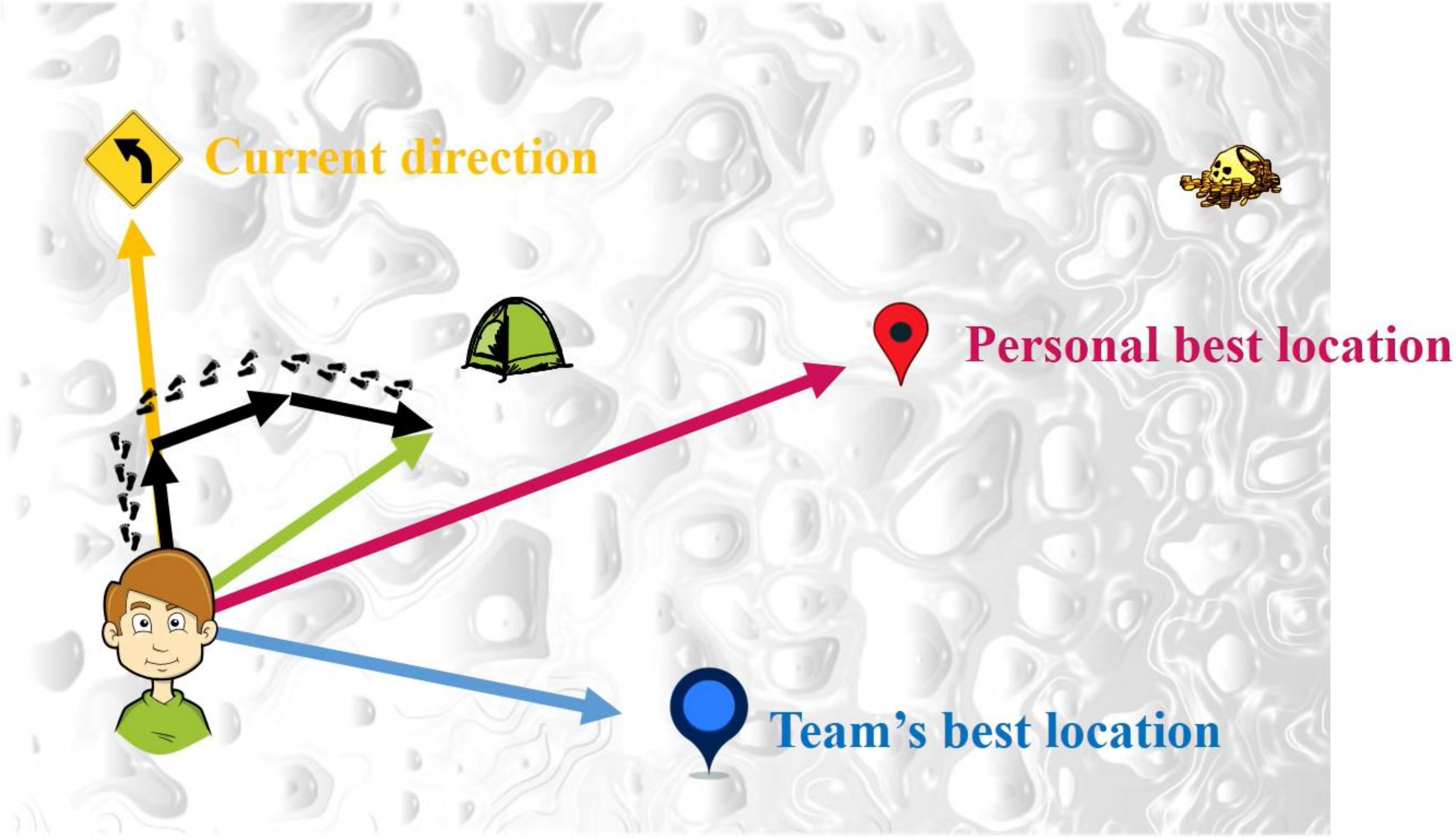


Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

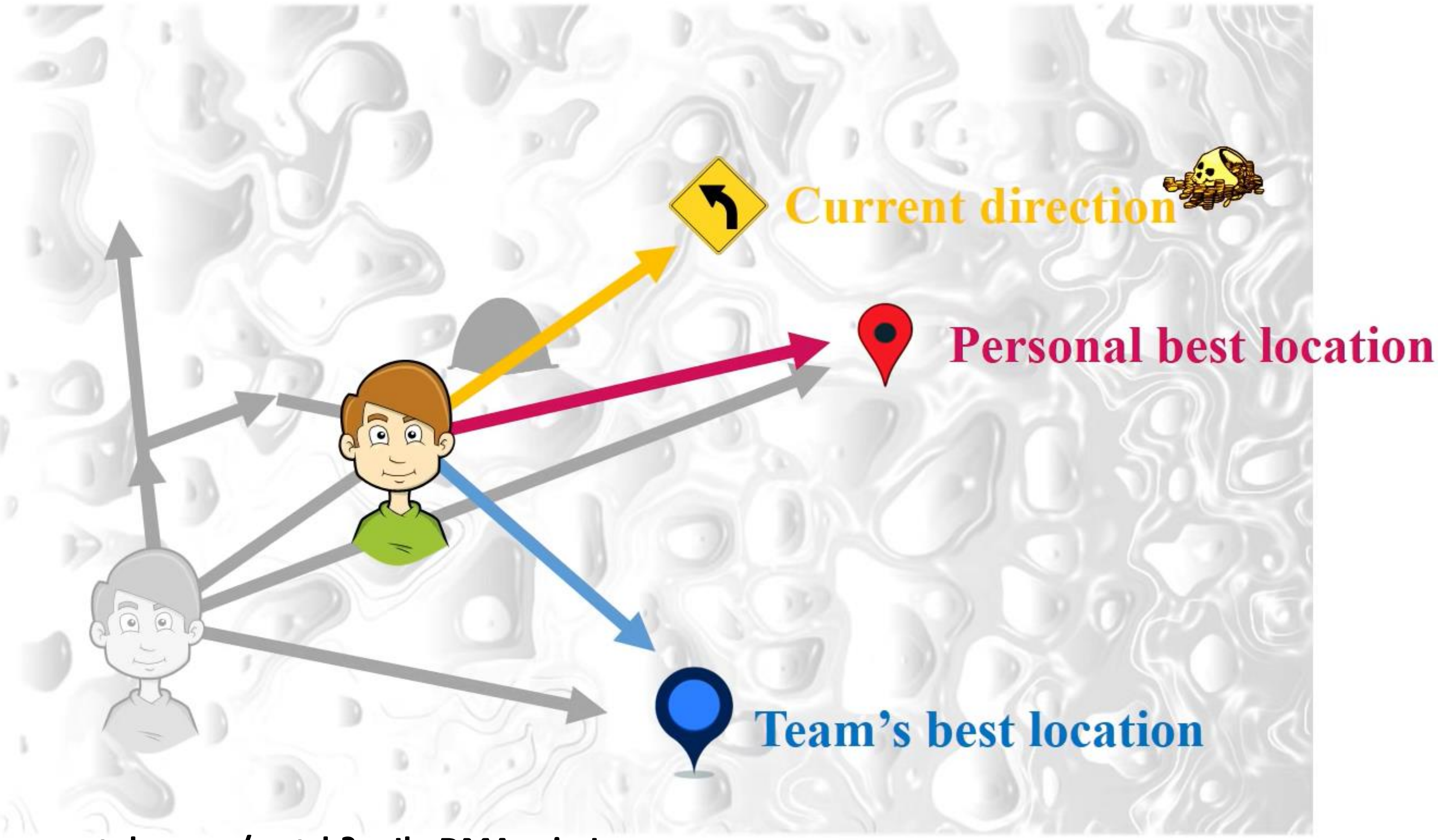
Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure



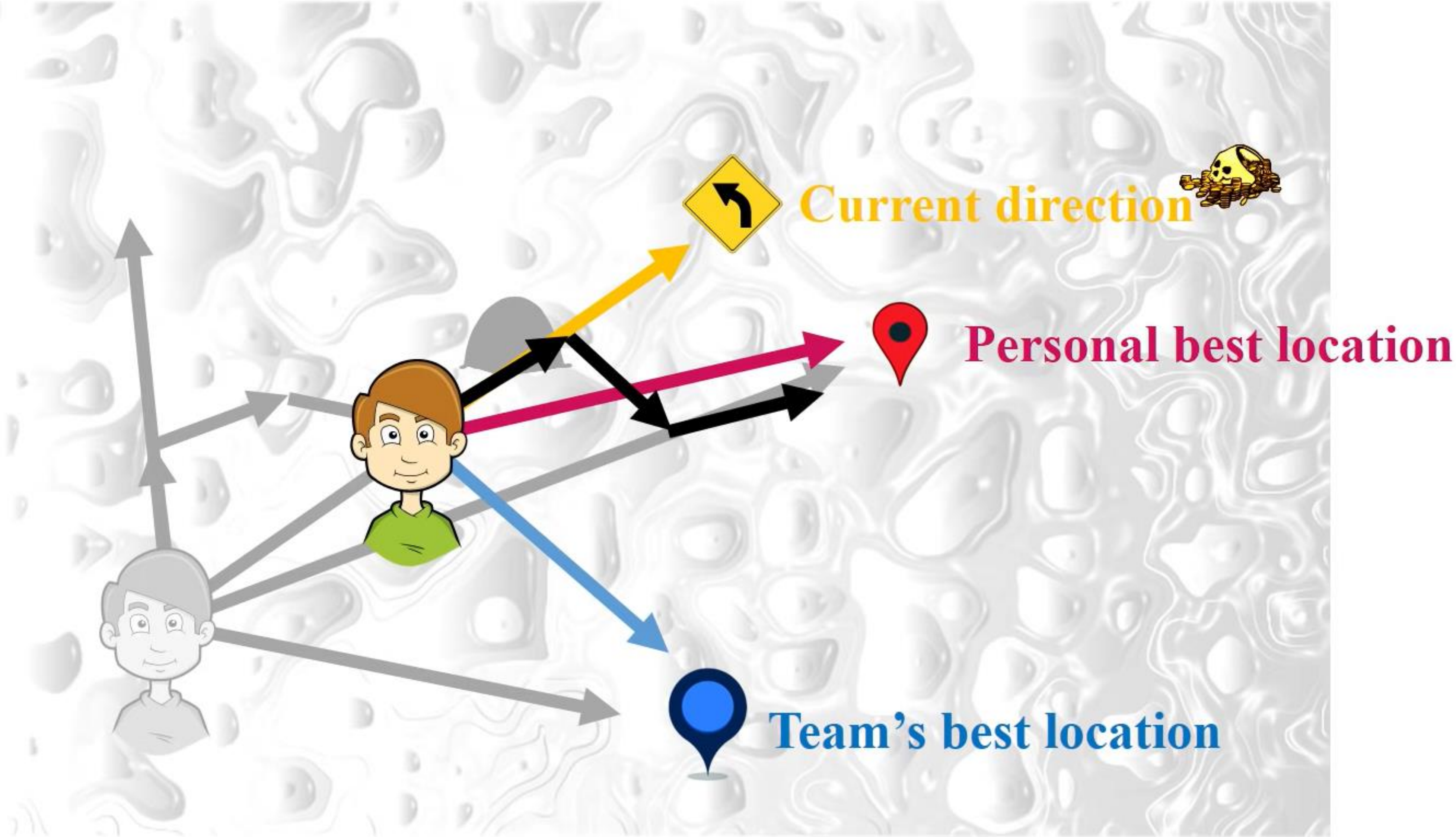
Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure



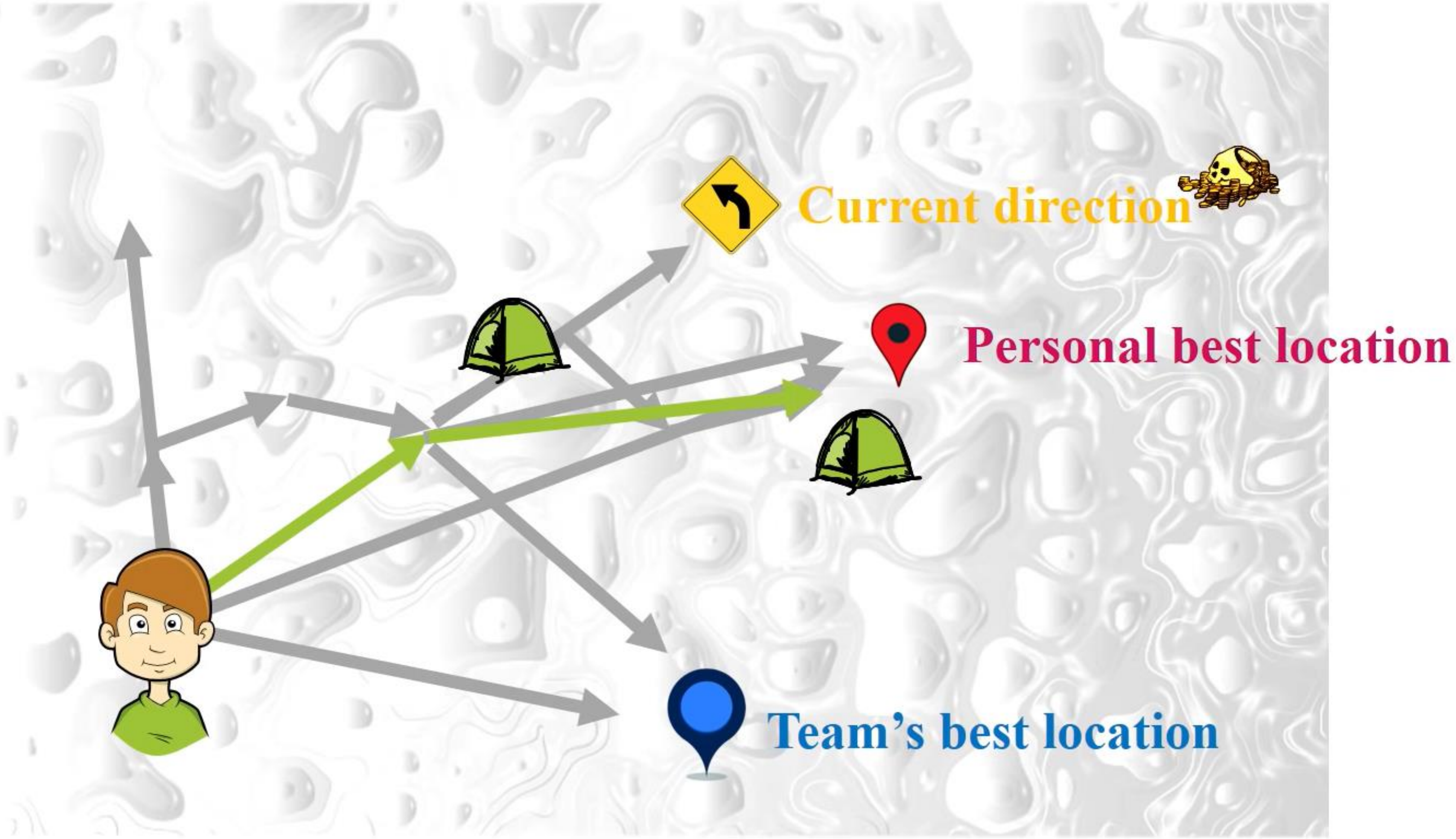
Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure



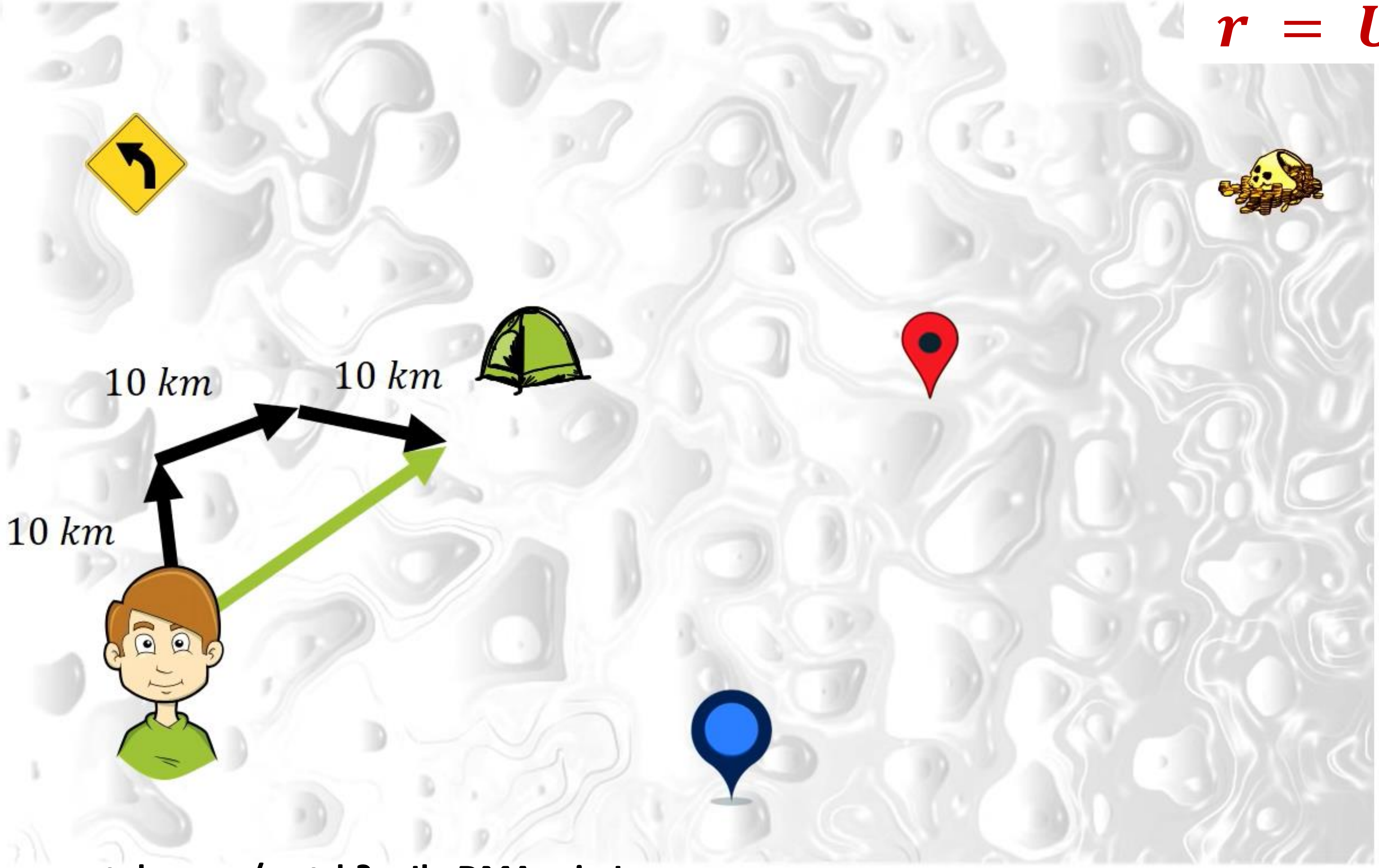
Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

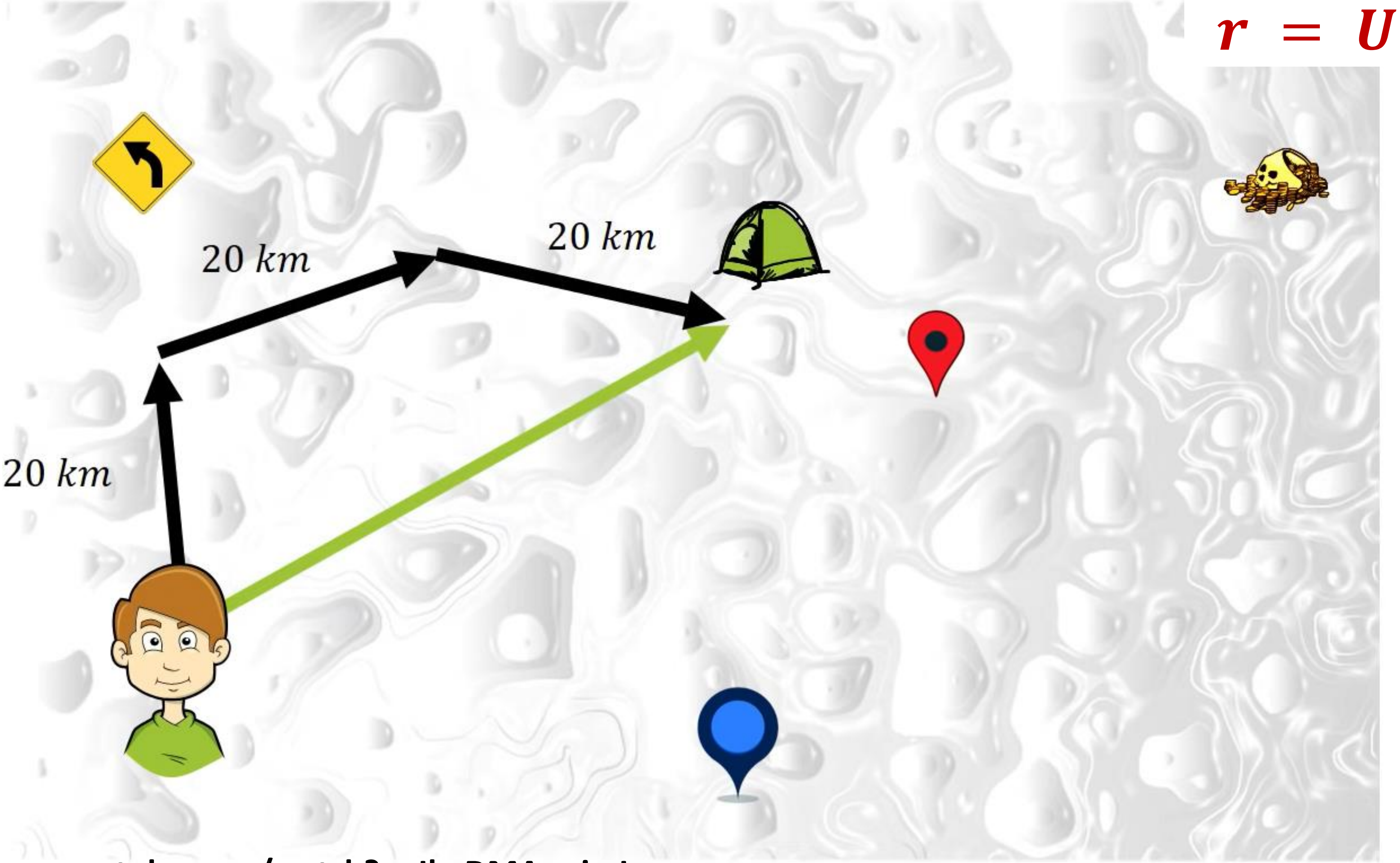
$$2 \times r \times 10km$$
$$r = U(0,1)$$



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

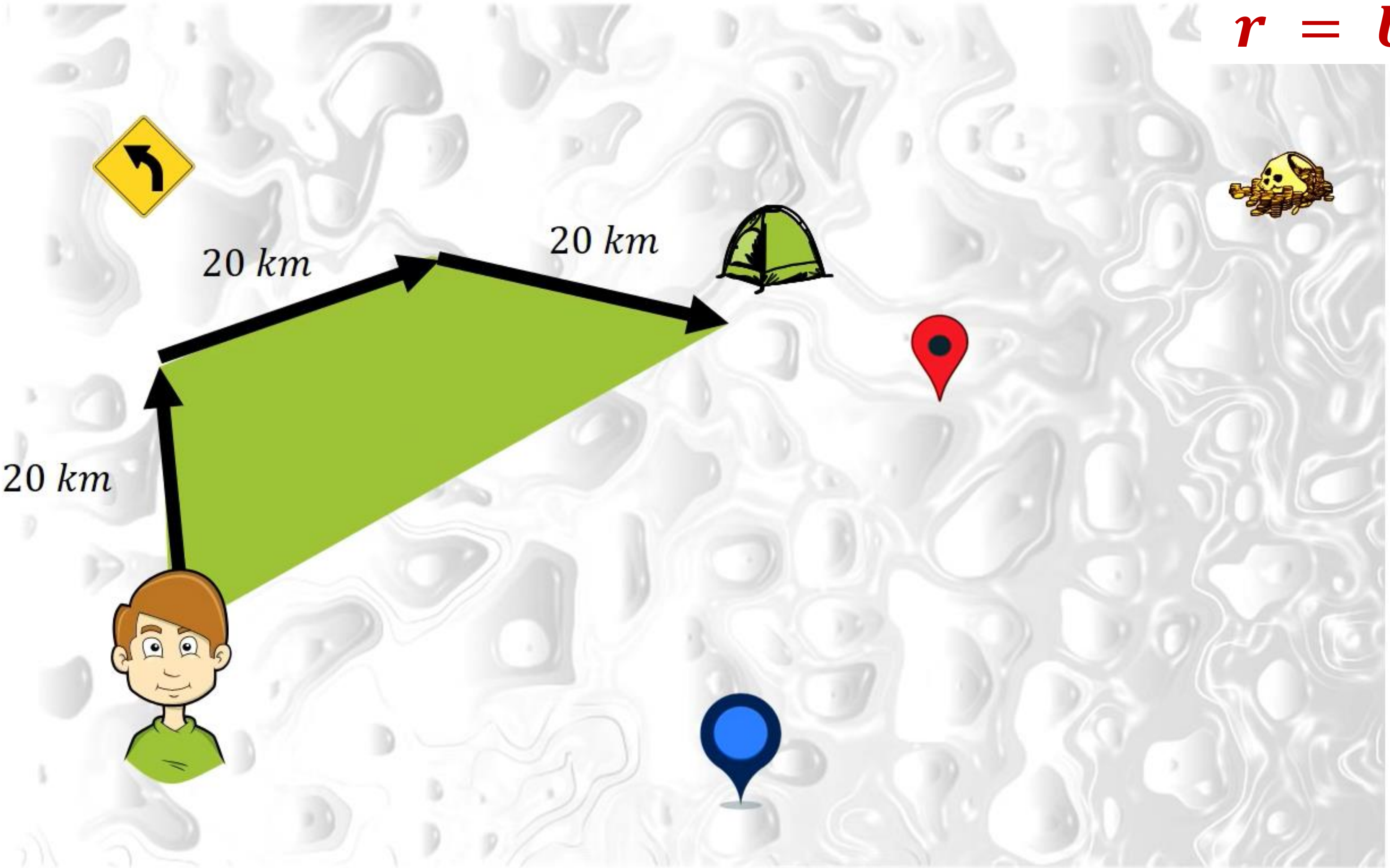
$$2 \times r \times 10km$$
$$r = U(0,1)$$



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

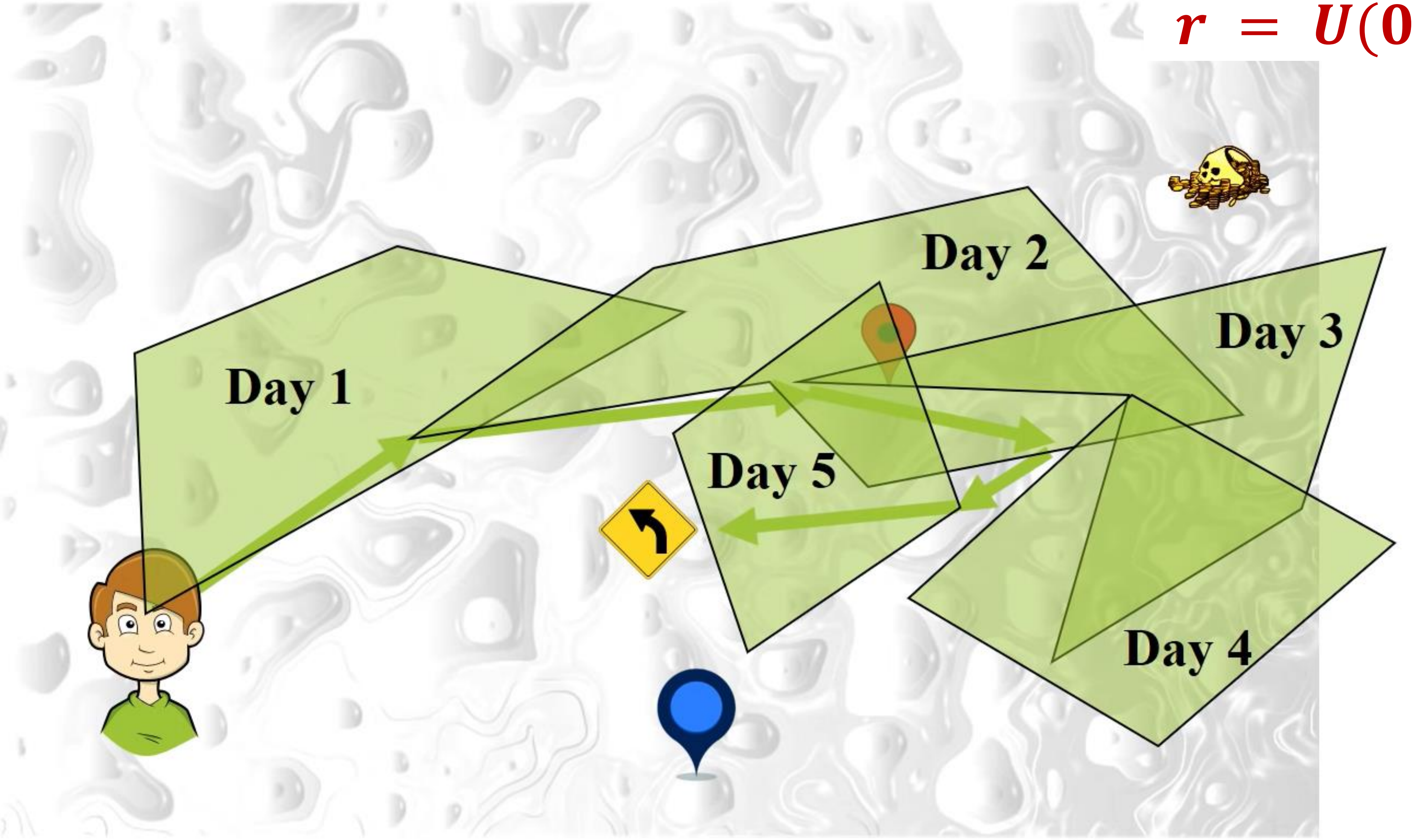
$$2 \times r \times 10km$$
$$r = U(0,1)$$



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

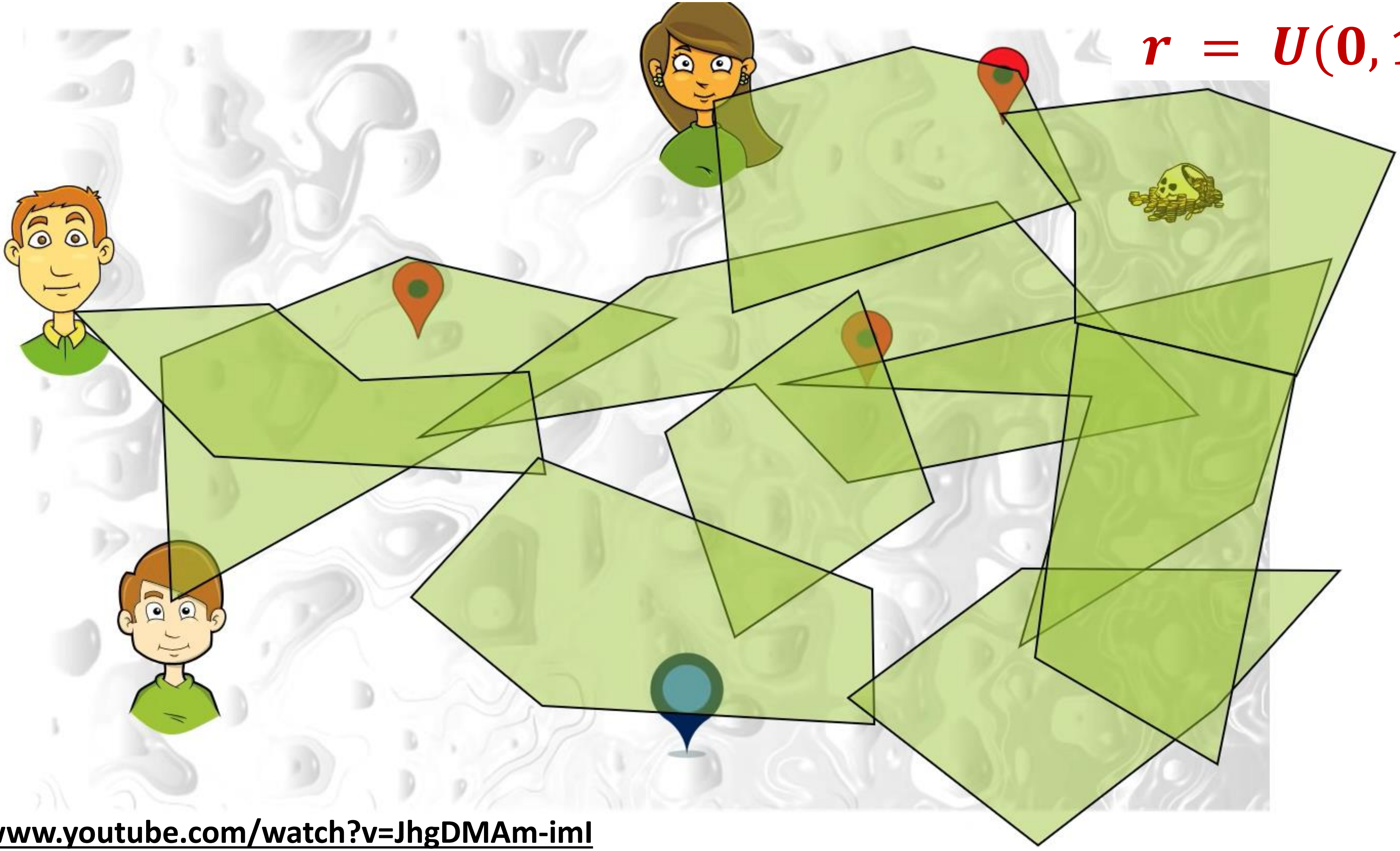
$$2 \times r \times 10km$$
$$r = U(0, 1)$$



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

$$\mathbf{2 \times \boldsymbol{r} \times 10\boldsymbol{km}}$$
$$\boldsymbol{r} = \boldsymbol{U}(\mathbf{0}, \mathbf{1})$$



Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

Bob  $\overrightarrow{X_1^d} = [x_1^d\,, y_1^d]$

Anthony  $\overrightarrow{X_2^d} = [x_2^d\,, y_2^d]$

Jennifer  $\overrightarrow{X_3^d} = [x_3^d\,, y_3^d]$

$$\overrightarrow{X_i^d} = [x_i^d\,, y_i^d, z_i^d, \ldots]$$

Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

$$\overrightarrow{X_i^{d+1}} = \overrightarrow{X_i^{d}} + \overrightarrow{V_i^{d+1}}$$

Position in day *d+1*	Position in day *d*	Velocity in day *d+1*

Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Search Procedure

$$\overrightarrow{V_i^{d+1}} = 2r_1 \boxed{\overrightarrow{V_i^{d}}} + 2r_2 \boxed{\left(\boxed{\overrightarrow{P_i^{d}}} - \overrightarrow{X_i^{d}} \right)} + 2r_3 \boxed{\left(\boxed{\overrightarrow{G^{d}}} - \overrightarrow{X_i^{d}} \right)}$$

Next velocity (tomorrow)

Current velocity (today)

Personal best solution

Global best solution

Distance to the personal best

Distance to the global best

Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO Mathematical Model

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^{t}} + \overrightarrow{V_i^{t+1}}$$

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^{t}} + c_1 r_1 \left(\overrightarrow{P_i^{t}} - \overrightarrow{X_i^{t}}\right) + c_2 r_2 \left(\overrightarrow{G^{t}} - \overrightarrow{X_i^{t}}\right)$$

Inertia | Cognitive component | Social component

Source: https://www.youtube.com/watch?v=JhgDMAm-imI

PSO - Example

- Objective: $\text{maximize } f(X) = x_1^2 - x_1x_2 + x_2^2 + 2x_1 + 4x_2 + 3$
- Where: $-5 \le x_1, x_2 \le 5$

- Population size $= \ 5$
- Inertia weight: $w = 0.9$
- Cognitive weight: $c_1 = 1.5$
- Social weight: $c_2 = 1.5$

PSO - Example

- Iteration 1

$f(X) = x_1^2 - x_1 x_2 + x_2^2 + 2x_1 + 4x_2 + 3$

Since there is no previous iterations $pbest = x$

v_1	v_2
0.2194	0.2449
0.1908	0.2228
0.3828	0.3232
0.3976	0.3547
0.0934	0.3773

Velocity randomly generated $U(0,1)$

x_1	x_2
3.1472	-4.0246
4.0579	-2.2150
-3.7301	0.4688
4.1338	4.5751
1.3236	4.6489

Position randomly generated $U(0,1)$

$f(\text{X})$
31.9645
32.6168
13.2071
48.6753
41.4537

P_1	P_2
3.1472	-4.0246
4.0579	-2.2150
-3.7301	0.4688
4.1338	**4.5751**
1.3236	4.6489

$f(\text{X})$
31.9645
32.6168
13.2071
48.6753
41.4537

G_1	G_2
4.1338	**4.5751**

$f(\text{X})$
48.6753

PSO - Example

- Iteration 2

$$f(X) = x_1^2 - x_1 x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2
0.3240	

x_1	x_2
3.4712	

$f(X)$

P_1	P_2

$f(X)$

G_1	G_2

$f(X)$

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1 r_1 \left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2 r_2 \left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$V_1^2 = 0.9 \times 0.2194 + 1.5 \times 0.5949 \times (3.1472 - 3.1472) + 1.5 \times 0.0855 \times (4.1338 - 3.1472) = 0.3240$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

$$x_1 = 3.1472 + 0.3240 = 3.4712$$

PSO - Example

- Iteration 2

$$f(X) = x_1^2 - x_1x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2
0.3240	1.3233
0.1815	1.0714
1.3531	0.8175
0.3578	0.3192
0.4445	0.3301

x_1	x_2
3.4712	-2.7013
4.2394	-1.1436
-2.3770	1.2863
4.4916	4.8943
1.7681	4.9790

$f(X)$
27.8600
31.0325
13.7537
53.7063
45.5655

P_1	P_2

$f(X)$

G_1	G_2

$f(X)$

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1r_1\left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2r_2\left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

PSO - Example

pbest – previous iteration

P_1	P_2	$f(X)$
3.1472	-4.0246	31.9645
4.0579	-2.2150	32.6168
-3.7301	0.4688	13.2071
4.1338	**4.5751**	**48.6753**
1.3236	4.6489	41.4537

- Iteration 2

$$f(X) = x_1^2 - x_1x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2	x_1	x_2	$f(X)$
0.3240	1.3233	3.4712	-2.7013	27.8600
0.1815	1.0714	4.2394	-1.1436	31.0325
1.3531	0.8175	-2.3770	1.2863	13.7537
0.3578	0.3192	4.4916	4.8943	53.7063
0.4445	0.3301	1.7681	4.9790	45.5655

P_1	P_2	$f(X)$
3.1472	-4.0246	31.9645
4.0579	-2.2150	32.6168
-2.3370	1.2863	13.7537
4.4916	4.8943	53.7063
1.7681	4.9790	45.5655

G_1	G_2	$f(X)$

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1r_1\left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2r_2\left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

PSO - Example

gbest – previous iteration

G_1	G_2	$f(X)$
4.1338	**4.5751**	**48.6753**

- Iteration 2

$$f(X) = x_1^2 - x_1x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2
0.3240	1.3233
0.1815	1.0714
1.3531	0.8175
0.3578	0.3192
0.4445	0.3301

x_1	x_2
3.4712	-2.7013
4.2394	-1.1436
-2.3770	1.2863
4.4916	4.8943
1.7681	4.9790

$f(X)$
27.8600
31.0325
13.7537
53.7063
45.5655

P_1	P_2
3.1472	-4.0246
4.0579	-2.2150
-2.3370	1.2863
4.4916	**4.8943**
1.7681	4.9790

$f(X)$
31.9645
32.6168
13.7537
53.7063
45.5655

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1r_1\left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2r_2\left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

G_1	G_2
4.4916	**4.8943**

$f(X)$
53.7063

PSO - Example

- Iteration 3

$$f(X) = x_1^2 - x_1 x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2	x_1	x_2	$f(X)$	P_1	P_2	$f(X)$
0.1334	0.9843	3.6045	-1.7170				
0.0337	0.7826	4.2731	-0.3611				
2.0987	1.1985	-0.2784	2.4848				
0.3221	0.2873	4.8137	**5.2131**				
0.7493	0.2862	2.5174	**5.3678**				

G_1	G_2	$f(X)$
4.4916	**4.8943**	**53.7063**

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1 r_1 \left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2 r_2 \left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

out of the bounds [-5,5]

PSO - Example

pbest – previous iteration

P_1	P_2	$f(X)$
3.1472	-4.0246	31.9645
4.0579	-2.2150	32.6168
-2.3370	1.2863	13.7537
4.4916	4.8943	53.7063
1.7681	4.9790	45.5655

- Iteration 3

$$f(X) = x_1^2 - x_1 x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2	x_1	x_2	$f(X)$
0.1334	0.9843	3.6045	-1.7170	25.4710
0.0337	0.7826	4.2731	-0.3611	30.0346
2.0987	1.1985	-0.2784	2.4848	19.3259
0.3221	0.2873	4.8137	**5.0000**	56.7306
0.7493	0.2862	2.5174	**5.0000**	46.7851

P_1	P_2	$f(X)$
3.1472	-4.0246	31.9645
4.0579	-2.2150	32.6168
-0.2784	2.4848	19.3259
4.8137	5.0000	56.7306
2.5174	5.0000	46.7851

G_1	G_2	$f(X)$
4.4916	**4.8943**	**53.7063**

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1 r_1 \left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2 r_2 \left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

out of the bounds [-5,5]

PSO - Example

gbest – previous iteration

G_1	G_2	$f(X)$
4.4916	**4.8943**	**53.7063**

- Iteration 3

$$f(X) = x_1^2 - x_1x_2 + x_2^2 + 2x_1 + 4x_2 + 3$$

v_1	v_2	x_1	x_2	$f(X)$	P_1	P_2	$f(X)$
0.1334	0.9843	3.6045	-1.7170	25.4710	3.1472	-4.0246	31.9645
0.0337	0.7826	4.2731	-0.3611	30.0346	4.0579	-2.2150	32.6168
2.0987	1.1985	-0.2784	2.4848	19.3259	-0.2784	2.4848	19.3259
0.3221	0.2873	**4.8137**	**5.0000**	**56.7306**	4.8137	5.0000	56.7306
0.7493	0.2862	2.5174	5.0000	46.7851	2.5174	5.0000	46.7851

G_1	G_2	$f(X)$
4.8137	**5.0000**	**56.7306**

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + c_1r_1\left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2r_2\left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

PSO Parameters

Exploration: Search for new regions of the solution space. Aims to find the regions with potentially the best solutions

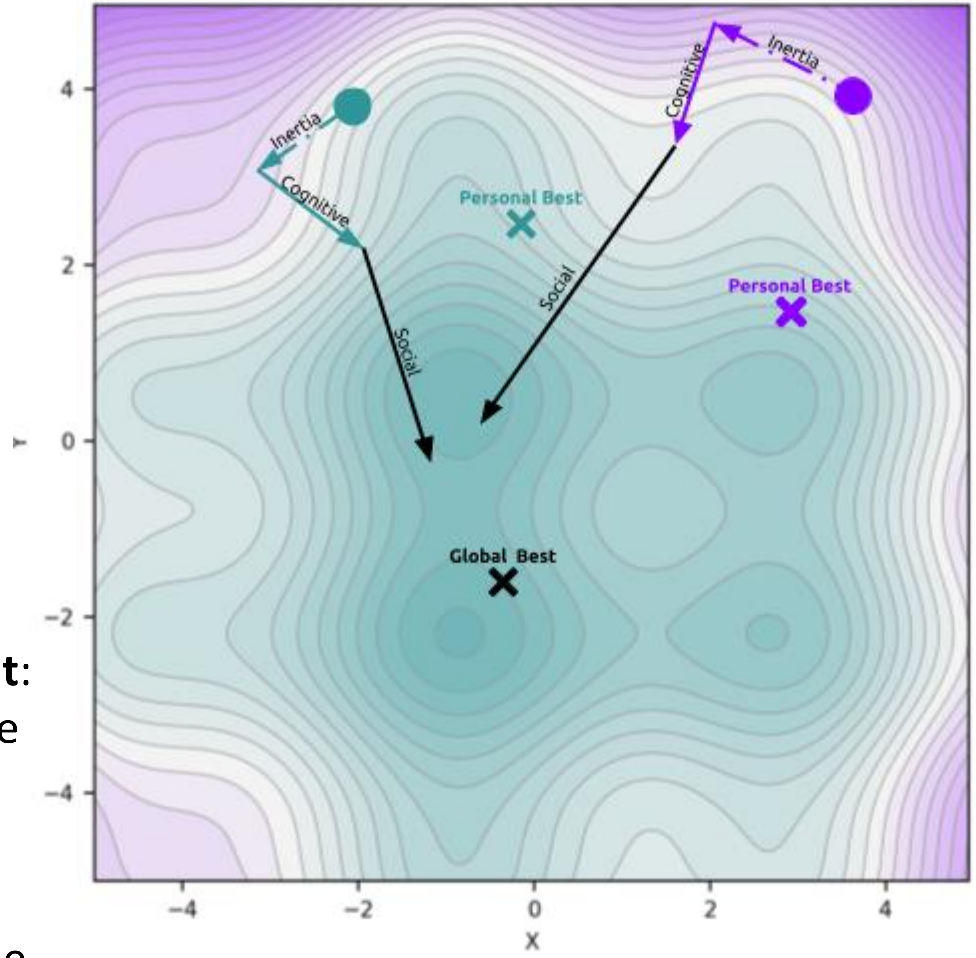
Exploitation: Explores previous promising regions of the solution space.

$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^{t}} + \underbrace{c_1 r_1 \left(\overrightarrow{P_i^{t}} - \overrightarrow{X_i^{t}}\right) + c_2 r_2 \left(\overrightarrow{G^{t}} - \overrightarrow{X_i^{t}}\right)}$$

Inertia: Makes the particle move in the same direction and velocity. The parameter W is important for balancing exploration and exploitation

Cognitive Component: Makes the particle to return to a previous promising position

Social Component: Makes the particle to move in the direction of the best solution found so far by the team.



PSO Parameters

- **Swarm size** (N) - number of particles in the swarm: the more particles in the swarm, the larger the **initial diversity**. A large swarm allows larger parts of the search space to be covered per iteration. However, more particles increase the per iteration **computational complexity**, and the search degrades to a parallel random search
- It has been shown in a number of empirical studies that small swarm sizes of **10 to 30** tend to provide better results – however note that the optimal swarm size is problem-dependent

PSO Parameters

- **Inertia Coefficient** (w) allows to define the ability of the swarm to change its direction.

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

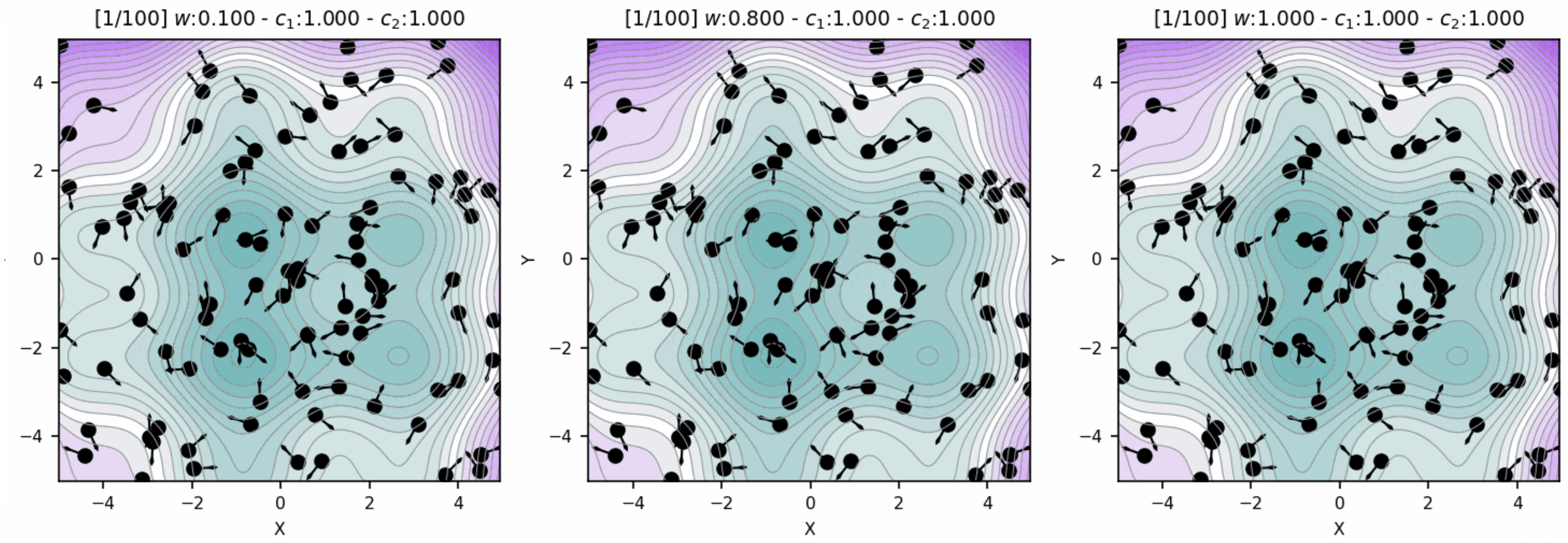
$$\overrightarrow{V_i^{t+1}} = \boxed{w\overrightarrow{V_i^t}} + c_1 r_1 \left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2 r_2 \left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)$$

Inertia

Cognitive component

Social component

A low coefficient w facilitates the exploitation of the best solutions found so far while a high coefficient w facilitates the exploration around these solutions. Note that it is recommended to avoid $w > 1$ which can lead to a divergence of our particles.



PSO Parameters

- **Cognitive Coefficient** ($c1$) allows defining the ability of the group to be influenced by the best personal
- **Social Coefficient** ($c2$) allows defining the ability of the group to be influenced by the best global solution found over the iterations.

$$\overrightarrow{X_i^{t+1}} = \overrightarrow{X_i^t} + \overrightarrow{V_i^{t+1}}$$

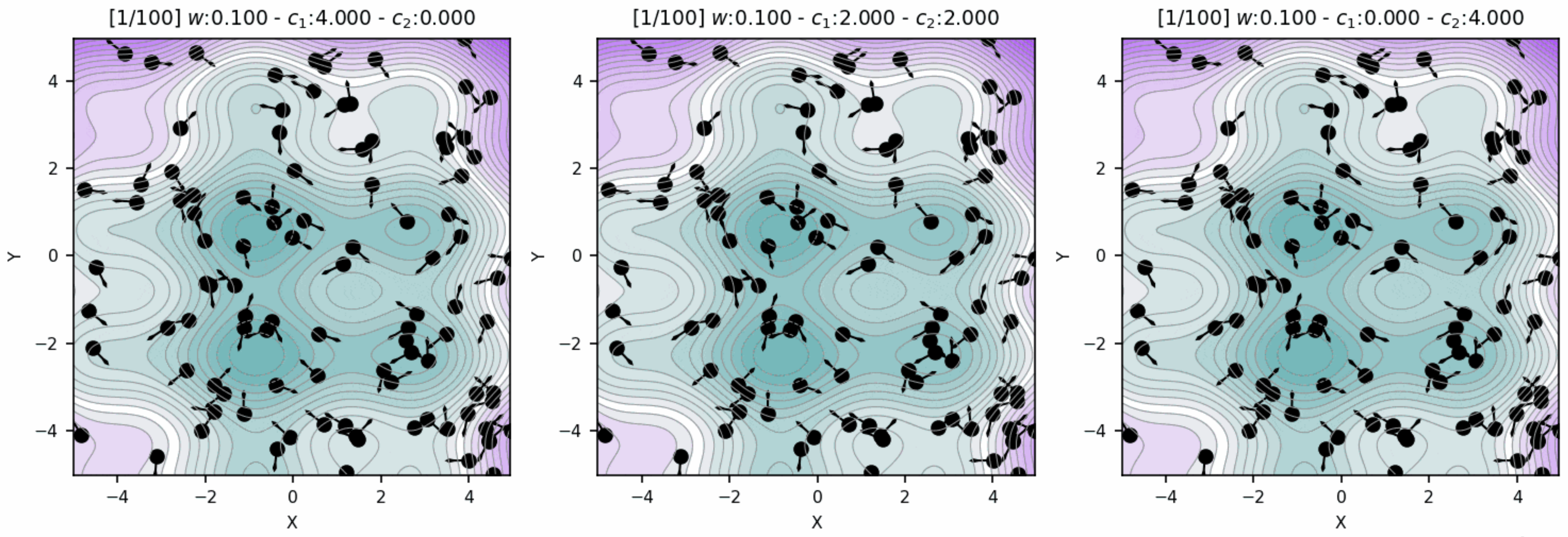
$$\overrightarrow{V_i^{t+1}} = w\overrightarrow{V_i^t} + \boxed{c_1 r_1 \left(\overrightarrow{P_i^t} - \overrightarrow{X_i^t}\right) + c_2 r_2 \left(\overrightarrow{G^t} - \overrightarrow{X_i^t}\right)}$$

Inertia

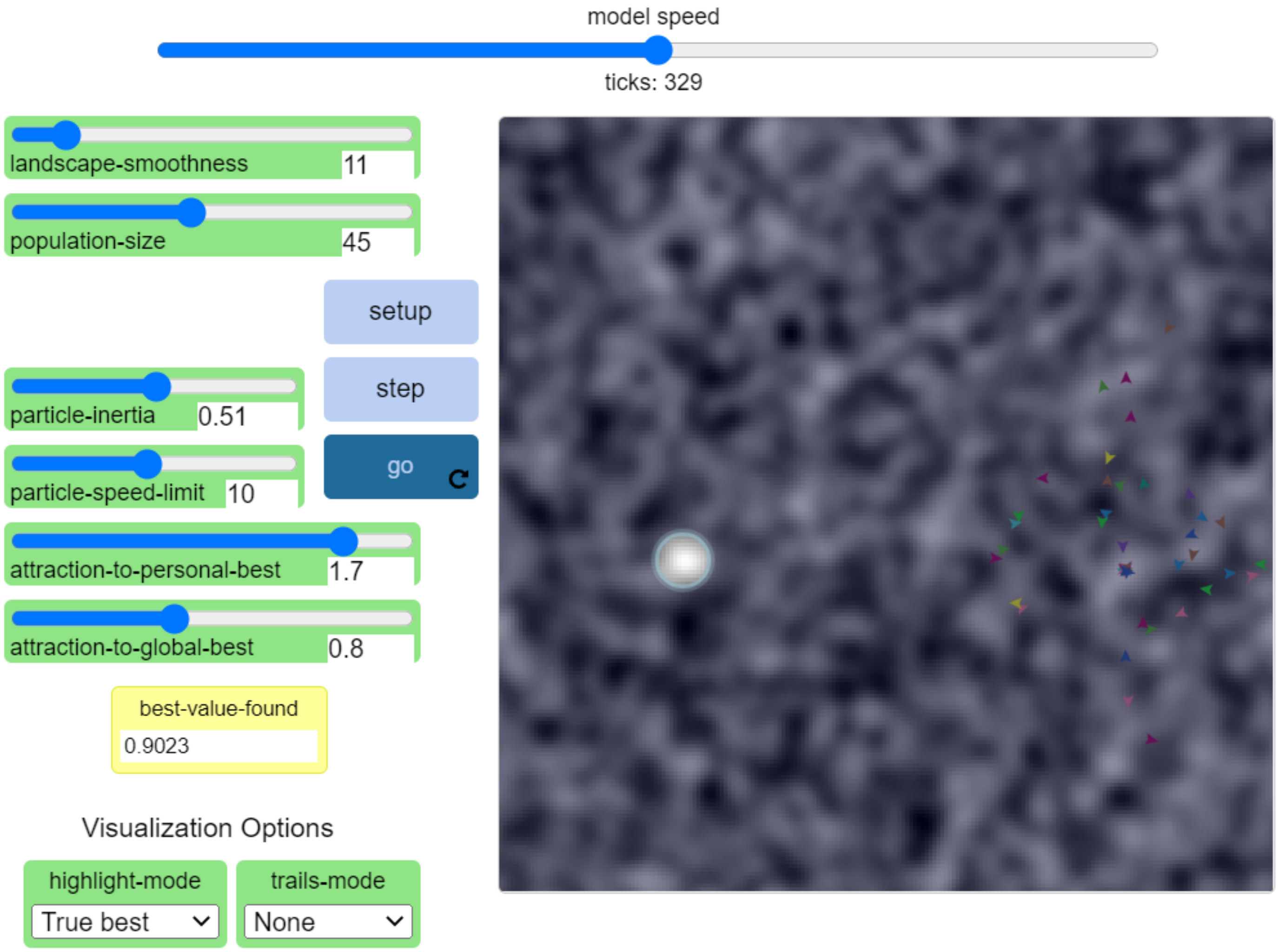
Cognitive component

Social component

The particles of the swarm are more individualistic when $c1$ is high (exploration) . There is, therefore, no convergence because each particle is only focused on its own best solutions. In contrast, the particles of the swarm are more influenced by the others when c2 is high.



Try it yourself



http://www.netlogoweb.org/launch#http://ccl.northwestern.edu/netlogo/models/models/Sample%20Models/Computer%20Science/Particle%20Swarm%20Optimization.nlogo

Adaptive PSO

- According to the paper by M. Clerc and J. Kennedy to define a standard for Particle Swarm Optimization, the best static parameters are $w \approx 0.73$ and $c1 + c2 > 4$. More exactly $c1 = c2 = 2.05$ (obtained empirically)
- An adaptive procedure may lead to better balance between exploration and exploitation - starting with a strong c_1, strong w, and weak c_2 to increase the exploration in the first iterations. Then, the parameters are updated, towards a weak c1, weak w, and strong c2 to increase exploitation around the best region.

$$w^t = 0.4\frac{(t-N)}{N^2} + 0.4$$

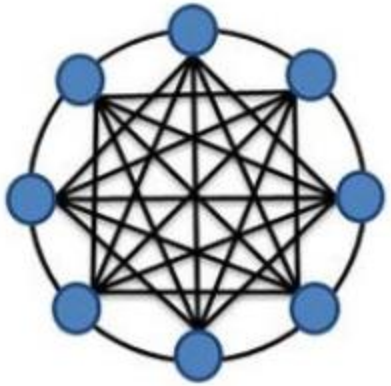
$$c_1^t = -3\frac{t}{N} + 3.5$$

$$c_2^t = +3\frac{t}{N} + 0.5$$

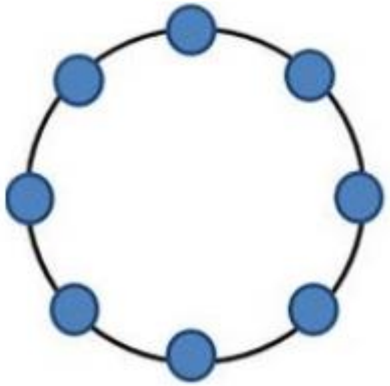
t – *current iteration*
N – *total number of iterations*

Swarm Topology

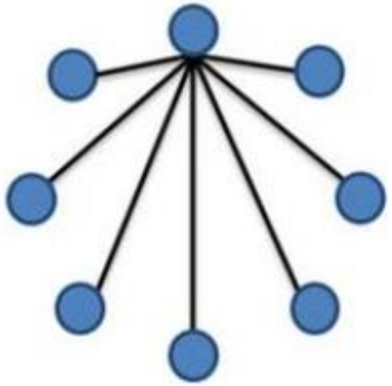
- The topology of the swarm defines the subset of particles with which each particle can exchange information.
- The basic version of PSO uses the global topology as the swarm communication structure. This topology allows all particles to communicate with all the other particles, thus the whole swarm share the same best position g from a single particle.
- However, this approach might lead the swarm to be trapped into a local minimum, thus different topologies have been used to control the flow of information among particles.



Global topology: each particle is attracted to the best group particle noted gbest, and communicates with the others.

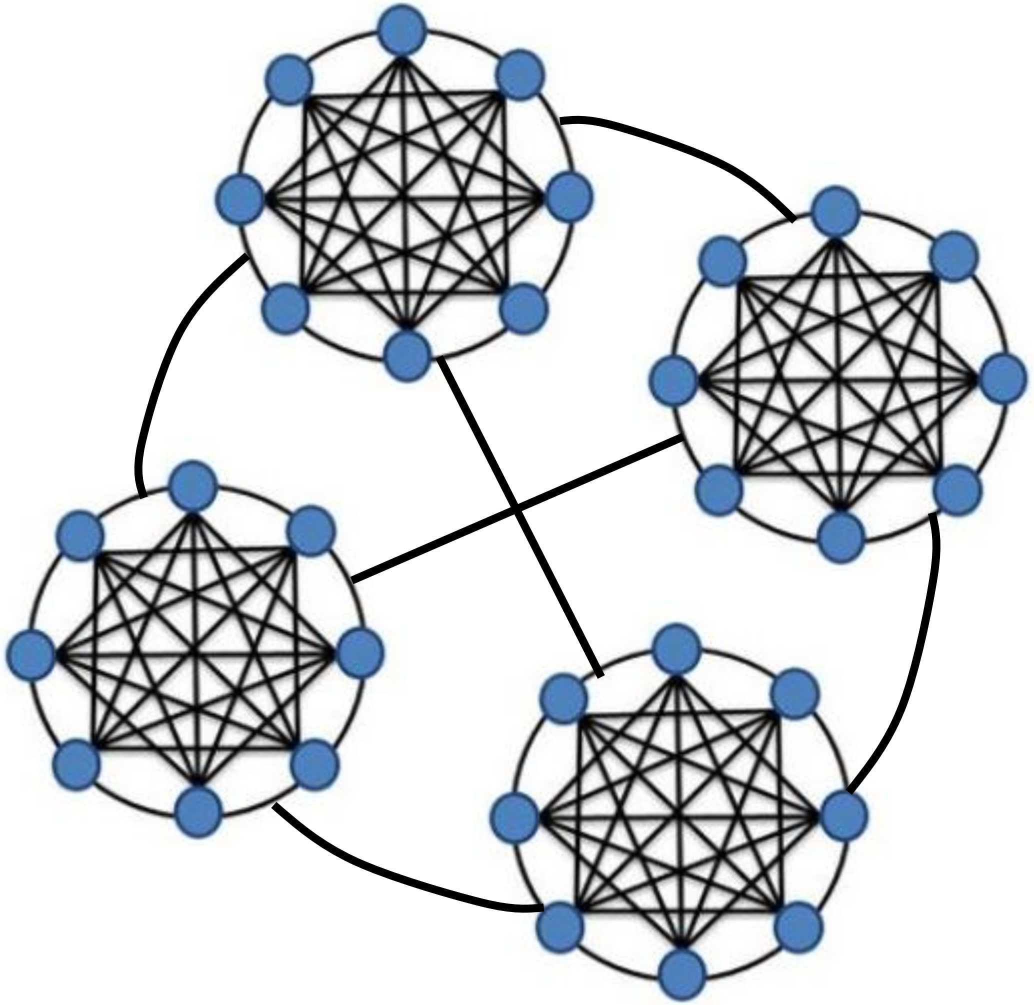


Ring topology: each particle communicates
with n immediate neighbors, and tends to move towards
the best position in the local neighborhood called nbest



Star topology : a central particle is connected to all others. Only this central particle adjusts its position towards the best, if this causes an improvement
the information is propagated to the others.

Swarm Topology



Cluster-based global topology

Discrete Optimization

- PSO algorithms are applied traditionally to continuous optimization problems. Some adaptations must be made for discrete optimization problems. Two methods are often used:
  - **Discrete PSO with Crossover Operators**: Crossover operators are used to guide the particles towards the gbest and the lbest (cognitive and social moves); Mutation operators are often used to facilitate exploration (inertia move)
  - **Binary PSO with Sigmoid Function**: velocity assume real values, however a sigmoid functions is used to transform the velocities into the binary interval

PSO with Crossover Operators

Recall from Evolutionary Algorithms

- **Discrete & Binary Operators**

Single-Point Crossover (SPX)

Discrete

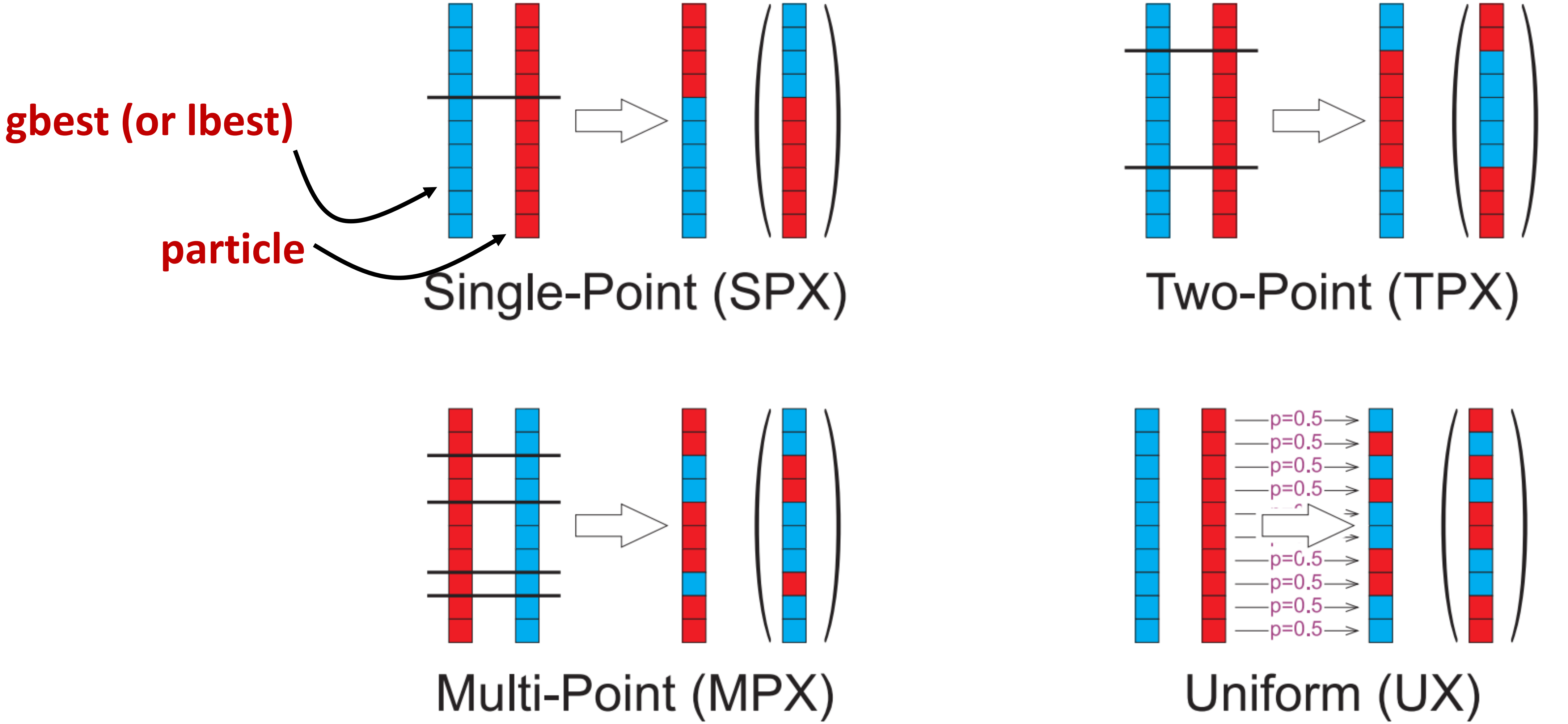
Parents										
gbest (or lbest)	5	4	4	2	5	2	1	1	4	5
particle	3	3	3	2	5	3	3	1	5	6
Offsprings										
	5	4	4	2	5	3	3	1	5	6
	3	3	3	2	5	2	1	1	4	5

Binary

Parents										
	1	0	0	0	1	1	1	0	0	1
	0	0	1	1	1	1	0	1	0	0
Offsprings										
	1	0	0	0	1	1	0	1	0	0
	0	0	1	1	1	1	1	0	0	1

PSO with Crossover Operators

Recall from Evolutionary Algorithms



PSO with Crossover Operators

Particle	3	3	3	2	5	3	3	1	5	3

lbest	5	4	3	1	1	5	3	4	1	5
gbest	5	4	4	2	5	2	1	1	4	5

Step 1 – mutation operator to simulate inertia in continuous PSO

Particle	3	3	3	2	2	3	3	4	5	3

Step 2 – crossover operator to simulate cognitive component in continuous PSO

Acceptance Rate=0.5

lbest	5	4	3	1	1	5	3	4	1	5
$U(0,1)$	0.1	**0.6**	0.3	0.2	**0.5**	**0.8**	0.2	0.1	**0.7**	**0.9**
Particle	3	3	3	2	2	3	3	4	5	3

→

3	4	3	2	1	5	3	4	1	5

Step 3 – crossover operator to simulate social component in continuous PSO

Acceptance Rate=0.5

gbest	5	4	4	2	5	2	1	1	4	5
$U(0,1)$	**0.9**	0.1	0.3	**0.8**	0.1	**0.5**	**0.6**	0.4	0.4	**0.5**
Particle	3	4	3	2	1	5	3	4	1	5

→

5	4	3	2	1	2	1	4	1	5

PSO in Python

Nuno Antunes Ribeiro

Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Engineering Systems and Design

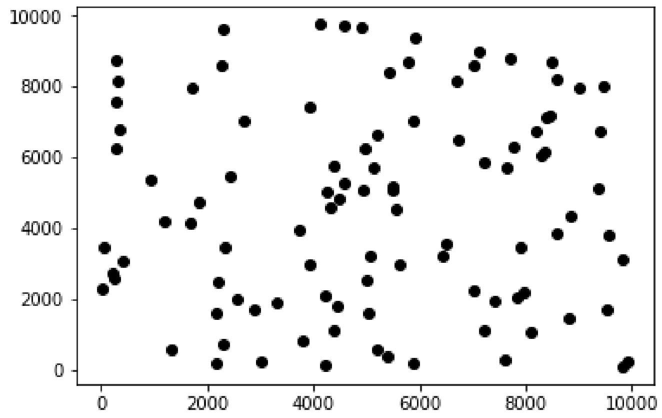
TSP Instance

Generate and Process Instance Data

```
#Generate Data Inputs

# Select random see
random.seed(1)

# Number of cities
n=100

#Coordinate Range
rangelct=10000

#No. of swaps at each iteration
no_swap=1

#Generate random locations
coordlct_x = random.choices(range(0, rangelct), k=n)
coordlct_y = random.choices(range(0, rangelct), k=n)
```

Same seed – same instances solved using local search metaheuristics

```
plt.plot(coordlct_x, coordlct_y, 'o', color='black');
```



Object City

- A city is an object with data concerning the corresponding coordinates and functions (methods) to compute distance between city objects

```
class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, city):
        return math.hypot(self.x - city.x, self.y - city.y)

    def __repr__(self):
        return f"({self.x}, {self.y})"

cities = []
for line in range(n):
    cities.append(City(coordlct_x[line], coordlct_y[line]))
```

```
#Compute Distance Between Cities
City.distance(cities[1],cities[2])
```

```
3115.5368718729683
```

```
cities
```

```
[(1343, 561),
 (8474, 8700),
 (7637, 5699),
 (2550, 1998),
 (4954, 5047),
 (4494, 4849),
 (6515, 3567),
 (7887, 3460),
 (938, 5384),
 (283, 6234),
 (8357, 6124),
 (4327, 4581),
```

Generate Initial Population

- **2-approaches to generate the initial population:**
  - Completely at random (good to ensure diversity)
  - Greedy approach (initiate the search with good solutions)

```python
#Function to generate a completely random route
def random_route():
    return random.sample(cities, len(cities))
```

Function to generate random route

```python
#Funtion to generate route using greedy approach
def greedy_route(start_index, cities):
    unvisited = cities[:]
    del unvisited[start_index]
    route = [cities[start_index]]
    while len(unvisited):
        index, nearest_city = min(enumerate(unvisited), key=lambda num: num[1].distance(route[-1]))
        route.append(nearest_city)
        del unvisited[index]
    return route
```

Function to generate greedy route:
nearest neighbour heuristic

```python
def initial_population(self):
    random_population = [self.random_route() for _ in range(self.population_size-self.greedy_seed)]
    greedy_population = [self.greedy_route(0) for _ in range(self.greedy_seed)]
    return [*random_population, *greedy_population]
```

Overview of the Code

```python
def run(self):
    self.gbest = min(self.particles, key=lambda p: p.pbest_cost)
    print(f"initial cost is {self.gbest.pbest_cost}")
    while self.cputime_i<self.CPUtimemax:
        self.gbest = min(self.particles, key=lambda p: p.pbest_cost)
        self.cputime_i=time.time()-program_starts
        self.cputime=np.append(self.cputime, self.cputime_i)
        print(self.gbest.pbest_cost)

        self.gcost_iter.append(self.gbest.pbest_cost)

        for particle in self.particles:
            particle.clear_velocity()
            gbest_i = self.gbest.pbest[:]
            new_route = particle.route[:]

            swap_it=0
            while swap_it<self.no_swap:
                idx = range(len(self.cities))
                i1, i2 = random.sample(idx, 2)
                new_route[i1], new_route[i2] = new_route[i2], new_route[i1]
                swap_it=swap_it+1

            for i in range(len(self.cities)):
                 if new_route[i] != particle.pbest[i]:
                    swap = (i, particle.pbest.index(new_route[i]), self.pbest_probability)
                    if random.random() <= swap[2]:
                        new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]

            for i in range(len(self.cities)):
                if new_route[i] != gbest_i[i]:
                    swap = (i, new_route.index(gbest_i[i]), self.gbest_probability)
                    if random.random() <= swap[2]:
                        new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]

            particle.route = new_route
            particle.update_costs_and_pbest()
```

Compute gbest

For loop through all particles

Apply swap mutation (Inertia Operator)

Apply crossover to lbest (Cognitive Operator)

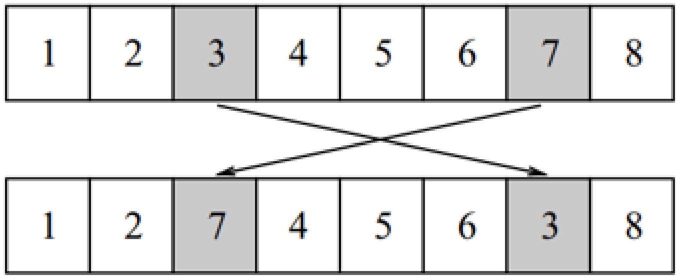
Apply crossover to gbest (Social Operator)

Swap Mutation – Inertia Operator

- Apply *n (self.no_swap)* random swaps

```
swap_it=0
while swap_it<self.no_swap:
    idx = range(len(self.cities))
    i1, i2 = random.sample(idx, 2)
    new_route[i1], new_route[i2] = new_route[i2], new_route[i1]
    swap_it=swap_it+1
```

Swap Operator



Crossover – Cognitive Operator

```python
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
    if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10

$U(0,1)$ **0.1**

Pbest_probability=0.5

new_route (t+1)	4	2	3	1	5	6	7	8	9	10

Crossover – Cognitive Operator

```
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
     if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7								

Pbest_probability=0.5

new_route (t+1)	4	2	3	1	5	6	7	8	9	10

Crossover – Cognitive Operator

```python
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
     if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3							

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	6	3	8	9	10

Crossover – Cognitive Operator

```python
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
    if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8

new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1						

Pbest_probability=0.5

new_route (t+1)	4	1	7	2	5	6	3	8	9	10

Crossover – Cognitive Operator

```
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
    if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1	0.7					

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	6	3	8	9	10

Crossover – Cognitive Operator

```
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
     if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1	0.7	0.6				

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	6	3	8	9	10

Crossover – Cognitive Operator

```python
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
     if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1	0.7	0.6	0.5			

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	3	6	8	9	10

Crossover – Cognitive Operator

```
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
     if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1	0.7	0.6	0.5	0.3		

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	8	6	3	9	10

Crossover – Cognitive Operator

```
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
     if new_route[i] != particle.pbest[i]:
        swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1	0.7	0.6	0.5	0.3	0.8	

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	8	6	3	9	10

Crossover – Cognitive Operator

```python
#Cognitive Component
pbest_probability=0.5
temp_velocity = []
for i in range(len(cities)):
      if new_route[i] != particle.pbest[i]:
         swap = (i, particle.pbest.index(new_route[i]), pbest_probability)
         if random.random() <= swap[2]:
             new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
```

pbest	4	5	7	2	9	1	6	3	10	8
new_route (t)	1	2	3	4	5	6	7	8	9	10
$U(0,1)$	0.1	0.7	0.3	0.1	0.7	0.6	0.5	0.3	0.8	0.2

Pbest_probability=0.5

new_route (t+1)	4	2	7	1	5	10	6	3	9	8

Crossover – Social Operator

```
gbest_probability=0.5
for i in range(len(cities)):
    if new_route[i] != gbest_i[i]:
        swap = (i, new_route.index(gbest_i[i]), gbest_probability)
        if random.random() <= swap[2]:
            new_route[swap[0]], new_route[swap[1]] = new_route[swap[1]], new_route[swap[0]]
            print(swap)
```

Similar to Cognitive Operator