Metaheuristics

Seth Bullock

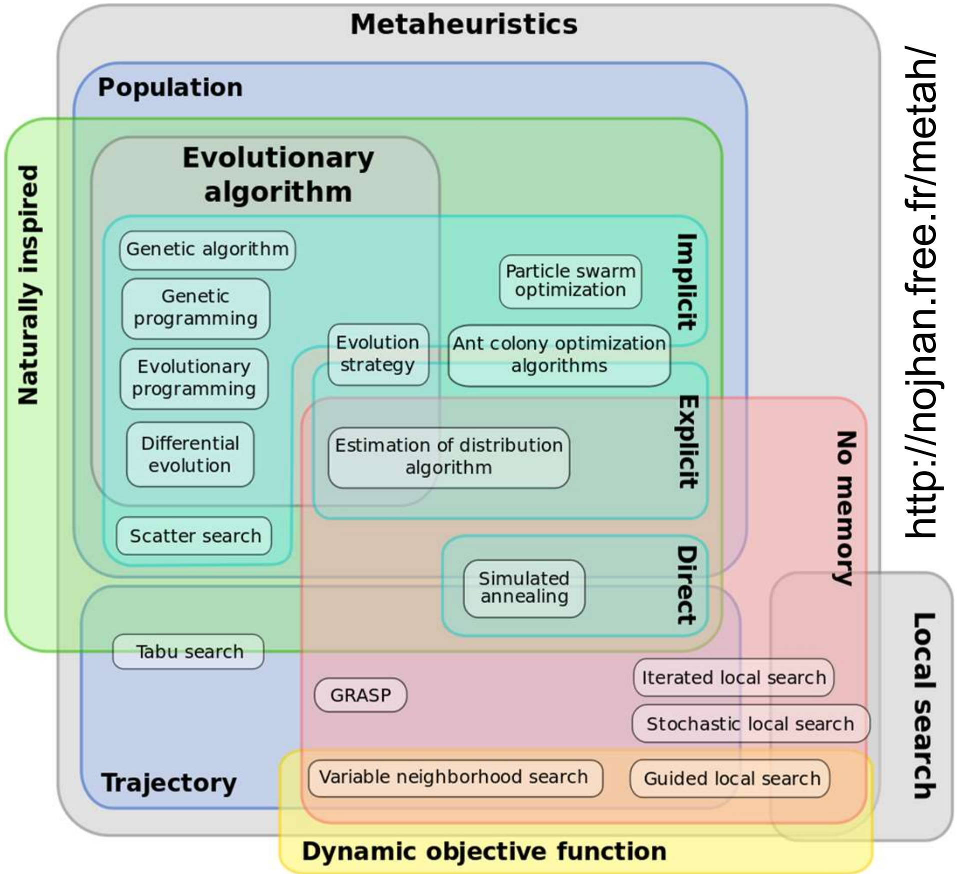
What is Evolutionary Computing?

Evolutionary Computing (EC) techniques:

- a diverse collection of ‘nature-inspired’ techniques
- *heuristic*: not guaranteed to find the best solution
  - (the opposite of an ‘exact’ algorithm)
- *stochastic*: they use random number generators
  - (the opposite of ‘deterministic’ algorithms)
- intended for hard optimization problems (e.g., NP-hard)

Metaheuristics

- EC techniques are examples of *metaheuristic* approaches.
  - A metaheuristic is a way to guide search
  - Some are nature inspired
  - Some are population based
  - Some build an explicit model of the problem as it’s being solved



EC Examples

There are several different kinds or flavours of EC algorithm:

- Genetic Algorithms (GA)
- Genetic Programming (GP)
- Evolutionary Strategies
- Differential Evolution
- Etc.

..often invented (quasi-)independently by different people.

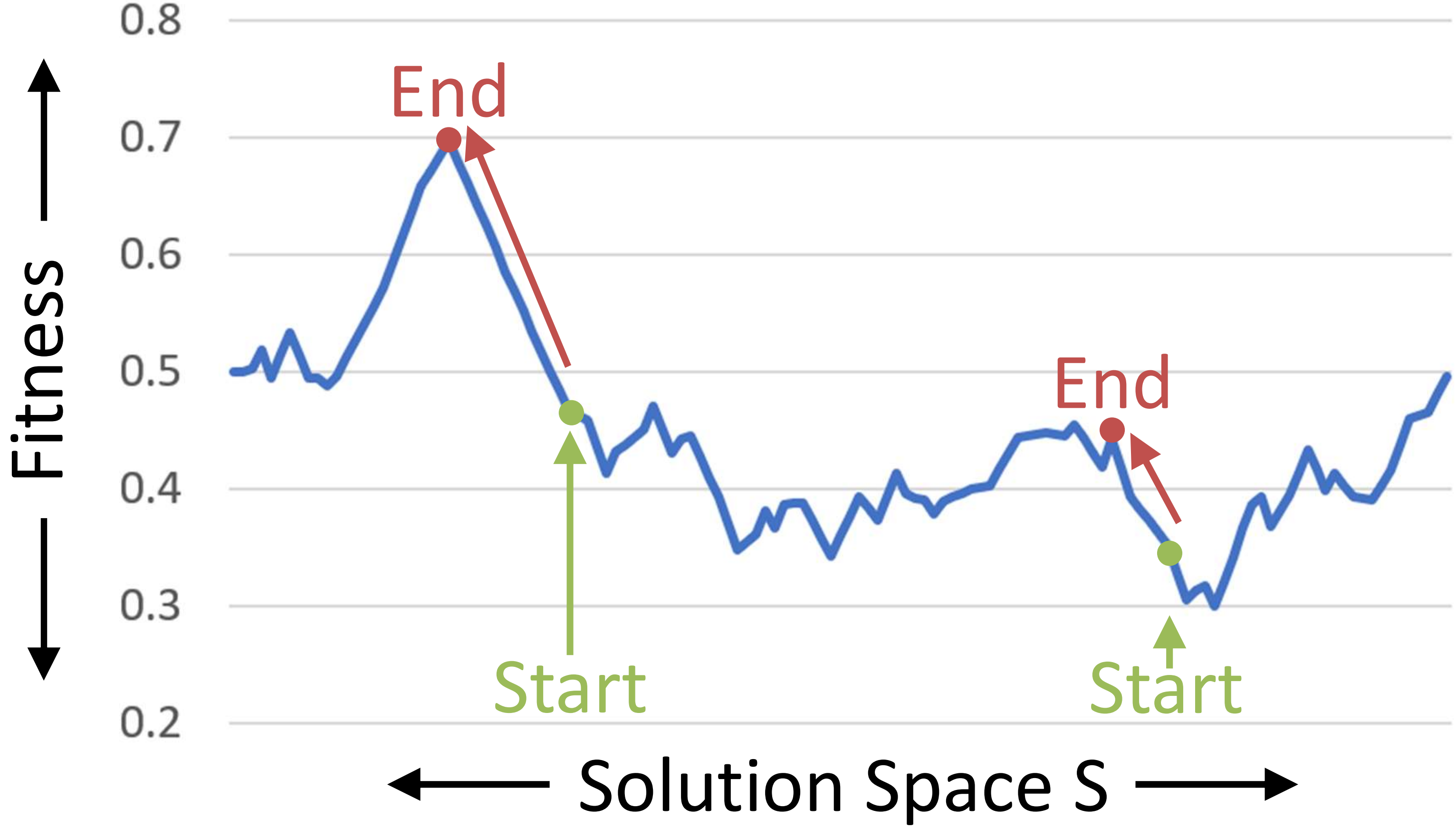
Individual-based Metaheuristics

In contrast to EC approaches, some metaheuristic approaches do *not* use a population of solutions; e.g.,

- Hillclimbing
- Simulated annealing
- Tabu search
- …

University of BRISTOL

Hill Climber



1. Make a random starting solution
2. Change it a little
3. If the new solution is better: keep it
   Else: discard it
4. If stuck: stop
   Else: go to 2

Consider: Is this a good diagram?

Simple Hill Climbing

Repeatedly switch to the first better solution found in the local neighbourhood of the current solution

```
start with any solution p chosen from solution space S
repeat:
     choose_a_better_neighbour(p) => q
     if q is None return p, else q => p

def choose_a_better_neighbour(p):
     for each neighbour q of p:
          if q is better than p:
               return q
     return None
```

Consider:

- *How do we pick the first solution p from S?*
- *What counts as the neighbourhood of a solution?*
- *In what order should we consider p's neighbours?*

Steepest Ascent Hill Climbing

Repeatedly switch to the best solution found in the local neighbourhood of the current solution

```
start with any solution p chosen from solution space S
repeat:
     choose_the_best_neighbour(p) => q
     if q is p return p, else q => p

def choose_the_best_neighbour(p):
     p => fittest
     for each neighbour q of p:
          if q is better than fittest: q => fittest
     return fittest
```

Consider:

- *Does the order in which we consider p’s neighbours still matter?*

Simple Tabu Search

Hillclimb, but don't consider solutions on a (finite) list of previously visited solutions

```
start with any solution p chosen from solution space S
p => best_so_far ; [p] => tabu_list
repeat:
     best(neighbours_of_p_not_in_tabu_list) => p
     if p is better than best_so_far: p => best_so_far
     add p to tabu_list
     if tabu_list too long:
          remove oldest item
     if time_to_stop: return best_so_far
```

Consider:

- *What's the significance of the tabu_list max length?*

Simulated Annealing

Hillclimb, but tolerate moves to worse solutions with a probability that is initially high, but decreases as more steps are made

```
start with any solution p chosen from solution space S

for step from 1 through max_step:
     pick_a_random_neigbour(p) => q
     if allow(fit(p), fit(q), temp(step)):
          q => p
return p

   def allow(p, q, T):
         return True if q>p or rand(1)<=exp(-(q-p)/T), else False
```

Consider:

- *The 'schedule' temp(step) needs to be defined*
- *It should return a high temperature for step 1*
- *And reduce it gradually as more steps are taken*

University of BRISTOL

The Explore/Exploit Tradeoff

- Recall: Breadth-first search *explores*, while depth-first *exploits*.
- Hill Climbers *exploit* current local knowledge of the search space – not even interested in fully exploring the local neighbourhood.
- Steepest Ascent Hill Climbers do explore the local neighbourhood fully before *greedily* exploiting the best local next step.
- Tabu is a little more *exploratory* than regular hill climbing. It keeps a record of solutions that have not year been fully exploited.
- Simulated Annealing starts off more *exploratory* (at high temps) but gets more *exploitative* over time (as temp falls).

University of BRISTOL

Going Populational

- All the algorithms that we've looked at work with only *one* hill climber moving across the solution space.
- By contrast, Genetic Algorithms maintain a *population* of solutions to help balance the explore/exploit trade-off.
- A population of solutions allows exploration to take place in *multiple* parts of the solution space.
- Competition & crossover allow success in one part of the space to be exploited by the rest of the population

Example Questions

- What's the difference between a stochastic algorithm and deterministic algorithm? *[1 mark]*
- Why does the temperature of a simulated annealing algorithm start off high and get lower over time? *[2 marks]*
- If a hill-climber is stuck, what can you infer about the solution that it has found? *[3 marks]*
- Explain the explore/exploit trade-offs made by best-first search, tabu search & simulated annealing. *[8 marks]*

University of BRISTOL

Thank you!