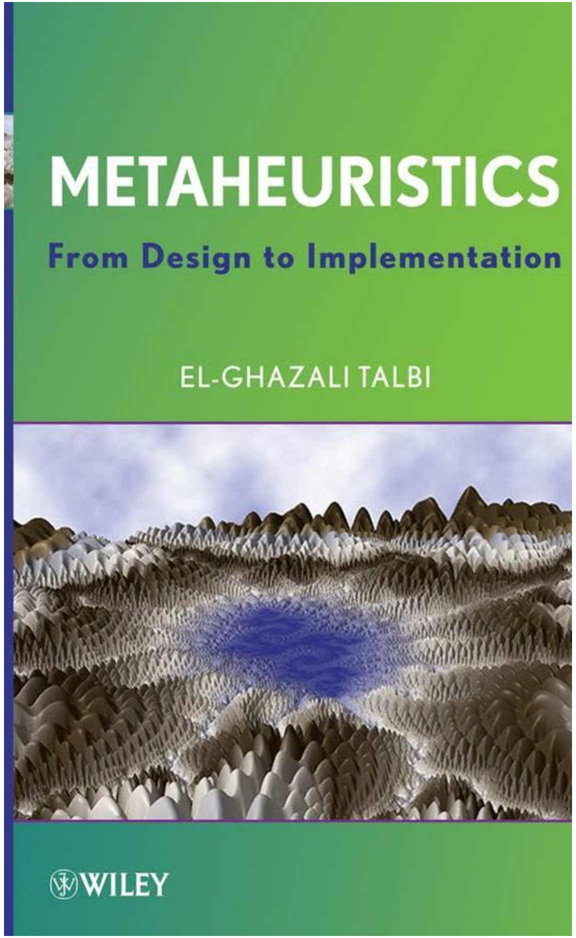


CHAPTER 3.4 AND 3.5

Sara Gestrelius

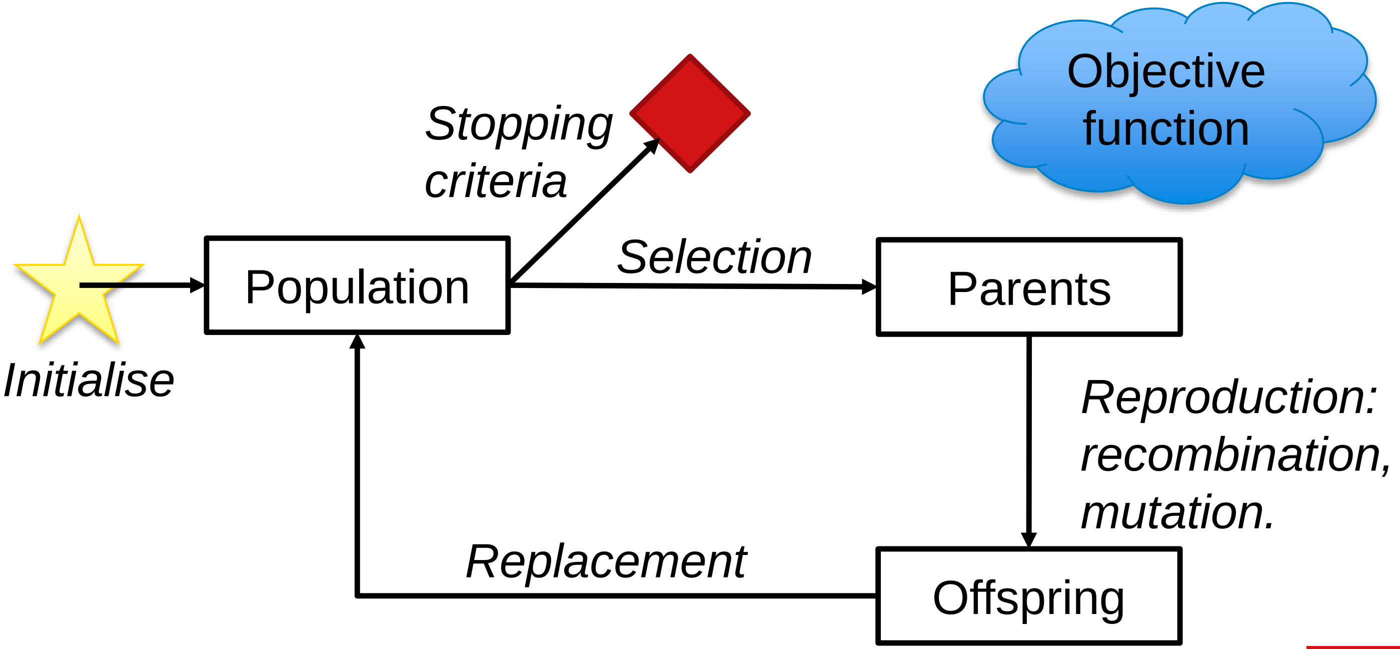
PART OF RI. SE

SWEDISH ICT SICS

3.4 OTHER EVOLUTIONARY ALGORITHMS

Estimation of Distribution algorithms
Differential Evolution
Coevolutionary algorithms
Cultural algorithms

LAST TIME: EVOLUTIONARY ALGORITHMS



SWEDISH ICT SICS

ESTIMATION OF DISTRIBUTION ALGORITHMS

Main idea
Variable interaction
Example: PBIL

ESTIMATION OF DISTRIBUTION ALGORITHMS

MAIN IDEA

- The population is represented by a probability distribution that is used to generate new individuals.
- Not yet competitive compared to more traditional metaheuristics.

Population of individuals

0 0 1 0 1 0 1

1 1 0 0 0 0 0

1 0 0 0 1 1 1

1 0 0 0 1 1 0

0 1 0 0 1 1 0

Probability distribution

P = 0.6 0.4 0.2 0.0 0.8 0.6 0.4

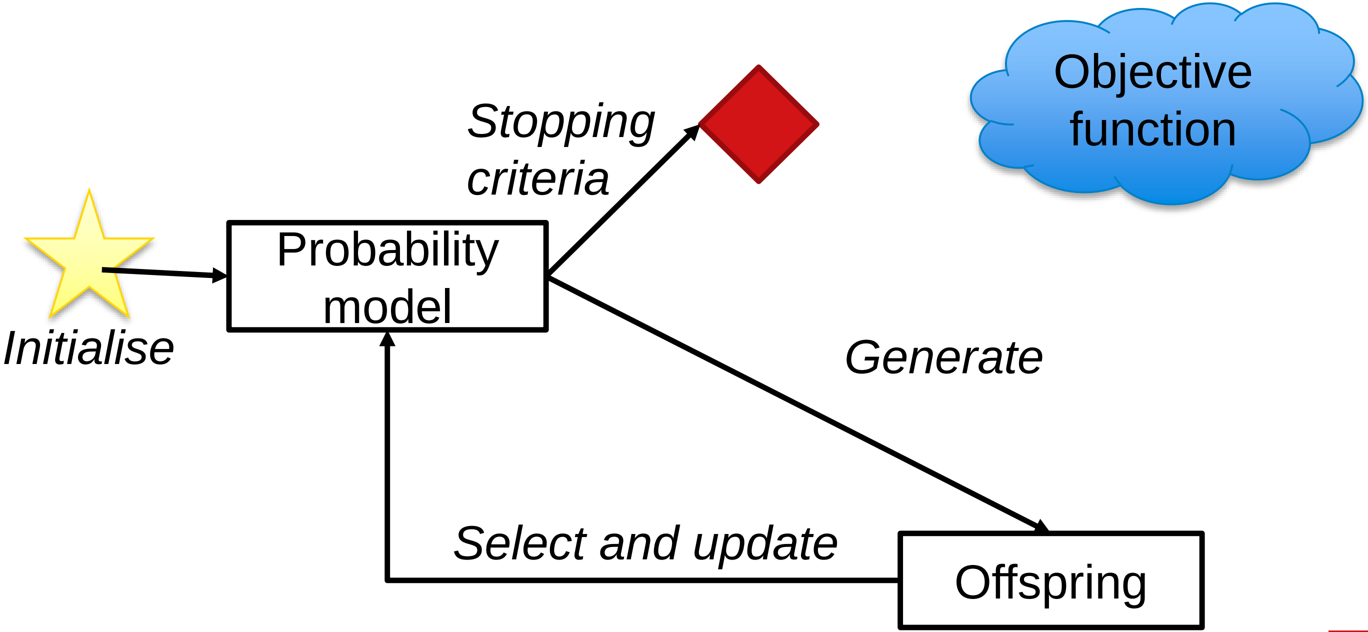
Example: A population of binary arrays can be represented by an array of probabilities that that entry is a 1.

MAIN IDEA

Algorithm 3.4 Template of the EDA algorithm

t = 1;
Generate randomly a population of *n* individuals;
Initialise a probability model *Q(x)*;
While Termination criteria are not met **Do**
    Create a population of *n* individuals by sampling from *Q(x)*;
    Evaluate the objective function for each individual;
    Select *m* individuals according to a selection methods;
    Update the probabilistic model *Q(x)* using selected population and *f()* values;
    t=t+1
End While
Output: Best found solution or set of solutions.

MAIN IDEA


Objective function
Stopping criteria
Probability model
Initialise
Generate
Select and update
Offspring

ESTIMATION OF DISTRIBUTION ALGORITHMS

VARIABLE INTERACTION

Sometimes the variables interact. Then this should be included in the probability model.

1. **Univariate EDA**: No interaction between the problem variables are taken into account.
2. **Bivariate EDA**: Interactions between two variables defines the probability model
3. **Multivariate EDA**: Interactions between many variables defined the probability model.

EXAMPLE: POPULATIONBASED INCREMENTAL LEARNING, PBIL

Algorithm 3.5 Template of the PBIL algorithm

Initial distribution D=(0.5, … , 0.5);
Repeat
    Repeat /*Generate population P of size *n**/
        Repeat /*For every entry *i* in D*/
            r=Uniform[0,1]
            If $r<D_i$ ***Then*** $X_i = 1$ ***Else*** $X_i = 0$
        Until *i*=|D|
    Until n
    Evaluate and sort the population P; /*Find best offspring*/
    Update the distribution $D=(1-\alpha)D+\alpha X_{best}$
Until Stopping Criteria

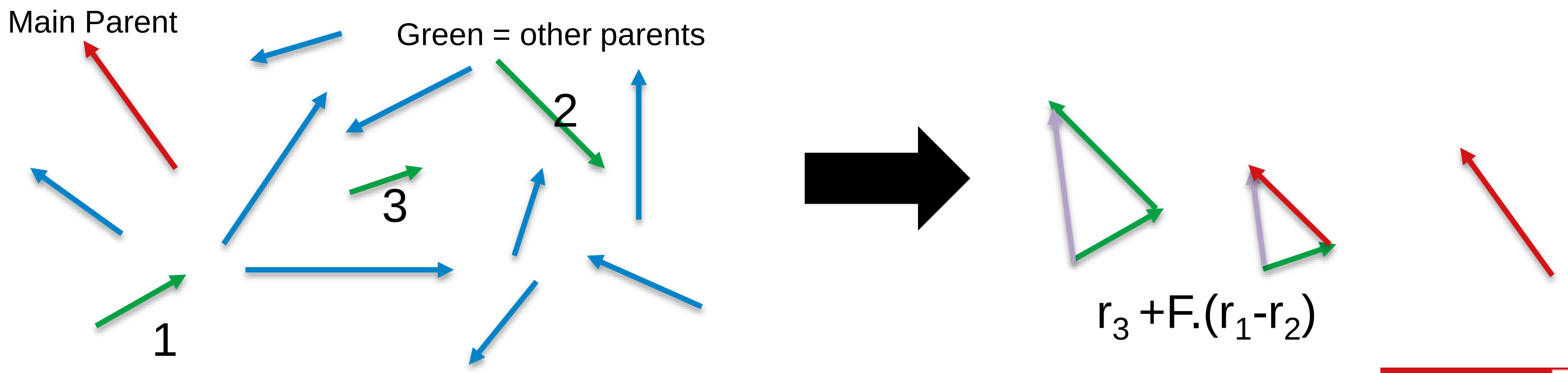
DIFFERENTIAL EVOLUTION

Main idea
Example: DE/rand/1/bin
Repair strategies
Tuning

DIFFERENTIAL EVOLUTION
MAIN IDEA

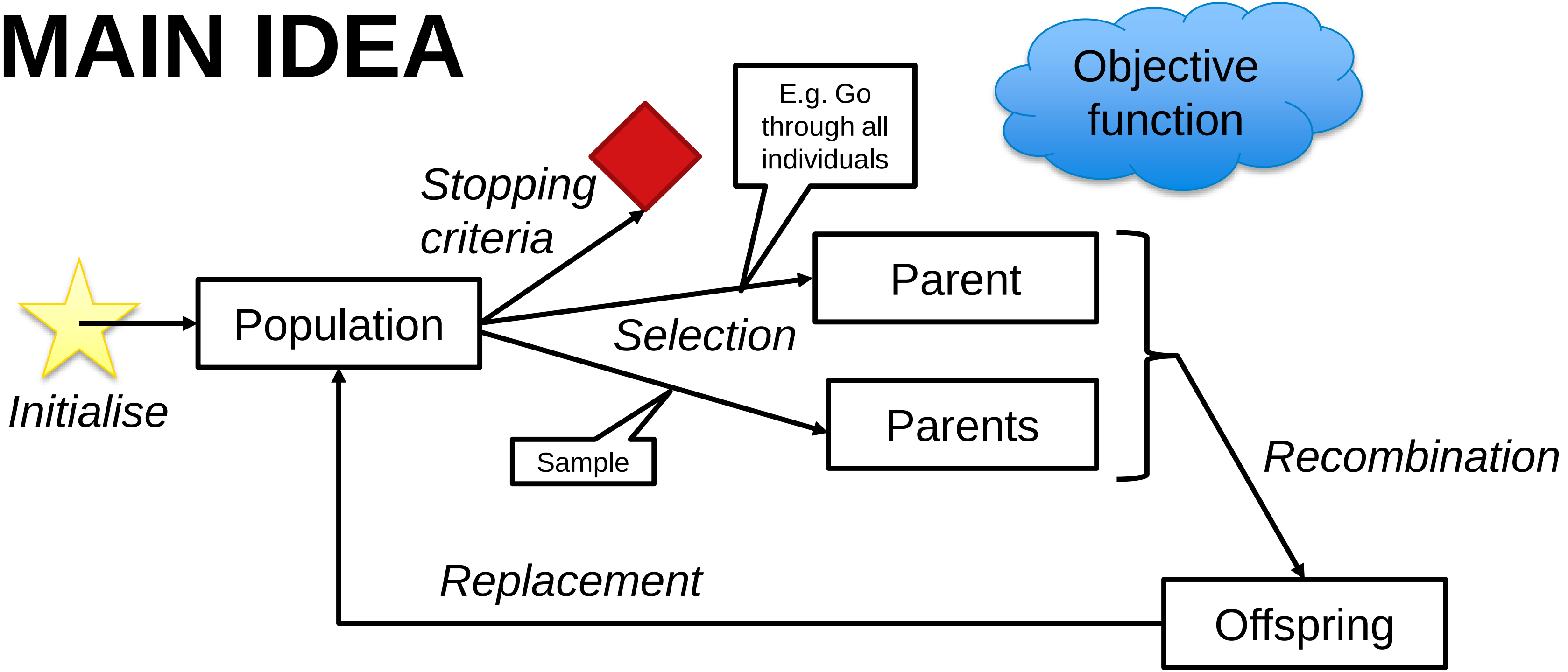
- When creating a new offspring: Use one main parent and a group of “other parents”. For each entry in the main parent vector, randomly choose whether to used main parent entry or new entry generated by the “other parents”.

- Very successful for continuous optimisation. All entries are bounded: $l_j \le x_{ij} \le u_j$



DIFFERENTIAL EVOLUTION

MAIN IDEA



DIFFERENTIAL EVOLUTION

EXAMPLE: DE/RAND/BIN

Algorithm 3.8 Template of the DE/rand/1/bin algorithm

Input: Parameters F (scaling factor) and CR (crossover constant).
Initialise the population (uniform random distribution);
Repeat
    For (i=1, i ≤ k, i++) **Do** /*Each individual*/
        /*Mutate and recombine***/**
        $j_{rand} = int(rand_i[0,1]*D)+1$;
        For (j=1, j ≤ D, j++) **Do**
            If $(rand_j[0,1]<CR)$ **or** $(j=j_{rand})$ ***Then***
                $u_{ij} = x_{r3j} + F.(x_{r1j} -.(x_{r2j})$
            Else
                $u_{ij} = x_{ij}$

        End For
        /Replace/
        If $f(u_i(t+1)) \leq f(u_i(t))$
            $x_i(t+1) = u_i(t+1)$
        Else
            $x_i(t+1) = x_i(t)$
    End For
Until Stopping Criteria
Output: Best population or solution found.

REPAIR STRATEGY

- Extreme strategies:
  1. Set variable to violated bound
  2. Randomly reinitialise value
- Intermediate strategy:
  1. Set to midway between old value and violated bound.

DIFFERENTIAL EVOLUTION
TUNING

- To get convergence:
  - Increase number of parents, decrease *F*.
- To increase convergence speed (and decrease robustness):
  - Increase *CR*
- DE much more sensitive to the value of *F* than the value of *CR*.

- From book: NP=10 times number of decision variables, CR=0.9, F=0.8

COEVOLUTIONARY ALGORITHMS

Main idea

The Rosenbrock function: competetive coevolution

The Rosenbrock function: cooperative coevolution

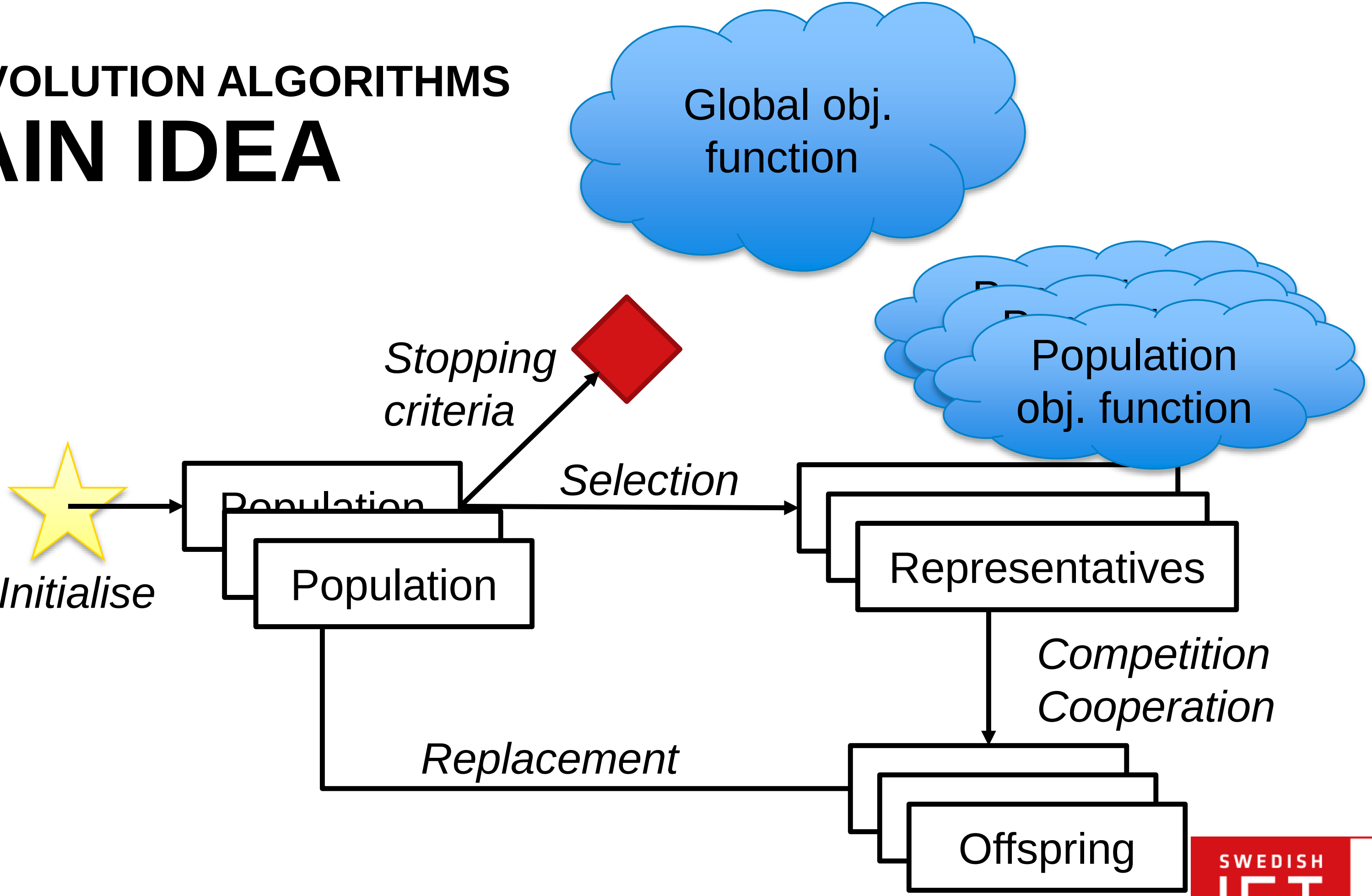
COEVOLUTION ALGORITHMS

MAIN IDEA

- Cooperative or competitive strategy involving different populations. The fitness of an individual in a given population depends on the fitness of individuals in other populations.

COEVOLUTION ALGORITHMS

MAIN IDEA



THE ROSENBROCK FUNCTION

$$f(x) = \sum_{i=1}^{n} (100(x_i - x_{i+1})^2 + (1 - x_i)^2) \ x \epsilon \mathbb{R}$$

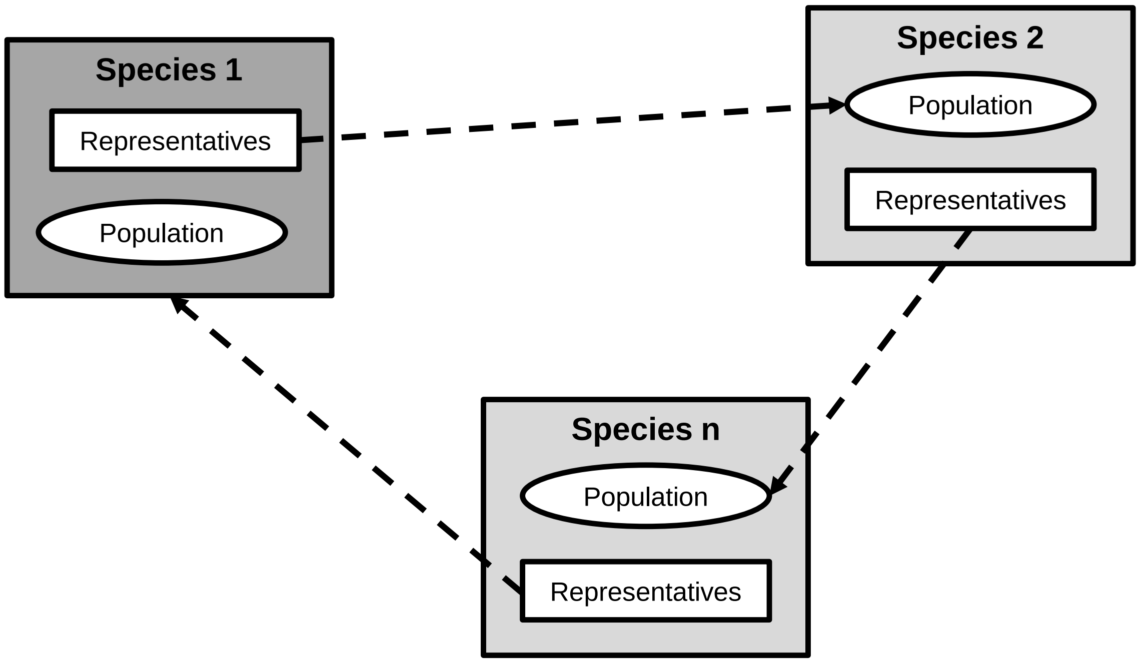
- n populations with the individuals x_i
- Objective functions $f_i(x_i; x_{i+1})$
- Interaction/communication graph G_{comm} defined by relations between subcomponents.

THE ROSENBROCK FUNCTION - COMPETITIVE

$$f(x) = \sum_{i=1}^{n} (100(x_i - x_{i+1})^2 + (1 - x_i)^2) \;\; x \epsilon \mathbb{R}$$

- Nodes x_i interact with nodes x_{i+1}.
- The coevlolving populations compete to minimize their local function.

COEVOLUTION ALGORITHMS
THE ROSENBROCK FUNCTION - COMPETITIVE



COEVOLUTION ALGORITHMS

THE ROSENBROCK FUNCTION - COOPERATIVE

$$f(x) = \sum_{i=1}^{n} (100(x_i - x_{i+1})^2 + (1 - x_i)^2) \;\; x \epsilon \mathbb{R}$$

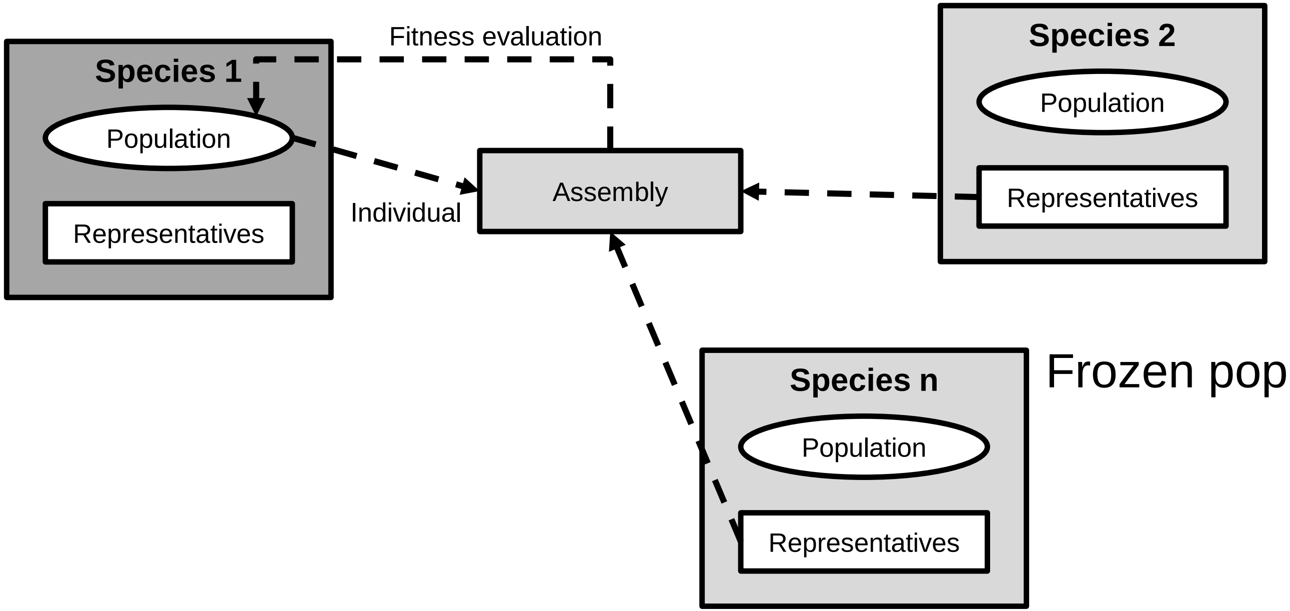
- Interaction graph fully connected.

Algorithm

1. Initialize by randomly connecting individuals of populations and find a best solution I_i^{best} for each population.
2. Repeat:
    1. For each population: combine each individual with best individuals of all other populations. Find new best solution in active population.
    2. For each population: match best individual with randomly selected individuals from other populations.
    3. For each population: Choose the best out of the two solutions.
3. Construct complete solution from all best individuals from each population.

COEVOLUTION ALGORITHMS

THE ROSENBROCK FUNCTION - COOPERATIVE



Frozen populations

CULTURAL ALGORITHMS

Main idea

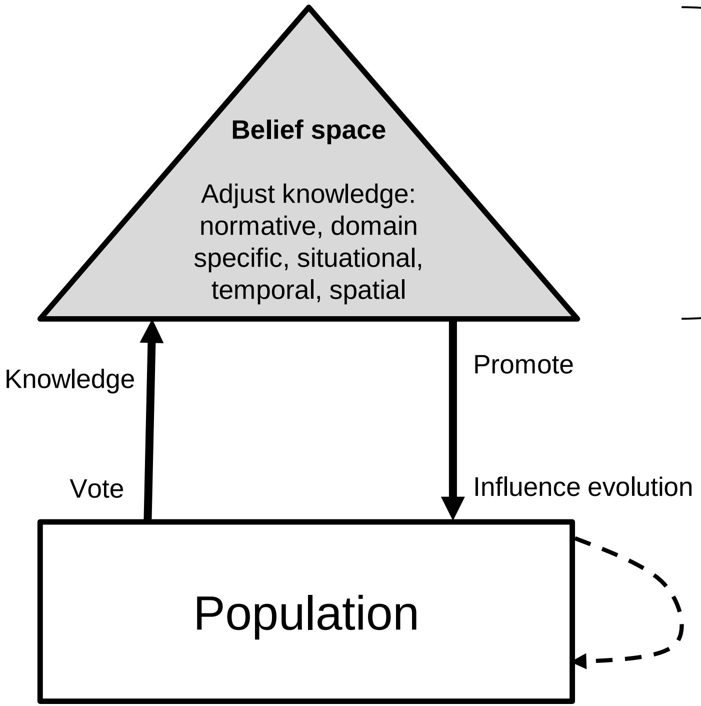
MAIN IDEA

- Two main elements:
  1. A **population space** at the mico-evolutionary level
  2. A **belief space** at the macro-evolutionary level

- Good when solutions require extensive domain knowledge

CULTURAL ALGORITHMS

MAIN IDEA



Symbolic resoning:

- Logic- and rule based reasoning models
- Schemata
- Graphical models
- Semantic networks
- Version spaces

Evolution of the population
(e.g. evolutionary algorithm)

CULTURAL ALGORITHMS

MAIN IDEA

Algorithm 3.9 Template of the cultural algorithm

Initialise the population *Pop(0)*;
Initialise the belief *BLF(0)*;
t = 0;
Repeat
    Evaluate the population *Pop(t)*;
    Adjust(*BLF(t)*, *Accept(Pop(t))*);
    Evolve(*Pop(t+1)*, *Influence(BLF(t))*);
    t=t+1
Until Stopping criteria
Output: Best found solution or set of solutions.

3.4 SCATTER SEARCH

Scatter search

Path relinking

SCATTER SEARCH

Main idea

Components

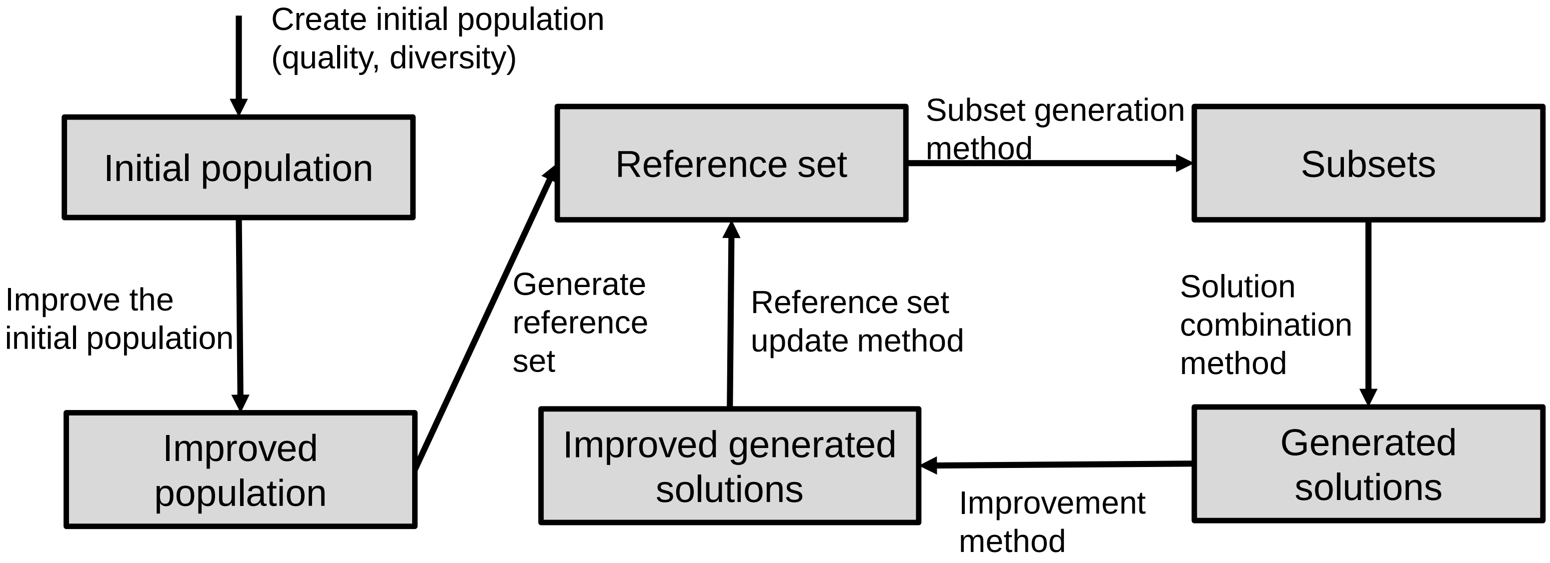
SCATTER SEARCH

MAIN IDEA

- Very many parents (aka reference set). Also, many parents can be used to generate new solutions.
- F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156-166, 1977.

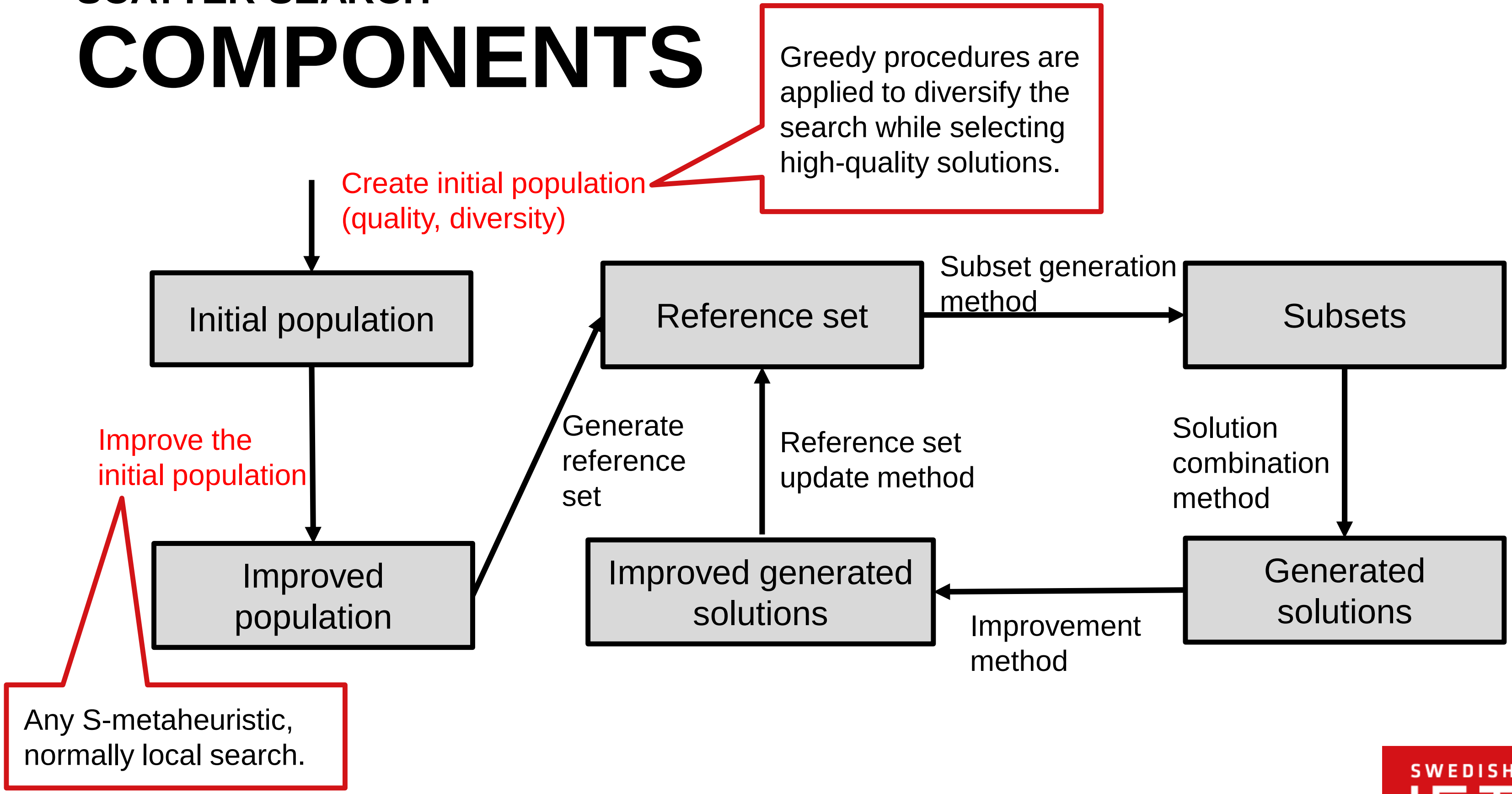
SCATTER SEARCH

MAIN IDEA



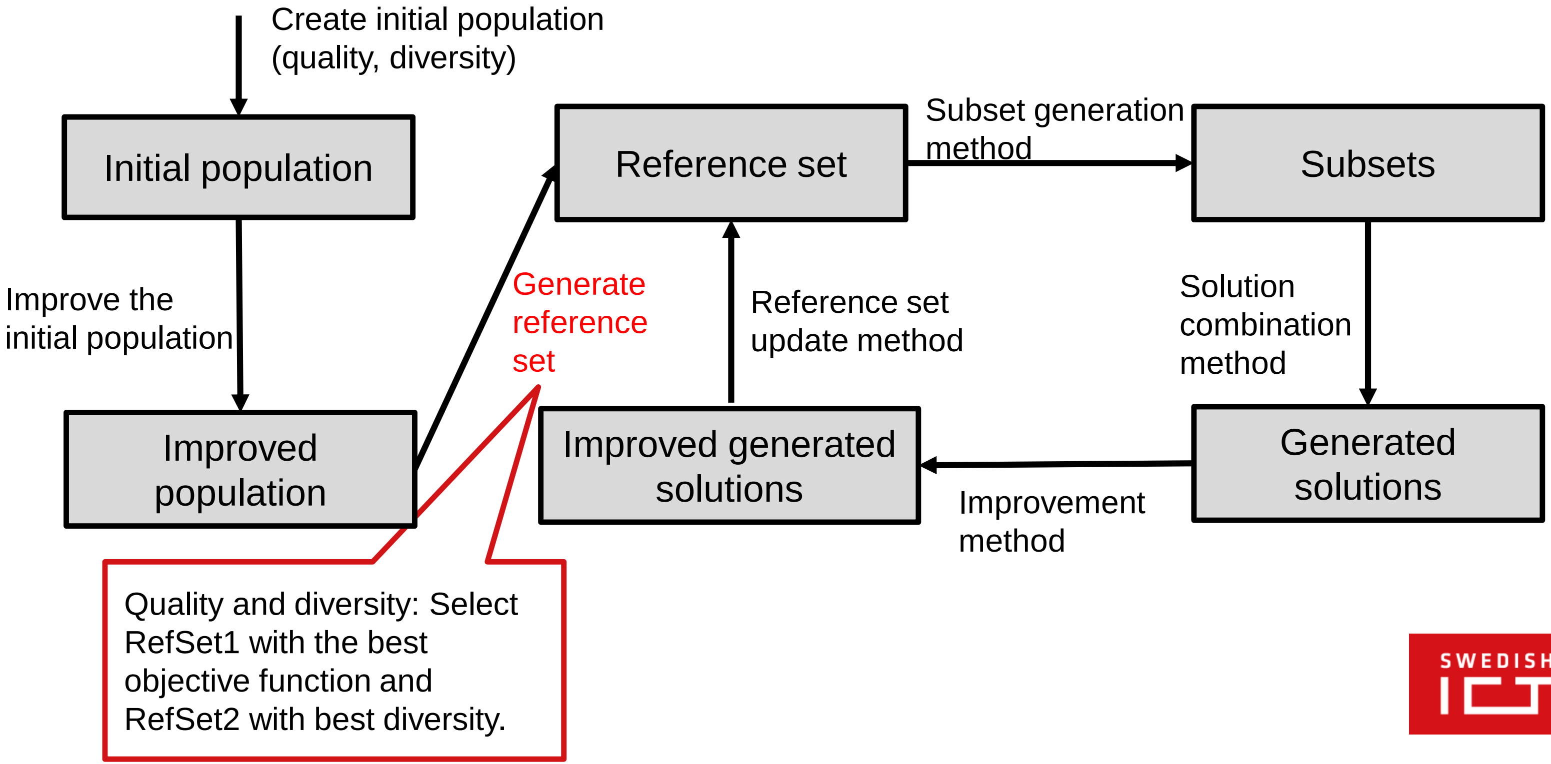
SCATTER SEARCH

COMPONENTS

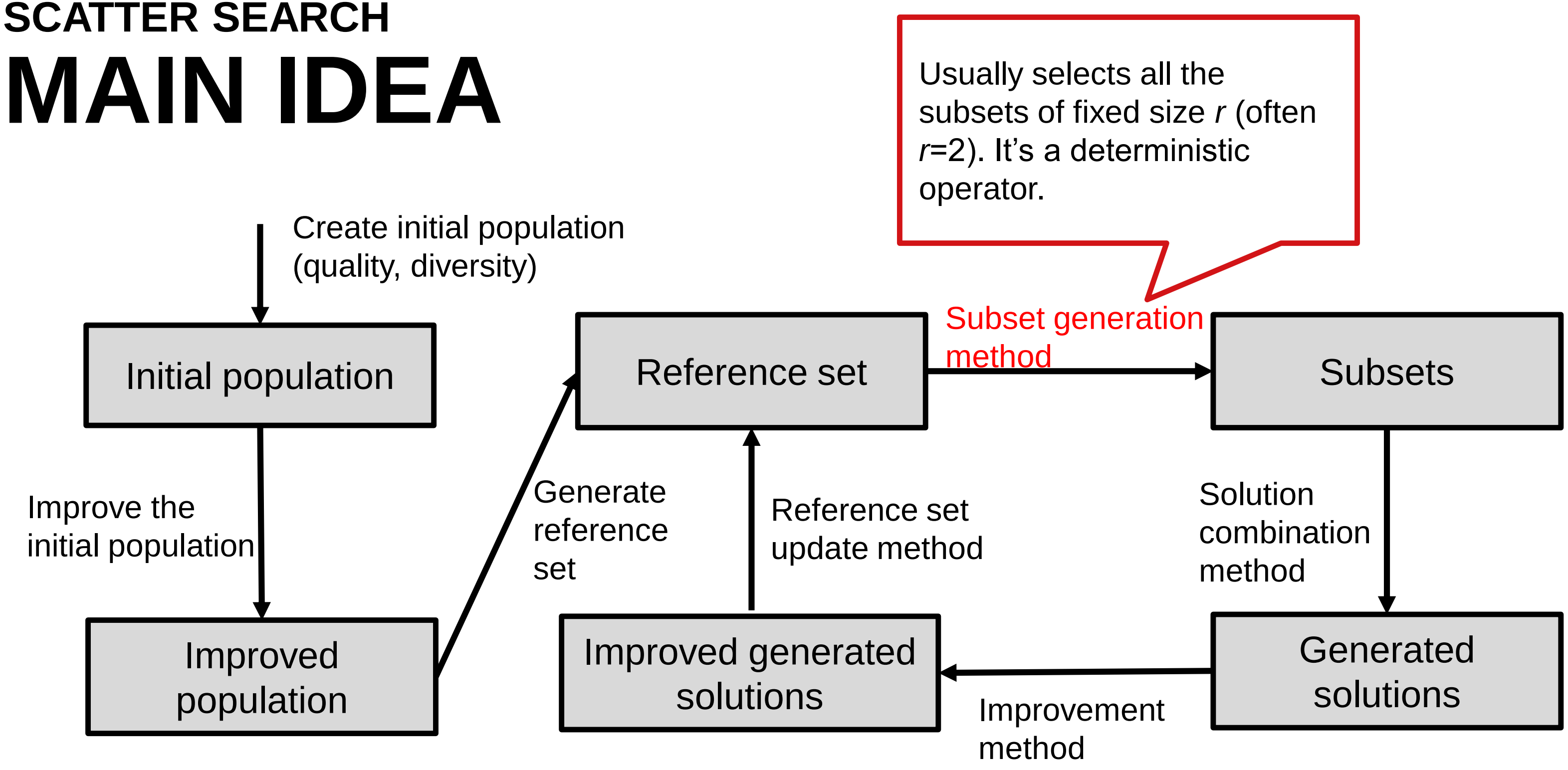


SCATTER SEARCH

COMPONENTS



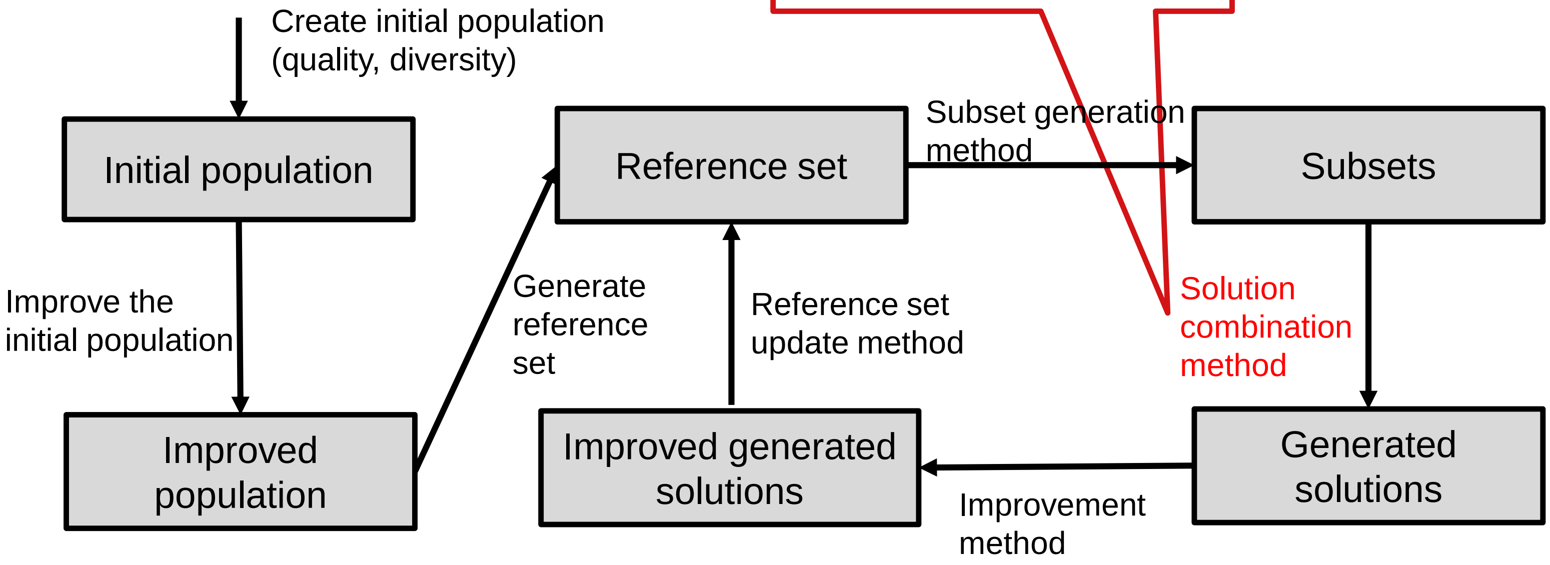
SCATTER SEARCH
MAIN IDEA



SCATTER SEARCH

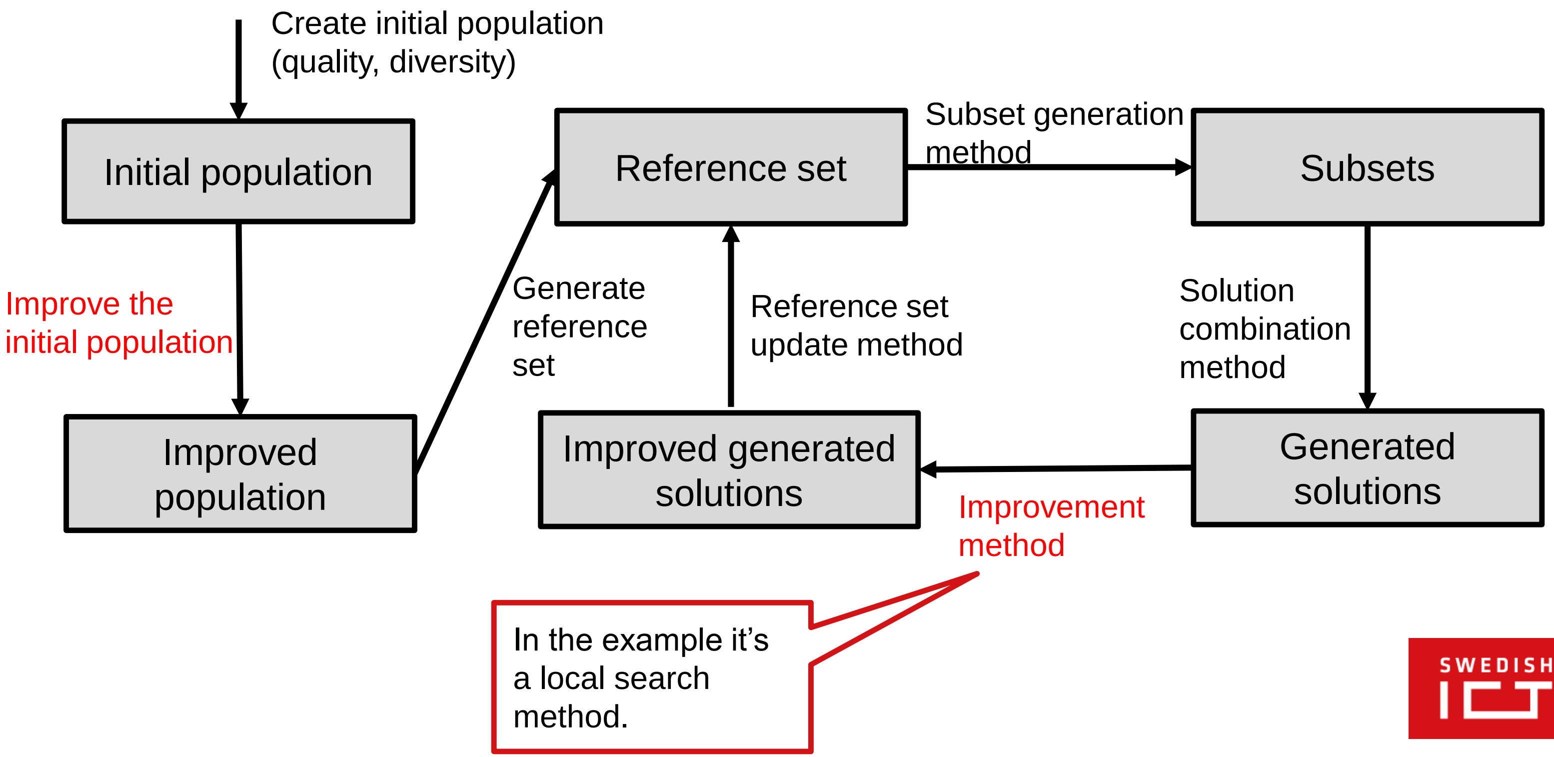
MAIN IDEA

In the example it's a cross-over method. Pretty much anything that can take a subset and turn it into a solution.



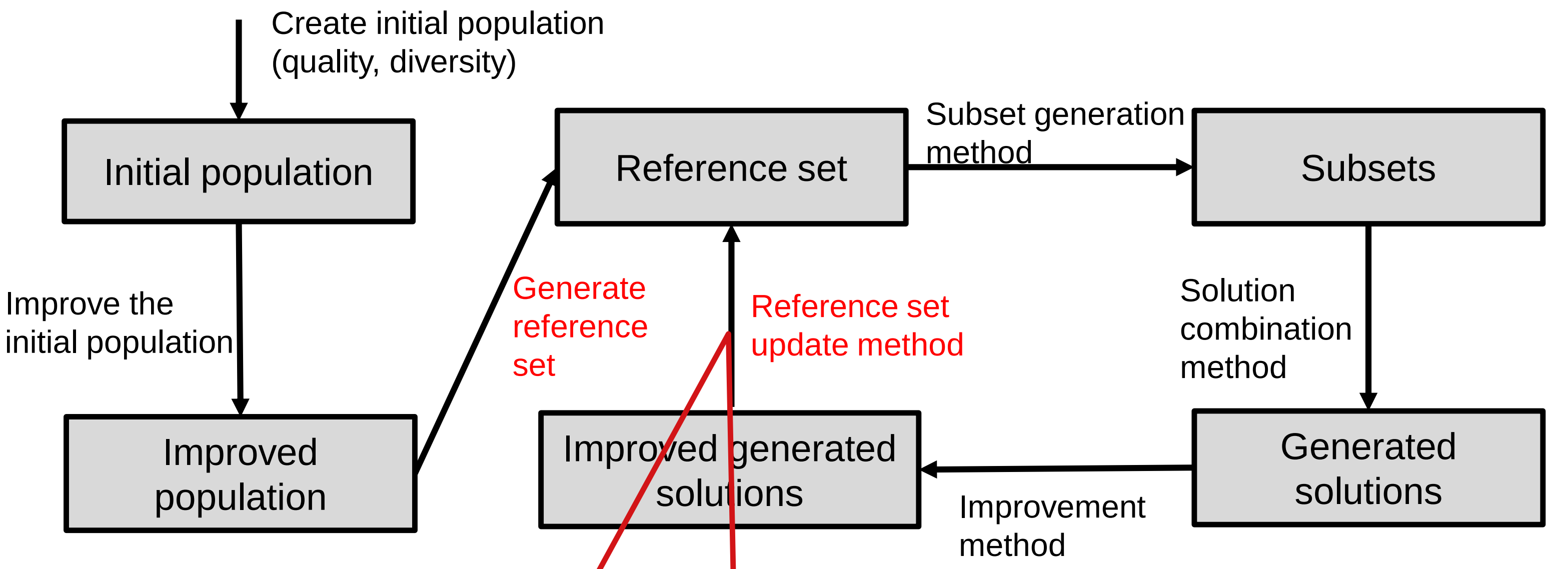
SCATTER SEARCH

MAIN IDEA



SCATTER SEARCH

MAIN IDEA



In example: use same method as in “Generate reference set”. Use solutions in Reference set and Improved generated solutions.

PATH RELINKING

Main idea
Components
Directions
Example: Binary search space

PATH RELINKING

MAIN IDEA

- Find a path between two elite solutions (*s* and *t*), return best solution on this path.
- The solutions on the path will generally share attributes with *s* and *t*.

PATH RELINKING

MAIN IDEA

Algorithm 3.11 Template of the basic PR algorithm

Input Starting solution *s* and target solution *t*
x=s;
While $dist(x,t) \neq 0$ **do**
    Find the best move *m* which decreases $dist(x+m,t)$;
    $x=x+m$; /*Apply the move *m* to the solution *x**/
Output: Best solution found in the trajectory between *s* and *t*.

PATH RELINKING

COMPONENTS

Algorithm 3.11 Template of the basic PR algorithm

Input Starting solution *s* and target solution *t*
x=s;
While *dist(x,t)≠0* **do**
    Find the **best** move *m* which decreases *dist(x+m,t);*
    x=x+m; /*Apply the move *m* to the solution *x**/
Output: Best solution found in the trajectory between *s* and *t.*

- Point with minimum distance to t.
- Point with best/worst objective function value.
- Use history of search (tabu search).

PATH RELINKING

COMPONENTS

Algorithm 3.11 Template of the basic PR algorithm

Input Starting solution *s* and target solution *t*
x=*s*;
While $dist(x,t) \neq 0$ **do**
    Find the best move *m* which decreases $dist(x+m,t)$;
    $x=x+m$; /*Apply the move *m* to the solution *x**/
    If *x* is valid solution
        x= local optimum; /*S-metaheuristic*/
Output: Best solution found in the trajectory between *s* and *t*.

Intermediate operations: Do something at each step of the path construction. E.g. find local optimum if valid solution.

SWEDISH ICT SICS

PATH RELINKING

DIRECTION

Algorithm 3.11 Template of the basic PR algorithm

Input Starting solution *s* and target solution *t*
x=s;
While *dist(x,t)≠0* **do**
    Find the best move *m* which decreases *dist(x+m,t);*
    x=x+m; /*Apply the move *m* to the solution *x**/
Output: Best solution found in the trajectory between *s* and *t.*

How to chose which solution that should be s and which t?

- **Forward:** From worst solution to best solution.
- **Backward:** From best solution to worst solution.
- **Back and forward relinking:** Both directions constructed in parallel (might not be worth the effort).
- **Mixed:** Chose intermediate point as *t* and both ends as *s*. Compute both paths in parallel.

EXAMPLE: BINARY SEARCH SPACE

Move: Flip operator.

Distance: Hamming distance.

WWW.SICS.SE

PART OF RI.SE

SWEDISH ICT SICS