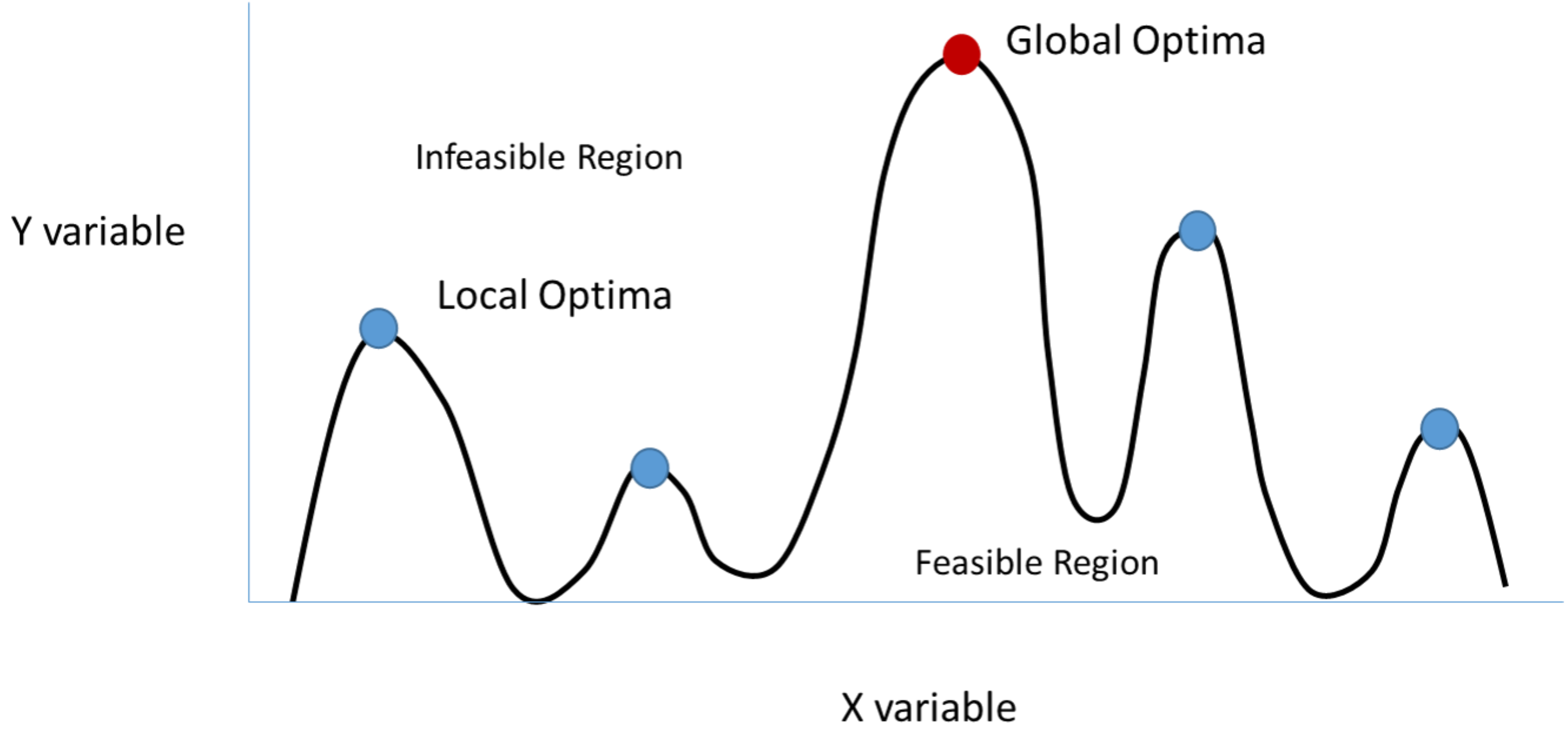
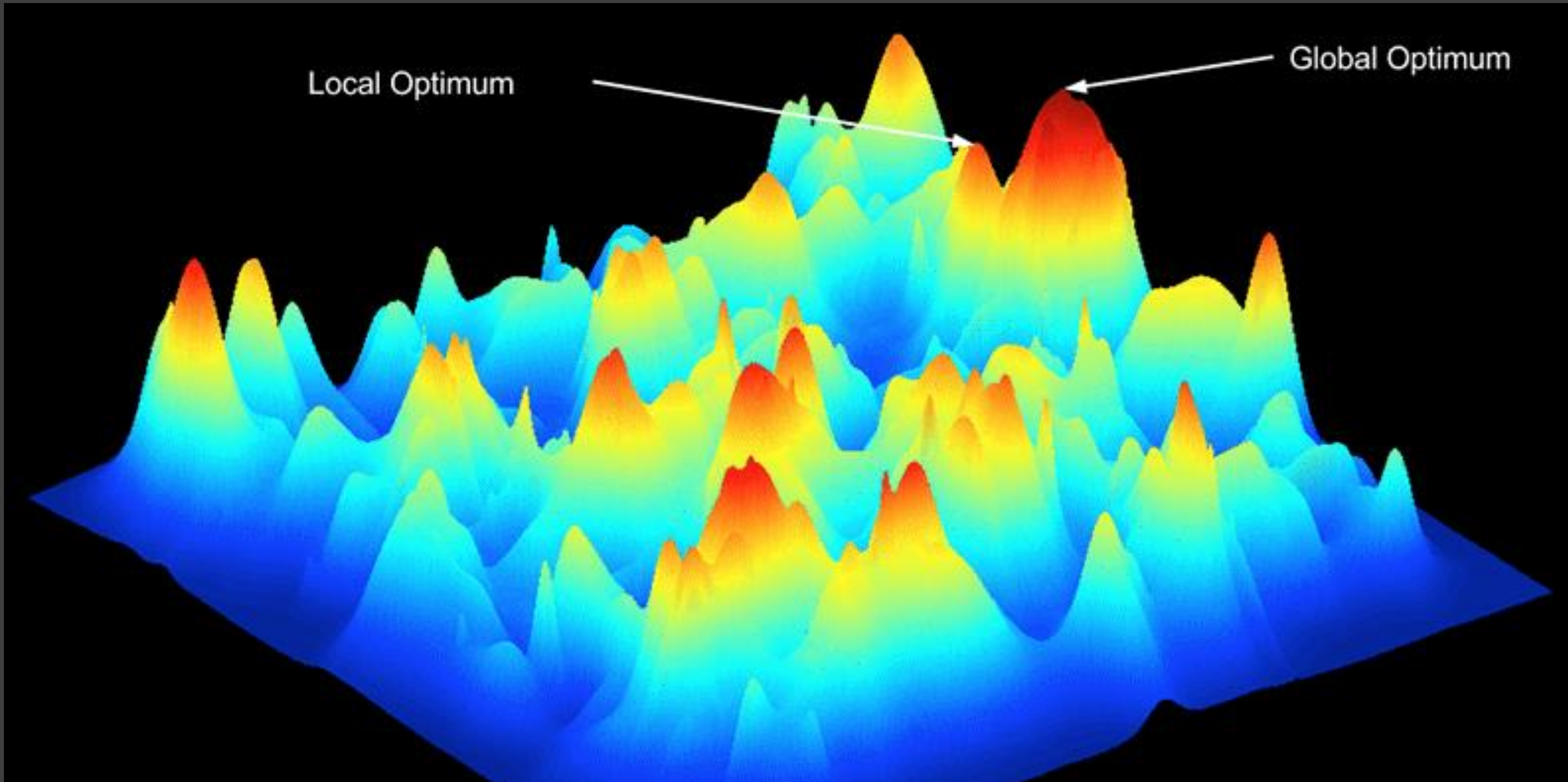
(Meta)Euristics methods



Local optimum vs global optimum



Limiti della ricercar locale

- Ricercare il minimo globale di una funzione di costo con molti gradi di libertà è un problema molto complesso, se questa funzione ammette un gran numero di minimi locali.
- Uno degli obiettivi principali dell'ottimizzazione è proprio quello di evitare di rimanere intrappolati in un minimo locale. Questo è uno dei limiti più grandi delle tecniche di ricerca locale.

From Local Search to Metaheuristics

- Local search is not able to escape from local minimum
- A way to solve it is the following
  - Stochastically explore only a subset of the neighborhood
  - Accept solutions that are worst than the previous

Metaheuristics

Heuristics

- "to find" (from ancient Greek "ευρίσκειν")

Meta-

- An abstraction from another concept
  - beyond, in an upper level
  - used to complete or add to
- E.g., meta-data = "data about data"

Metaheuristics

- "A heuristic around heuristics"

Popular metaheuristics

- Genetic Algorithms
- Simulated Annealing
- Tabu Search
- Scatter Search
- Ant Colony Optimization
- Particle Swarm Optimization
- Iterated Local Search
- Variable Neighborhood Search
- Adaptive Memory Programming
- …

ESCAPING LOCAL OPTIMA

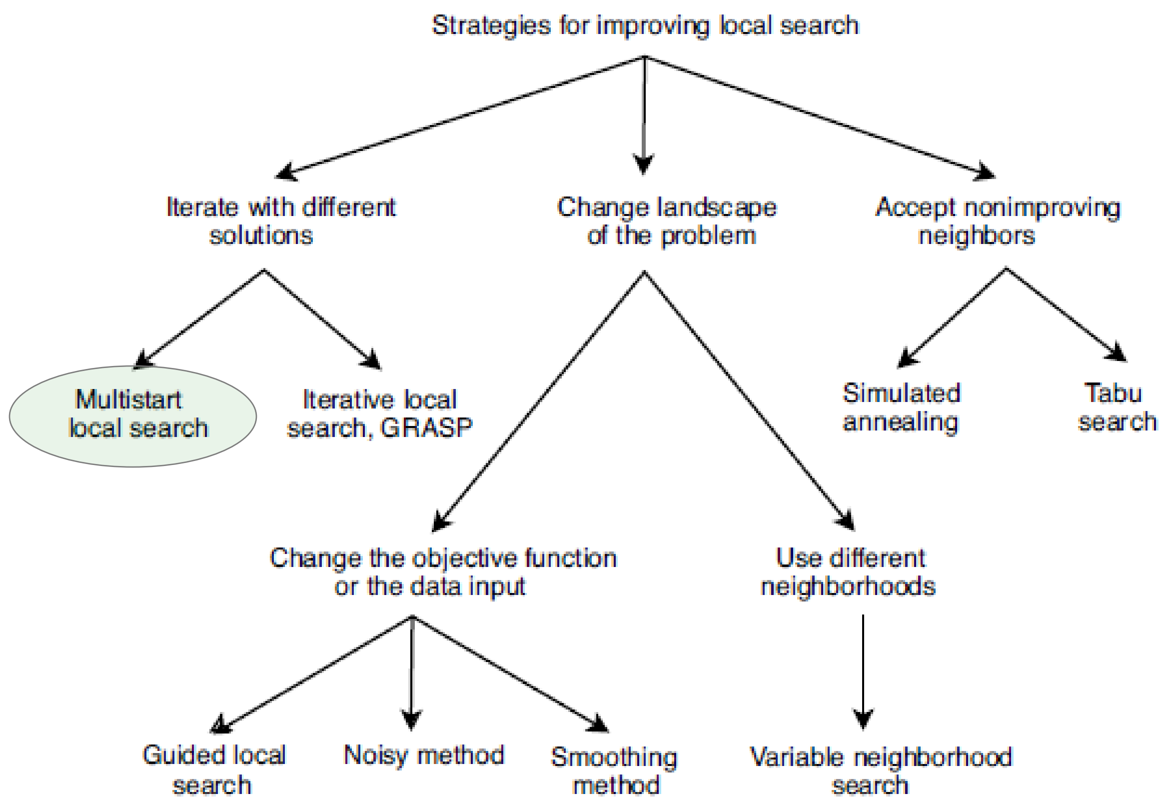


FIGURE 2.24 S-metaheuristic family of algorithms for improving local search and escaping from local optima.

Multistart Local Search

Generate more initial solutions and apply to each solution a local search procedure (until the given time is finished)

The performance gain is usually not so good due to the limited capability of each run to increase the starting solution

For instance 100 runs of 2-opt on a 100-city random geometric instance will be typically better than an average 3-opt

For 1000-city instance the best 100 runs of 2-opt is typically worse than the worst 100 runs of 3-opt

ESCAPING LOCAL OPTIMA

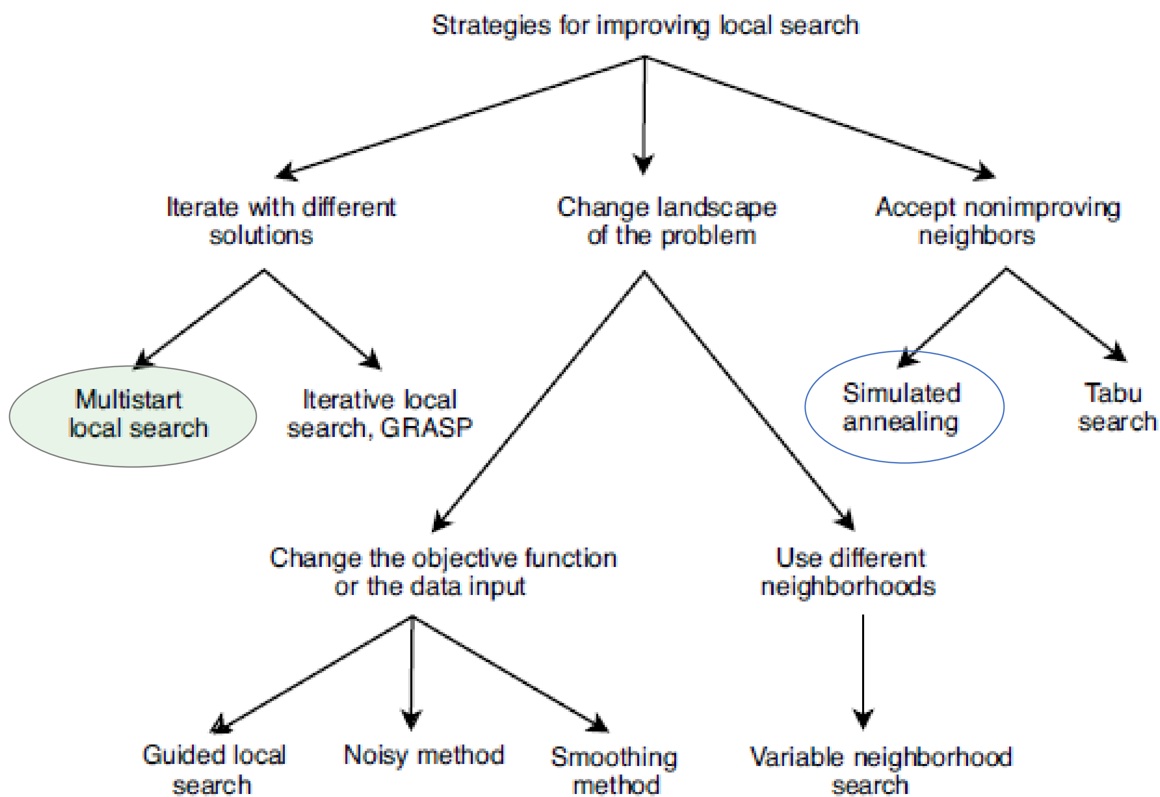


FIGURE 2.24 S-metaheuristic family of algorithms for improving local search and escaping from local optima.

Simulated Annealing

Meccanismo probabilistico che consente alla procedura di ricerca di fuggire da questi minimi locali.

L'idea è quella di accettare, in certi casi, oltre alle transizioni che corrispondono a miglioramenti nella funzione obiettivo, anche quelle transizioni che portano a peggioramenti nel valore di questa funzione di valutazione.

La probabilità di accettare tali deterioramenti varia nel corso del processo di ricerca, e discende lentamente verso zero.

Verso la fine della ricerca, quando vengono accettati solo miglioramenti, questo metodo diventa una semplice ricerca locale.

Tuttavia, la possibilità di transire in punti dello spazio di ricerca che deteriorano la soluzione ottima corrente, consente di abbandonare i minimi locali ed esplorare meglio l'insieme delle soluzioni ammissibili.

Simulated Annealing

Parametro

Prima soluzione

ALGORITMO DI SIMULATED ANNEALING

INIZIALIZZAZIONE DI C ED X

WHILE NOT (CONDIZIONE DI USCITA)

Parametro

    DO BEGIN

    FOR I:=1 TO L

        DO BEGIN

Funzione obiettivo

Sol. vicina

        GENERAZIONE DI Y DA X

        IF $f(Y) \leq f(X)$

        THEN X:=Y

        ELSE IF $e^{-(f(Y)-f(X))/C} > random(0,1)$

            THEN X:=Y

        END

    RIDUZIONE DEL PARAMETRO C

END

Probabilità di accettazione

ESCAPING LOCAL OPTIMA

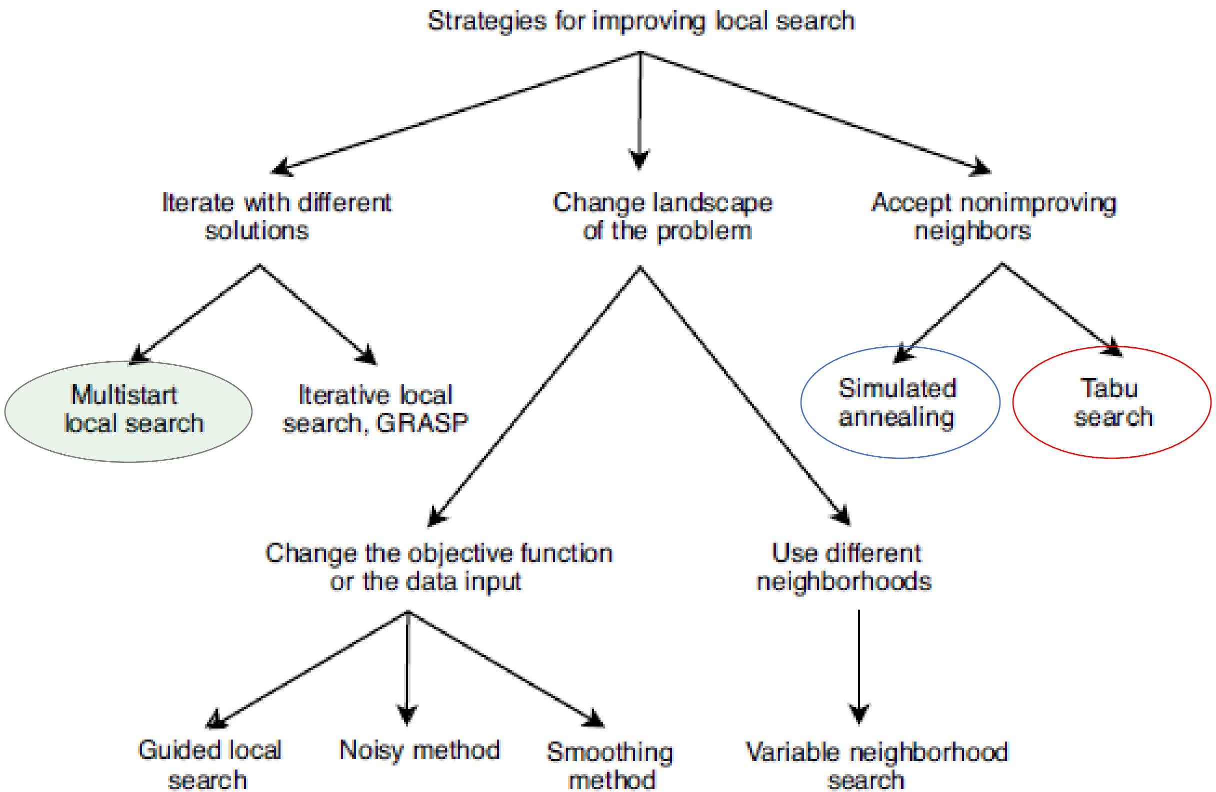


FIGURE 2.24 S-metaheuristic family of algorithms for improving local search and escaping from local optima.

TABU SEARCH

- **Taboo** (*English*): prohibited, disallowed, forbidden

- The tabu list constitutes “short-term memory”
- Size of tabu list (“tenure”) is finite
- Since solutions in tabu list are off-limits, it helps
  - escape local minima by forcing uphill moves (if no improving move available)
  - avoid cycling (up to the period induced by tabu list size)
- Solutions enter and leave the list in a FIFO order (usually)

TABU SEARCH

$s \leftarrow$ GenerateInitialSolution()
$TabuList \leftarrow \emptyset$
while termination conditions not met **do**
  $s \leftarrow$ ChooseBestOf($\mathcal{N}(s) \setminus TabuList$)
  Update($TabuList$)
endwhile

Fig. 3. Algorithm: Simple Tabu Search (TS).

TABU SEARCH

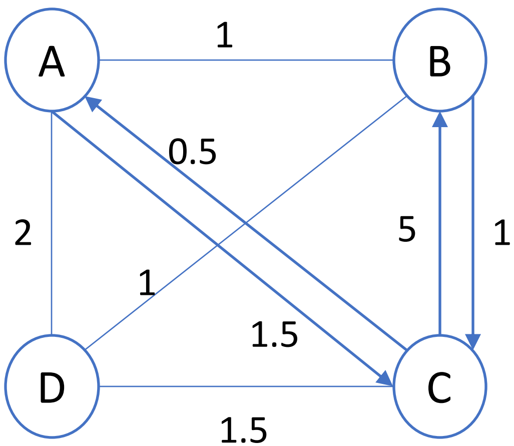
- Small tenure localizes search (intensification)
- Large tenure forces exploration of wider space (diversification)
- Tenure can change dynamically during search
- Size of tenure is a form of “long-term memory”

TABU SEARCH

- Storing complete solutions is inefficient
  - implementation perspective (storage, comparisons)
  - algorithm perspective (largely similar solutions offer no interesting information)
- Tabu search usually stores “solution attributes”
  - solution components or solution differences (“moves”)

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

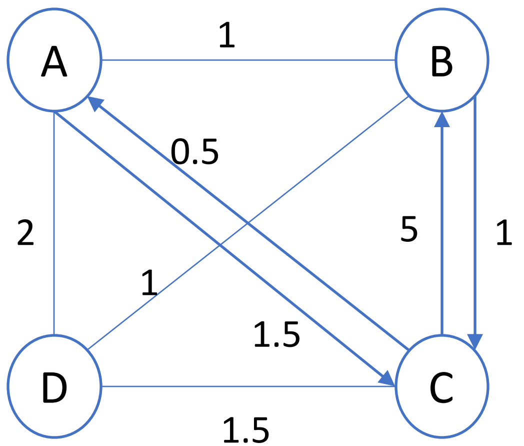
Solution Trajectory

A	B	C	D	A

5.5

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

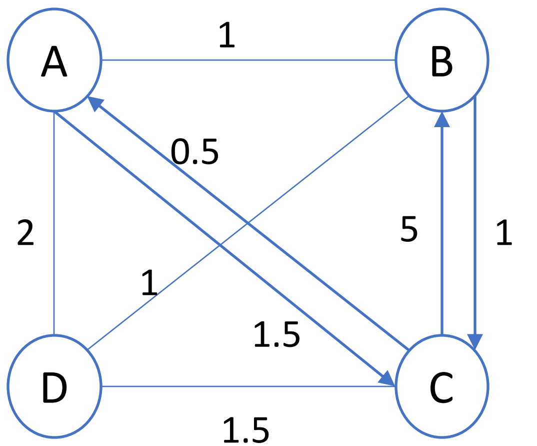
Solution Trajectory

A	B	C	D	A	**5.5**

AB	B	A	C	D	B	**5.0**
BC	A	C	B	D	A	**9.5**
CD	A	B	D	C	A	**4.0**
DA	D	B	C	A	D	**4.5**

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

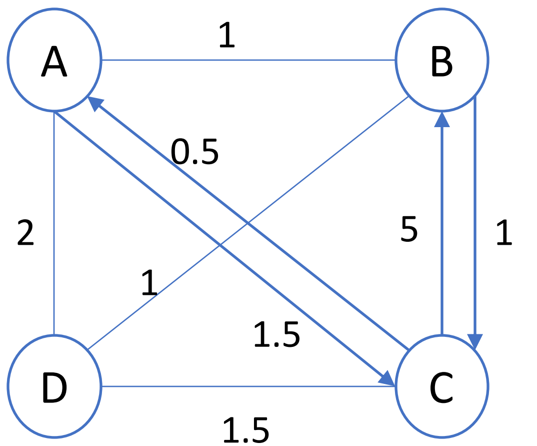
Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0

AB	B	A	C	D	B	5.0
BC	A	C	B	D	A	9.5
CD	A	B	D	C	A	4.0
DA	D	B	C	A	D	4.5

TABU SEARCH

Asymmetric TSP



Tabu List (tenure = 3)

CD

Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0

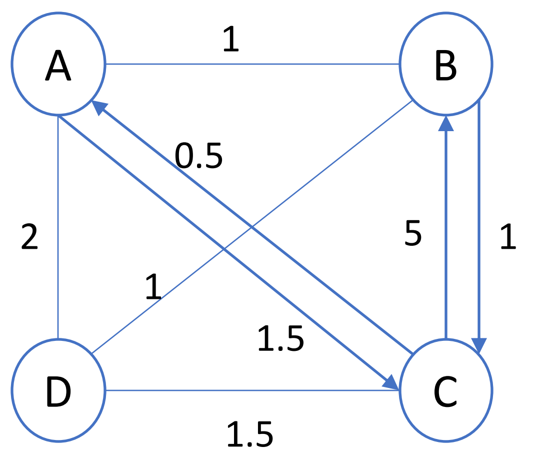
solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

CD

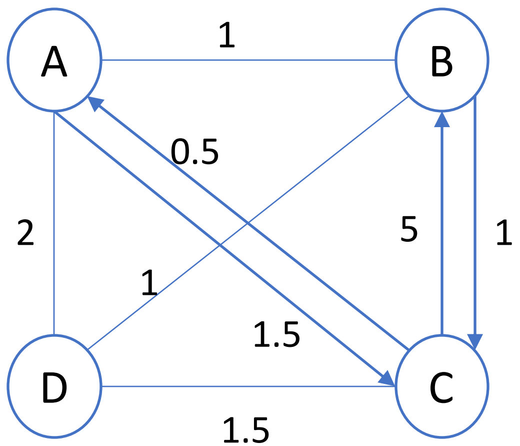
Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0

AB	B	A	D	C	A	5.0
BD	A	D	B	C	A	4.5
DC	A	B	C	D	A	5.5
CA	C	B	D	A	C	9.5

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

BD
CD

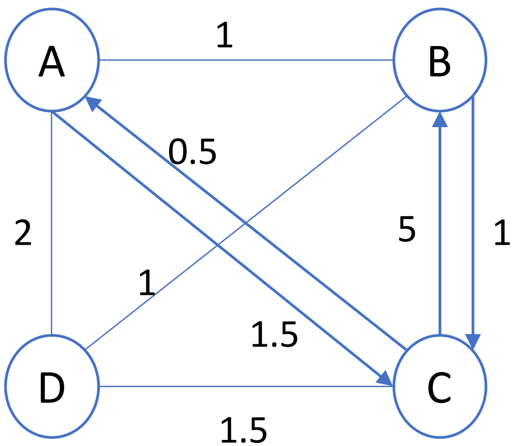
Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0
A	D	B	C	A	4.5

AB	B	A	D	C	A	5.0
BD	A	D	B	C	A	4.5
DC	A	B	C	D	A	5.5
CA	C	B	D	A	C	9.5

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

BD
CD

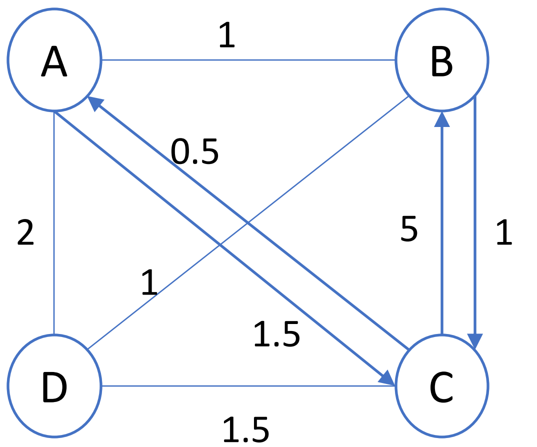
Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0
A	D	B	C	A	4.5

AD	D	A	B	C	D	5.5
DB	A	B	D	C	A	4.0
BC	A	D	C	B	A	9.5
CA	C	D	B	A	C	5.0

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List
(tenure = 3)

BD
CD

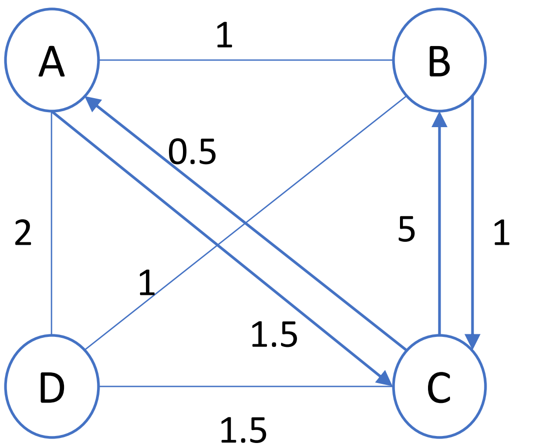
Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0
A	D	B	C	A	4.5

AD	D	A	B	C	D	5.5
DB	A	B	D	C	A	4.0
BC	A	D	C	B	A	9.5
CA	C	D	B	A	C	5.0

TABU SEARCH

Asymmetric TSP



Tabu List (tenure = 3)

CA
BD
CD

Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0
A	D	B	C	A	4.5
C	D	B	A	C	5.0

CD	D	C	B	A	D	9.5
DB	C	B	D	A	C	9.5
BA	C	D	A	B	C	5.5
AC	A	D	B	C	A	4.5

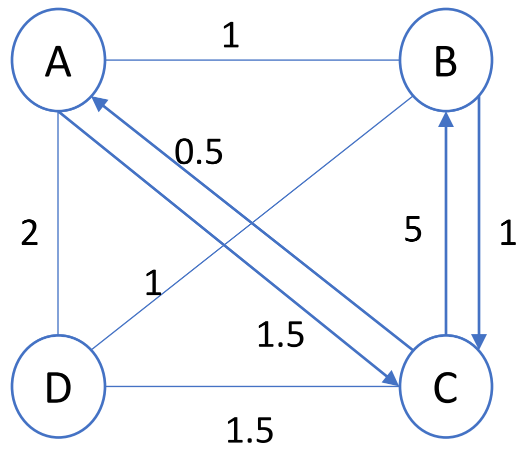
solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

CA
BD
CD

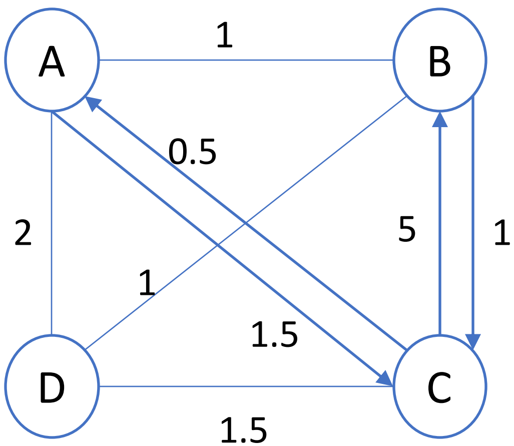
Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0
A	D	B	C	A	4.5
C	D	B	A	C	5.0

CD	D	C	B	A	D	9.5
DB	C	B	D	A	C	9.5
BA	C	D	A	B	C	5.5
AC	A	D	B	C	A	4.5

TABU SEARCH

Asymmetric TSP



solution = {tour}

move = {swap consecutive pair}

attributes = {moves}

Tabu List (tenure = 3)

BA
CA
BD

Solution Trajectory

A	B	C	D	A	5.5
A	B	D	C	A	4.0
A	D	B	C	A	4.5
C	D	B	A	C	5.0
C	D	A	B	C	5.5

CD	D	C	A	B	D	4.0
DA	C	A	D	B	C	4.5
AB	C	D	B	A	C	5.0
BC	B	D	A	C	B	9.5

ESCAPING LOCAL OPTIMA

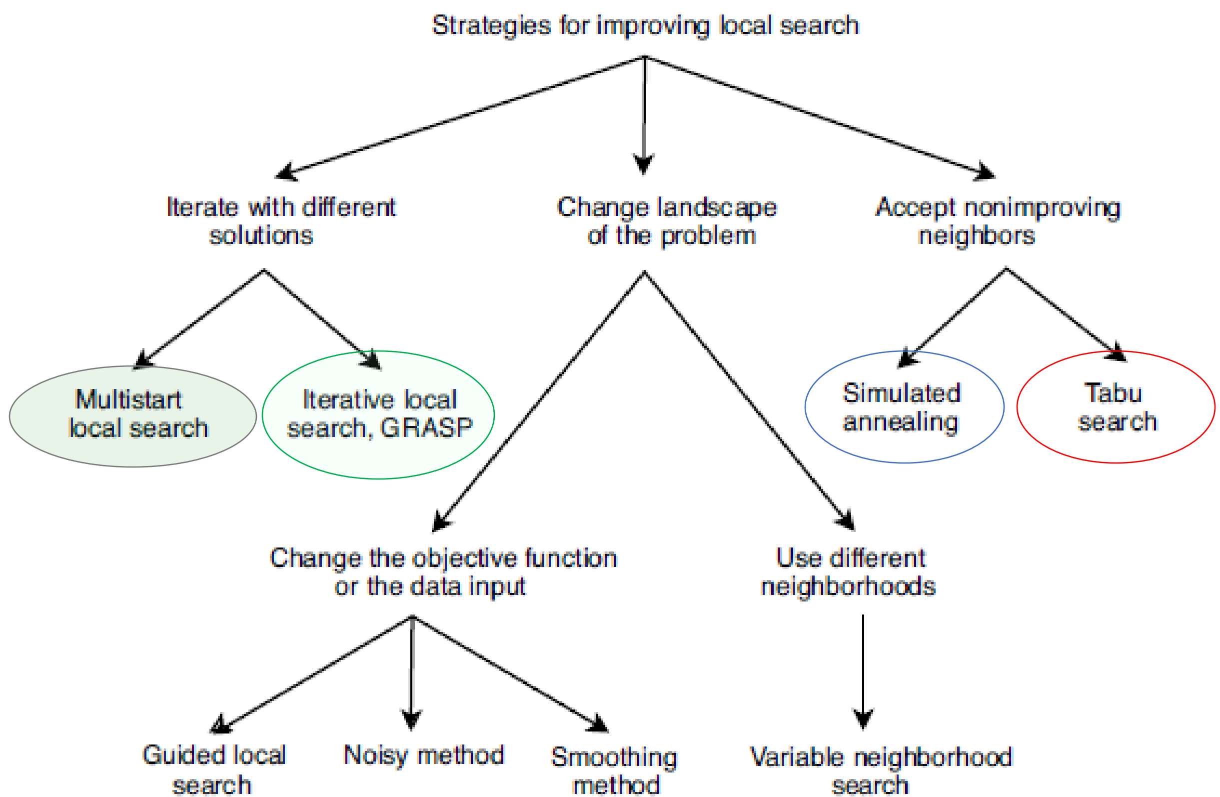


FIGURE 2.24 S-metaheuristic family of algorithms for improving local search and escaping from local optima.

ITERATED LOCAL SEARCH

ILS is a general-purpose "multi-restart" local search framework
Provides structure in selection of next initial point

$s_0 \leftarrow$ GenerateInitialSolution()
$\hat{s} \leftarrow$ LocalSearch(s_0)
while termination conditions not met **do**
    $s' \leftarrow$ Perturbation($\hat{s}$, *history*)
    $\hat{s'} \leftarrow$ LocalSearch(s')
    $\hat{s} \leftarrow$ ApplyAcceptanceCriterion($\hat{s}$, $\hat{s'}$, *history*)
endwhile

Fig. 11. Algorithm: Iterated Local Search (ILS).

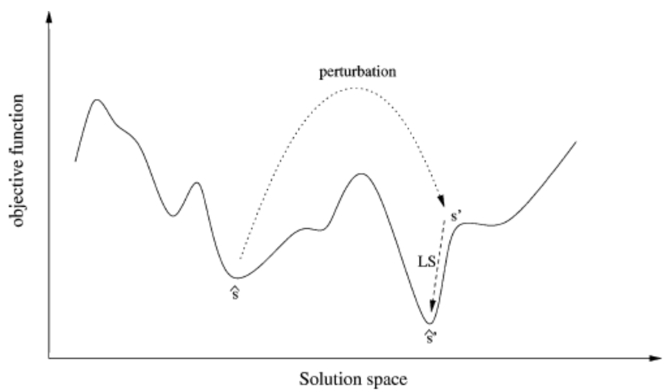


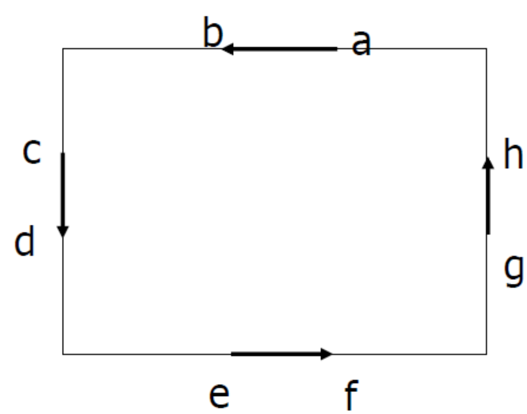
Fig. 12. A desirable ILS step: the local minimum $\hat{s}$ is perturbed, then LS is applied and a new local minimum is found.

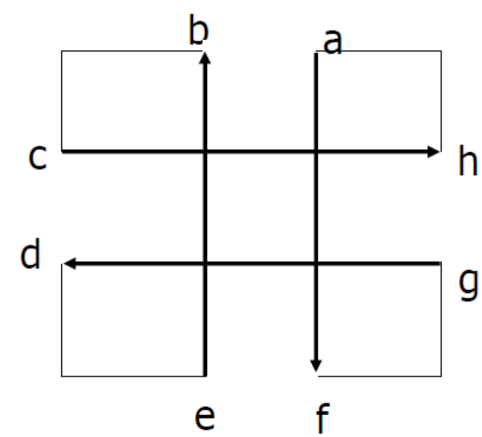
Perturbation() is non-deterministic (avoids cycling)

The "strength" of *Perturbation()* (i.e., how many solution feature changes are induced) varies along search process

ILS for TSP (perturbation)

Double bridge for its non-sequential nature can not be easily reverted by 3-opt or lin-kernighan





ESCAPING LOCAL OPTIMA

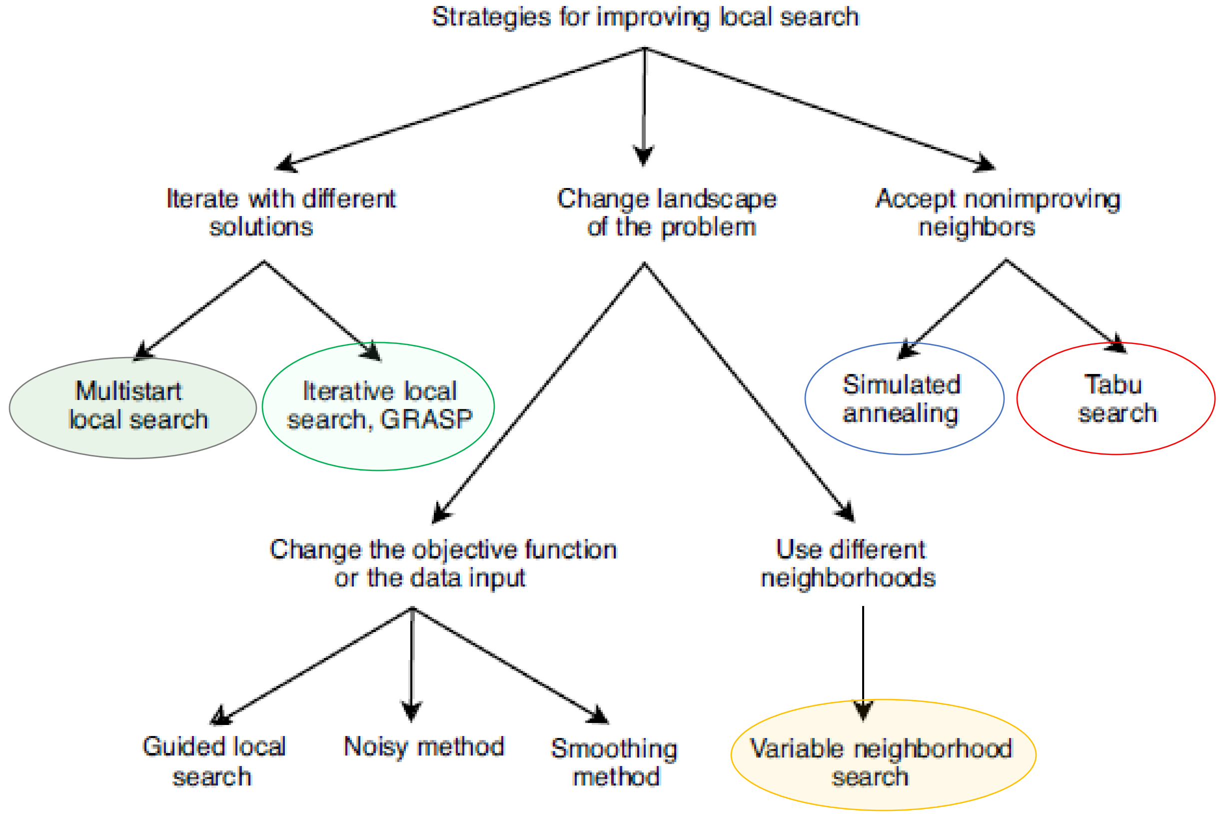


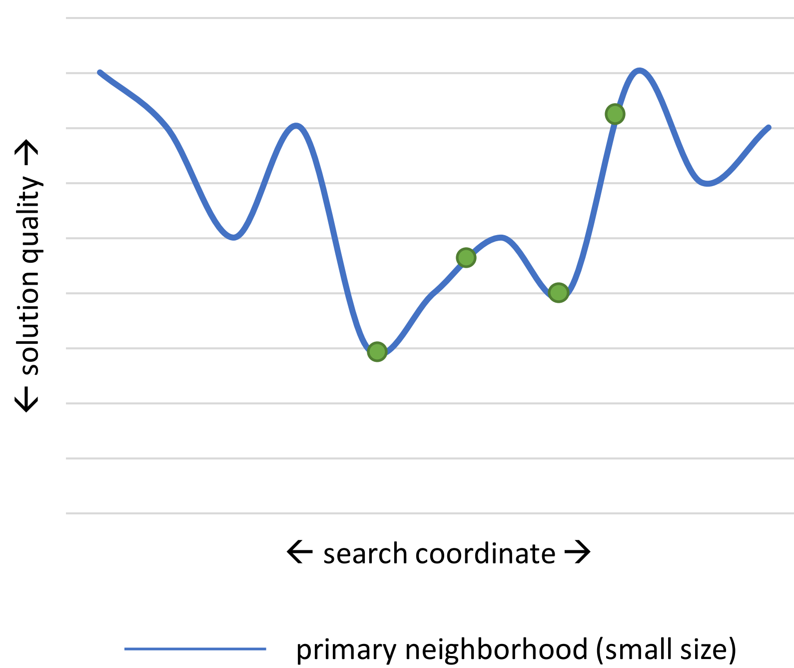
FIGURE 2.24 S-metaheuristic family of algorithms for improving local search and escaping from local optima.

VARIABLE NEIGHBORHOOD SEARCH

- VNS is a search strategy based on dynamically changing neighborhood structures

```
Select a set of neighborhood structures N_k, k = 1, ..., k_max
s ← GenerateInitialSolution()
while termination conditions not met do
  k ← 1
  while k < k_max do               % Inner loop
    s' ← PickAtRandom(N_k(s))            % Shaking phase
    s'' ← LocalSearch(s')
    if (f(s'') < f(s)) then
      s ← s''
      k ← 1
    else
      k ← k + 1
    endif
  endwhile
endwhile
```

Fig. 7. Algorithm: Variable Neighborhood Search (VNS).

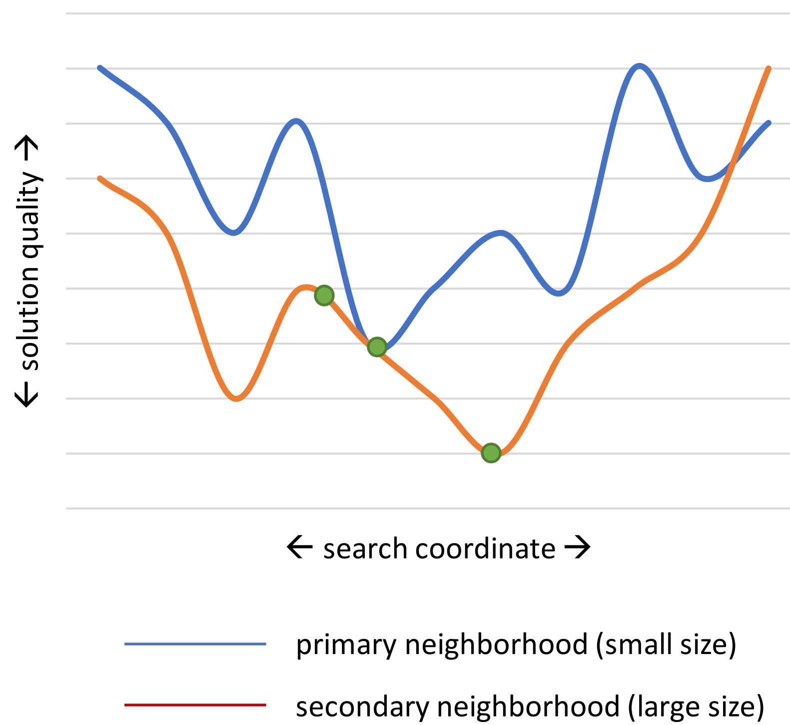


VARIABLE NEIGHBORHOOD SEARCH

- VNS is a search strategy based on dynamically changing neighborhood structures

Select a set of neighborhood structures $\mathcal{N}_k$, $k = 1, \ldots, k_{max}$
$s \leftarrow$ GenerateInitialSolution()
while termination conditions not met **do**
  $k \leftarrow 1$
  while $k < k_{max}$ **do** % Inner loop
    $s' \leftarrow$ PickAtRandom($\mathcal{N}_k(s)$) % Shaking phase
    $s'' \leftarrow$ LocalSearch(s')
    if $(f(s'') < f(s))$ **then**
      $s \leftarrow s''$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
    endif
  endwhile
endwhile

Fig. 7. Algorithm: Variable Neighborhood Search (VNS).

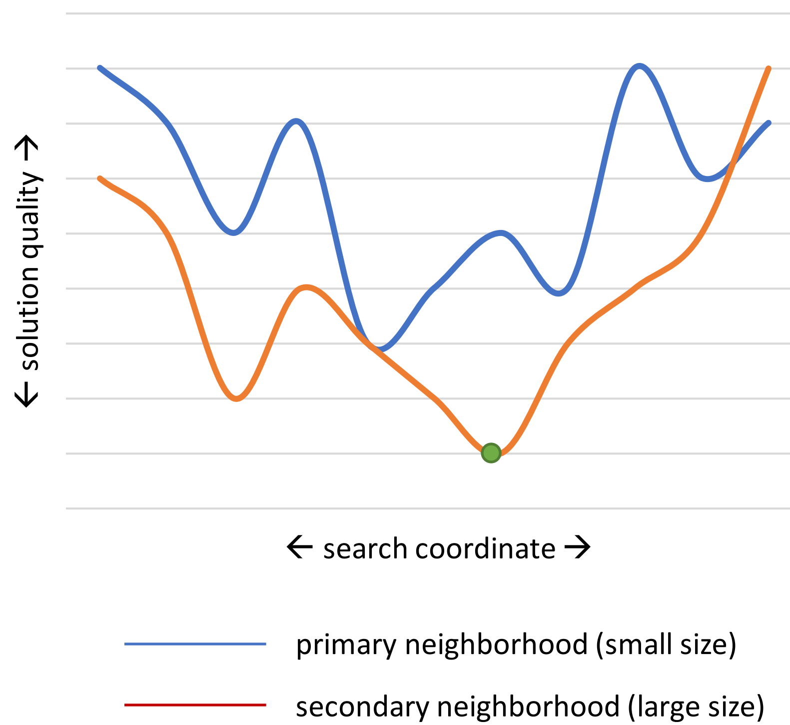


VARIABLE NEIGHBORHOOD SEARCH

- VNS is a search strategy based on dynamically changing neighborhood structures

Select a set of neighborhood structures $\mathcal{N}_k,\ k = 1, \ldots, k_{max}$
$s \leftarrow$ GenerateInitialSolution()
while termination conditions not met **do**
  $k \leftarrow 1$
  while $k < k_{max}$ **do**   % Inner loop
    $s' \leftarrow$ PickAtRandom($\mathcal{N}_k(s)$)   % Shaking phase
    $s'' \leftarrow$ LocalSearch(s')
    if $(f(s'') < f(s))$ **then**
      $s \leftarrow s''$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
    endif
  endwhile
endwhile

Fig. 7. Algorithm: Variable Neighborhood Search (VNS).



What controls the balance between intensification and diversification?

ESCAPING LOCAL OPTIMA

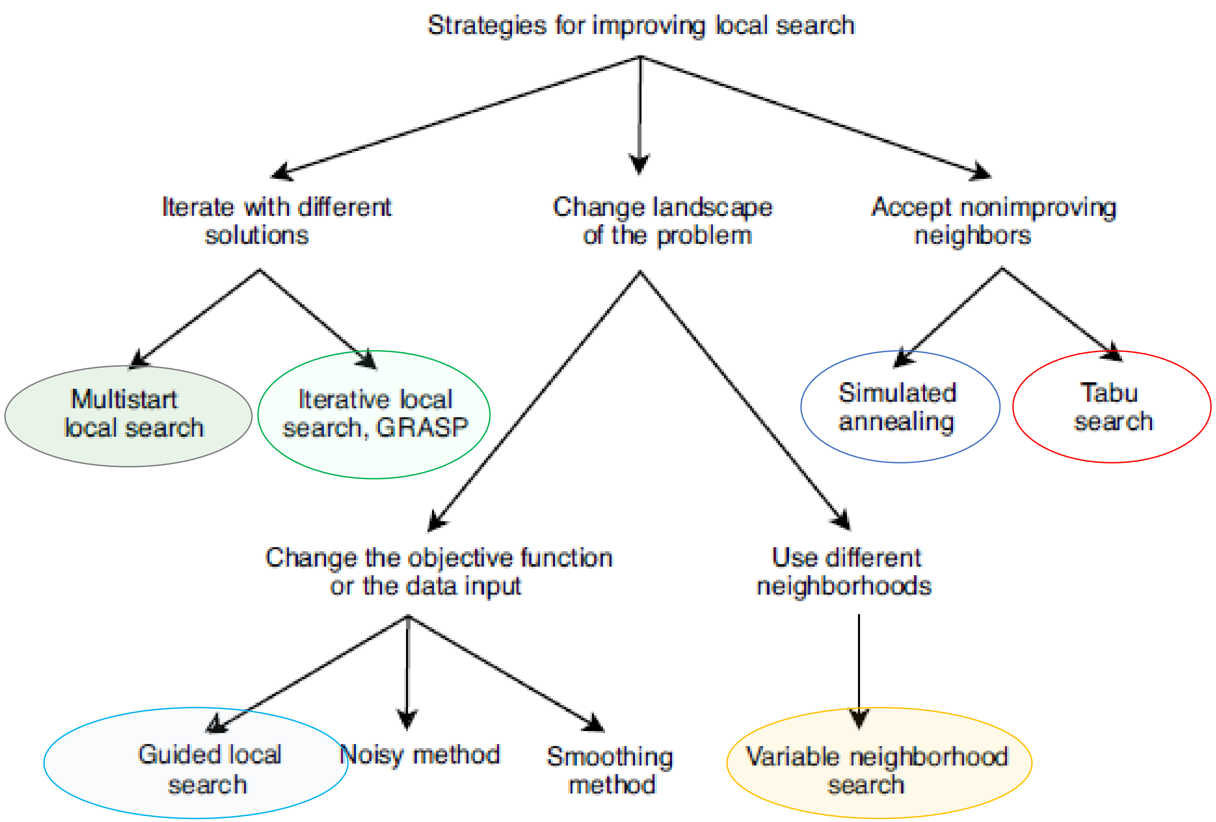


FIGURE 2.24 S-metaheuristic family of algorithms for improving local search and escaping from local optima.

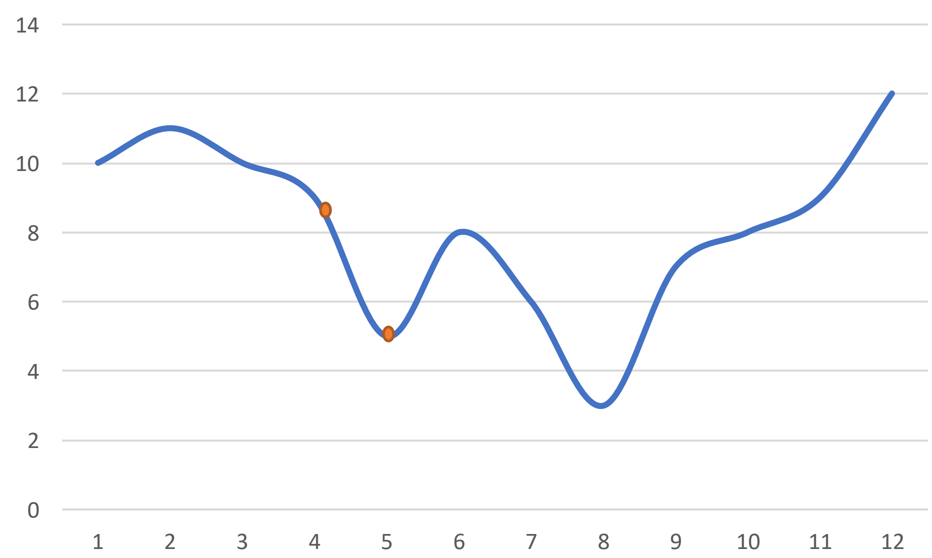
GUIDED LOCAL SEARCH

- Instead of dynamically changing search directions, GLS dynamically changes (augments) objective

- Main idea is to make current solution vicinity increasingly unattractive

GUIDED LOCAL SEARCH

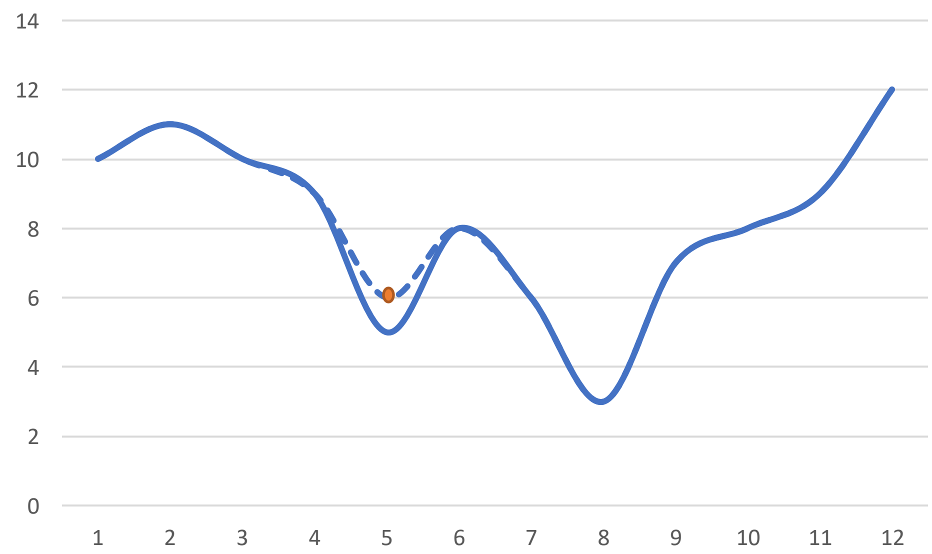
- Instead of dynamically changing search directions, GLS dynamically changes (augments) objective
- Main idea is to make current solution vicinity increasingly unattractive



GUIDED LOCAL SEARCH

- Instead of dynamically changing search directions, GLS dynamically changes (augments) objective

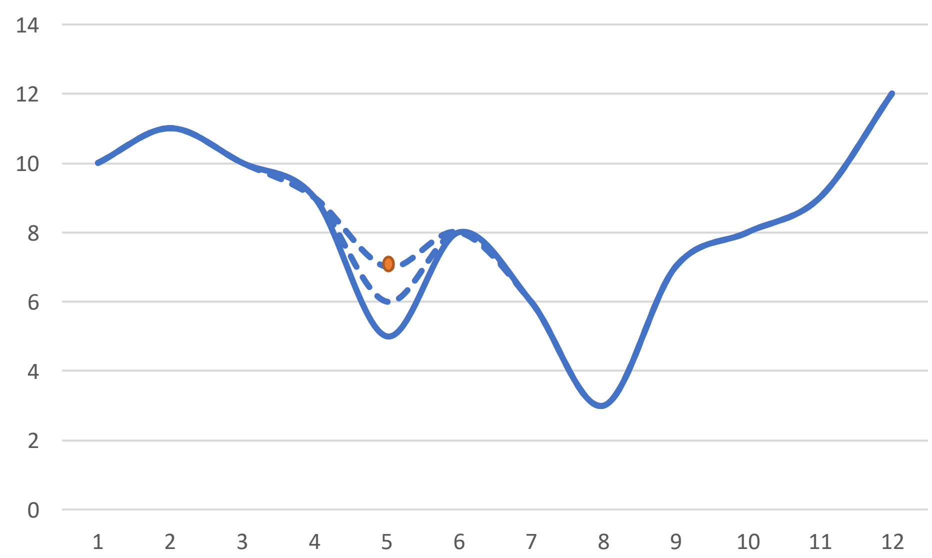
- Main idea is to make current solution vicinity increasingly unattractive



GUIDED LOCAL SEARCH

- Instead of dynamically changing search directions, GLS dynamically changes (augments) objective

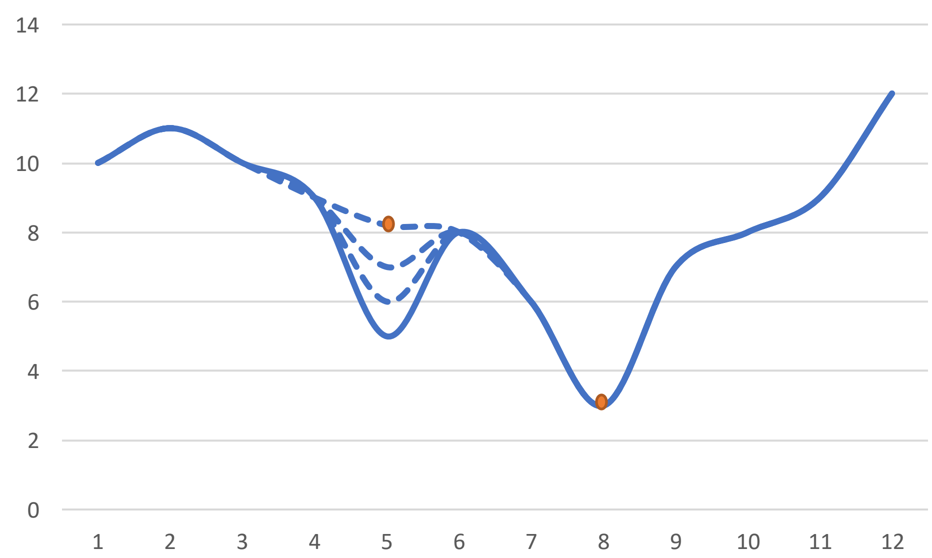
- Main idea is to make current solution vicinity increasingly unattractive



GUIDED LOCAL SEARCH

- Instead of dynamically changing search directions, GLS dynamically changes (augments) objective

- Main idea is to make current solution vicinity increasingly unattractive



GUIDED LOCAL SEARCH

- Instead of dynamically changing search directions, GLS dynamically changes (augments) objective

- Main idea is to make current solution vicinity increasingly unattractive

$$Util(s, i) = I_i(s) \frac{c_i}{1 + p_i}$$

any solution feature
(e.g., a particular
solution component)

```
s ← GenerateInitialSolution()
while termination conditions not met do
  s ← LocalSearch(s, f')
  for all feature i with maximum utility Util(s, i) do
    p_i ← p_i + 1
  endfor
  Update(f', p)          % p is the penalty vector
endwhile
```

Fig. 10. Algorithm: Guided Local Search (GLS).

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i I_i(s)$$

$$I_i(s) = \begin{cases} 1 \text{ , if feature } i \text{ is present in solution } s \\ 0 \text{ , o/w} \end{cases}$$

Trajectory vs Population-based

Trajectory
(S-metaheuristics)

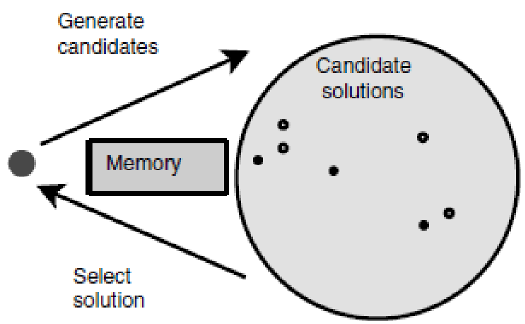


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
/* Generate candidate solutions (partial or complete neighborhood) from s_t */
Generate($C(s_t)$) ;
/* Select a solution from $C(s)$ to replace the current solution s_t */
$s_{t+1} = \text{Select}(C(s_t))$;
$t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.

Population-based
(P-metaheuristics)

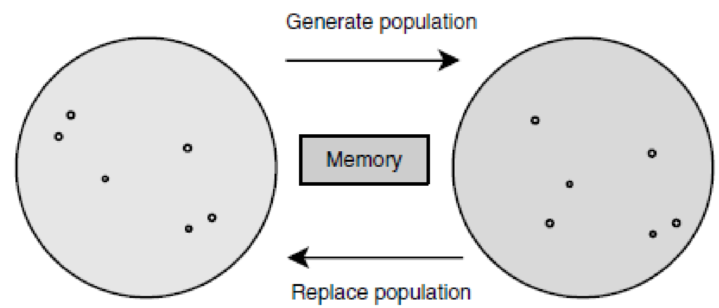


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
Generate(P_t'); /* Generation a new population */
P_{t+1} = Select-Population($P_t \cup P_t'$); /* Select new population */
$t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

EVOLUTIONARY COMPUTATION

- Populations evolve in “generations”
- New individuals (offspring) are created by combining features of current individuals (parents);
  - typically two parents combine to give offspring
- Individuals evolve using variation operators (e.g., “mutation”, “recombination”) acting directly on their solution representations
- The next population consists of a mix of offspring and parents (“survivor selection” strategy)

EVOLUTIONARY COMPUTATION

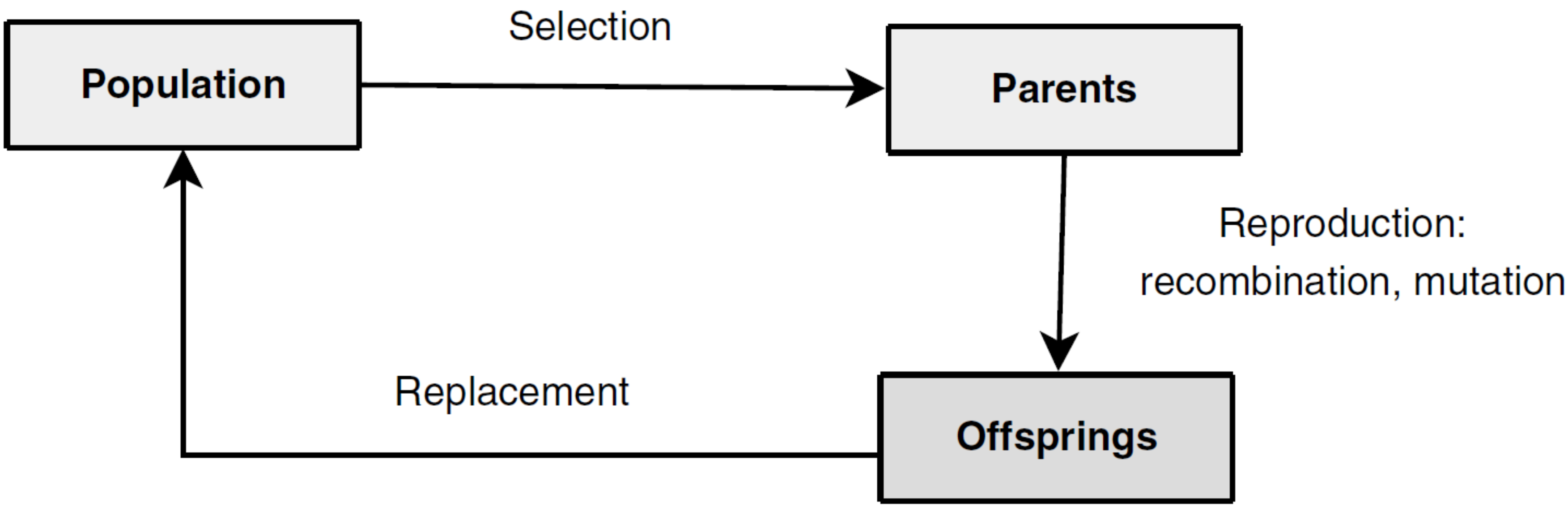
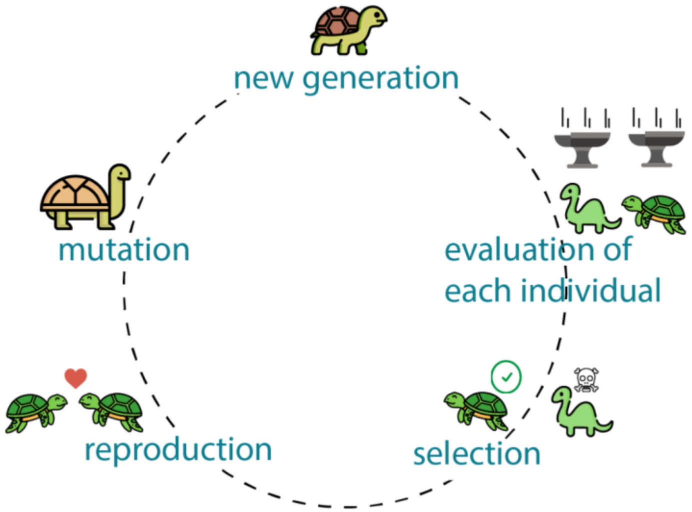


FIGURE 3.7 A generation in evolutionary algorithms.

EVOLUTIONARY COMPUTATION

$P \leftarrow \text{GenerateInitialPopulation}()$
$\text{Evaluate}(P)$
while termination conditions not met **do**
  $P' \leftarrow \text{Recombine}(P)$
  $P'' \leftarrow \text{Mutate}(P')$
  $\text{Evaluate}(P'')$
  $P \leftarrow \text{Select}(P'' \cup P)$
endwhile

Fig. 13. Algorithm: Evolutionary Computation (EC).

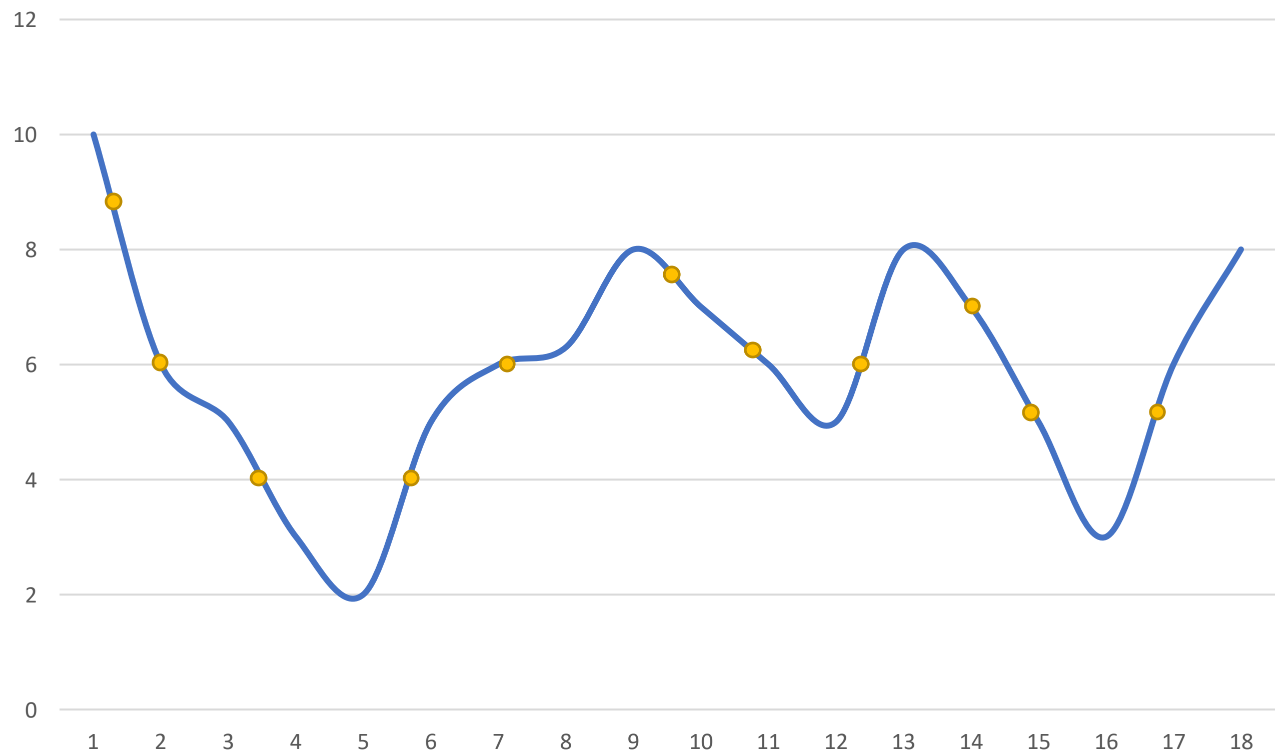


EVOLUTIONARY COMPUTATION

- The population size is (almost always) fixed
- Selection may involve multiple copies of a given parent individual
- The best individual is (almost always) carried over to the next generation
- Randomness plays a significant role in generating offspring (unlike other non-EC methods)
- Solutions are often represented in compact and “easily mutable” form, e.g., bit-strings or integer permutations

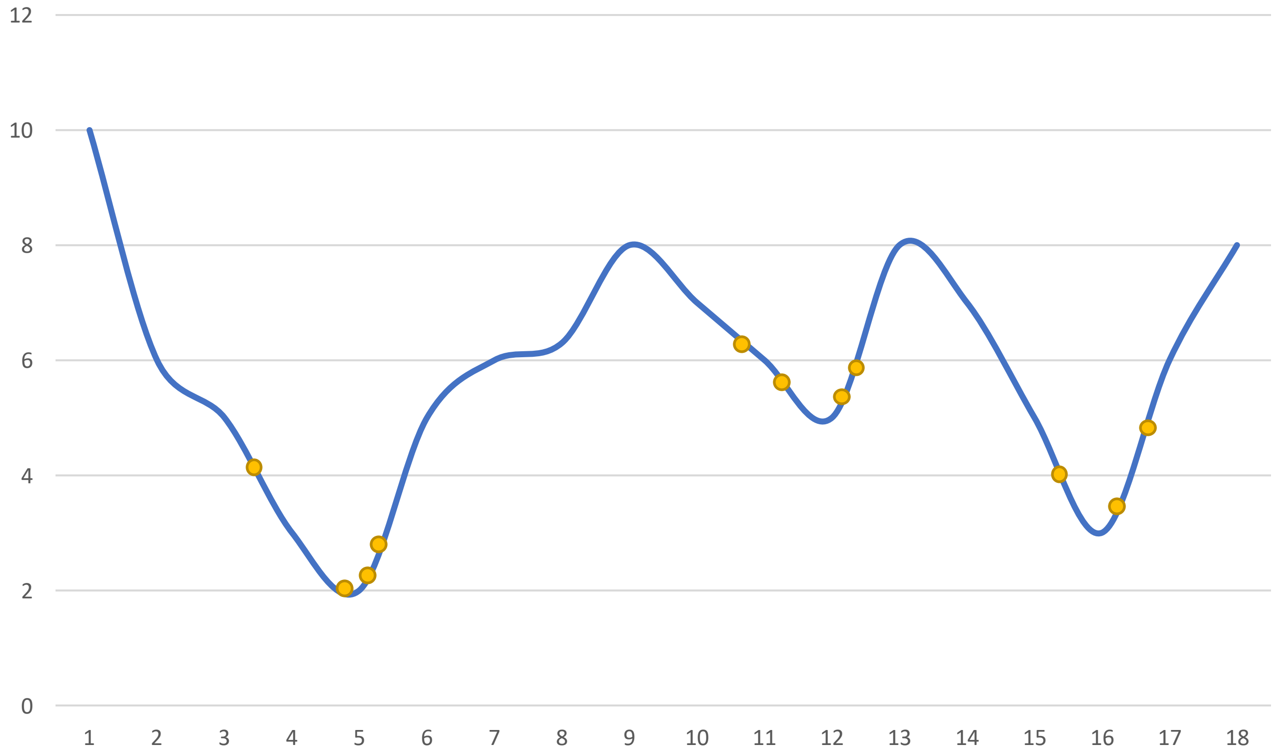
EVOLUTIONARY COMPUTATION

- 0^{th} population



EVOLUTIONARY COMPUTATION

- N^{th} population



GENETIC ALGORITHM

- Basic Evolutionary Computation Algorithm
  - Representation of individuals in binary code

    (e.g., “01101” = 13, “11010” = 26)
  - Use of the “crossover” recombination operator
  - Mutation via “bit-flipping”
  - Offspring always survive

Initial Population for TSP

(5,3,4,6,2)	(2,4,6,3,5)	(4,3,6,5,2)
(2,3,4,6,5)	(4,3,6,2,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	(3,4,2,6,5)	(3,6,5,1,4)

Select Parents

(5,3,4,6,2)	(2,4,6,3,5)	(4,3,6,5,2)
(2,3,4,6,5)	(4,3,6,2,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

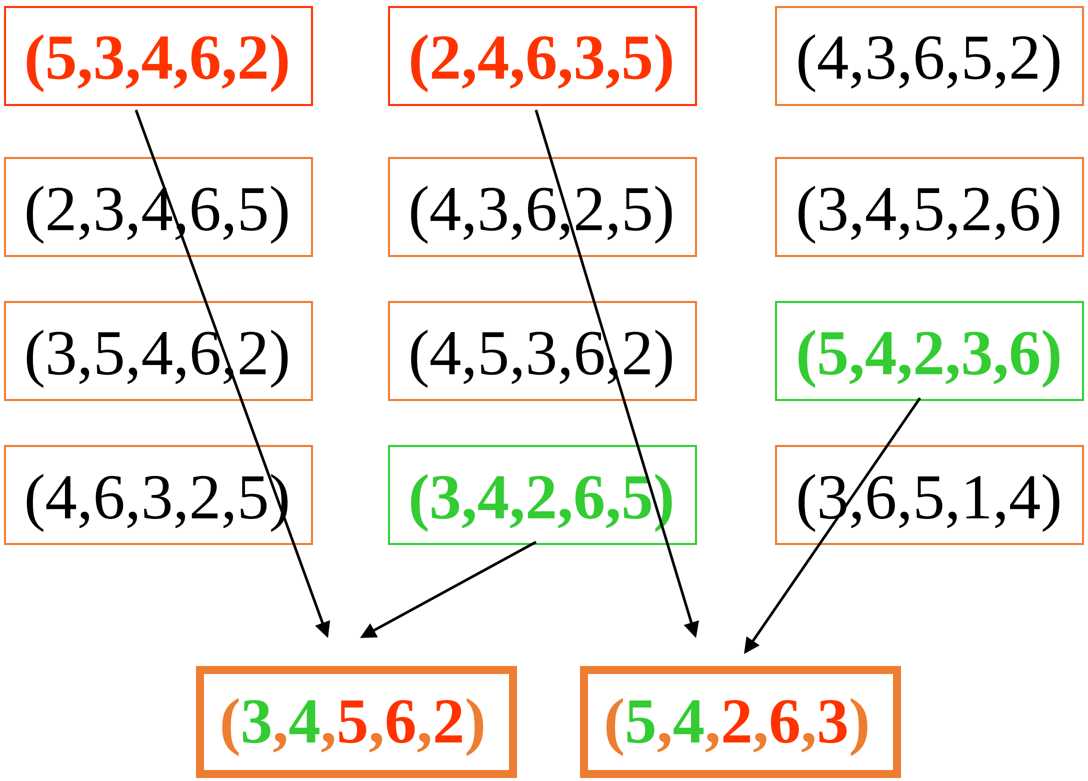
Try to pick the better ones.

Create Off-Spring – 1 point

(5,3,4,6,2)	(2,4,6,3,5)	(4,3,6,5,2)
(2,3,4,6,5)	(4,3,6,2,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2)

Create More Offspring



Mutate

(5,3,4,6,2)	**(2,4,6,3,5)**	(4,3,6,5,2)
(2,3,4,6,5)	**(4,3,6,2,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2) **(5,4,2,6,3)**

Mutate

(5,3,4,6,2)	**(2,4,6,3,5)**	(4,3,6,5,2)
(2,3,4,6,5)	**(2,3,6,4,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2) **(5,4,2,6,3)**

Eliminate

(5,3,4,6,2)	**(2,4,6,3,5)**	~~(4,3,6,5,2)~~
~~(2,3,4,6,5)~~	**(2,3,6,4,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

(3,4,5,6,2) **(5,4,2,6,3)**

Tend to kill off the worst ones.

Integrate

(5,3,4,6,2)	**(2,4,6,3,5)**	**(5,4,2,6,3)**
(3,4,5,6,2)	**(2,3,6,4,5)**	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	**(5,4,2,3,6)**
(4,6,3,2,5)	**(3,4,2,6,5)**	(3,6,5,1,4)

Restart

(5,3,4,6,2)	(2,4,6,3,5)	(5,4,2,6,3)
(3,4,5,6,2)	(2,3,6,4,5)	(3,4,5,2,6)
(3,5,4,6,2)	(4,5,3,6,2)	(5,4,2,3,6)
(4,6,3,2,5)	(3,4,2,6,5)	(3,6,5,1,4)

ANT COLONY OPTIMIZATION

How Ants Find Food

Social insects, following simple, individual rules, accomplish complex colony activities through: flexibility, robustness and self-organization

Pheromone Trail Following

Ants and termites follow pheromone trails

ANT COLONY OPTIMIZATION

- Each possible solution component is associated with a level of pheromone, which goes up and down depending on the probability that the component is part of the final solution

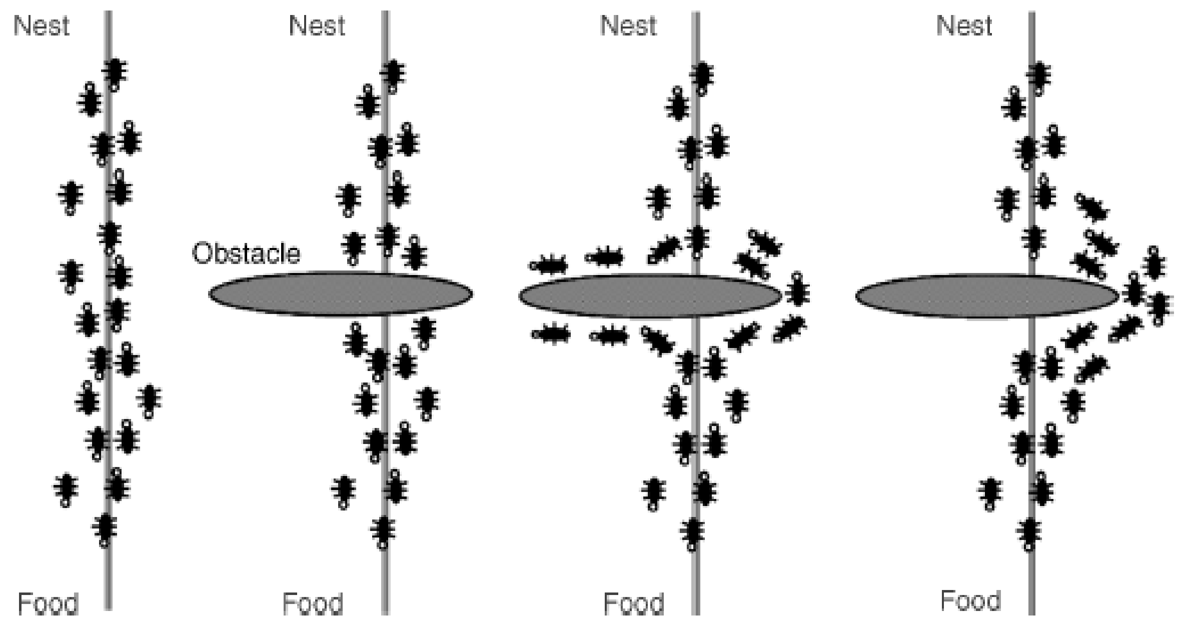
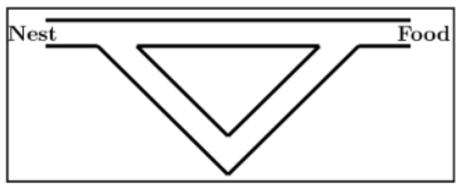


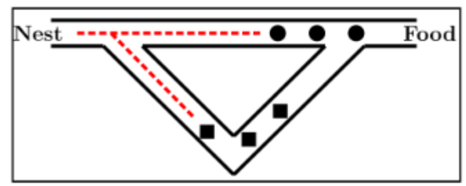
FIGURE 3.32 Inspiration from an ant colony searching an optimal path between the food and the nest.

ANT COLONY OPTIMIZATION

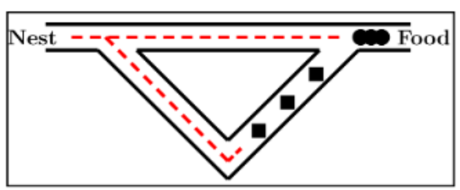
- Blum C. (2005), "Ant colony optimization: Introduction and recent trends." *Physics of Life Reviews,* 2:353-363.



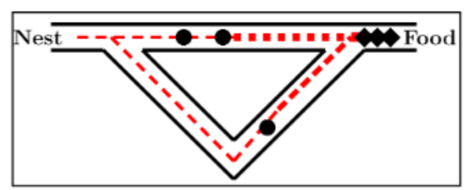
(a) All ants are in the nest. There is no pheromone in the environment.



(b) The foraging starts. In probability, 50% of the ants take the short path (symbolized by circles), and 50% take the long path to the food source (symbolized by rhombs).



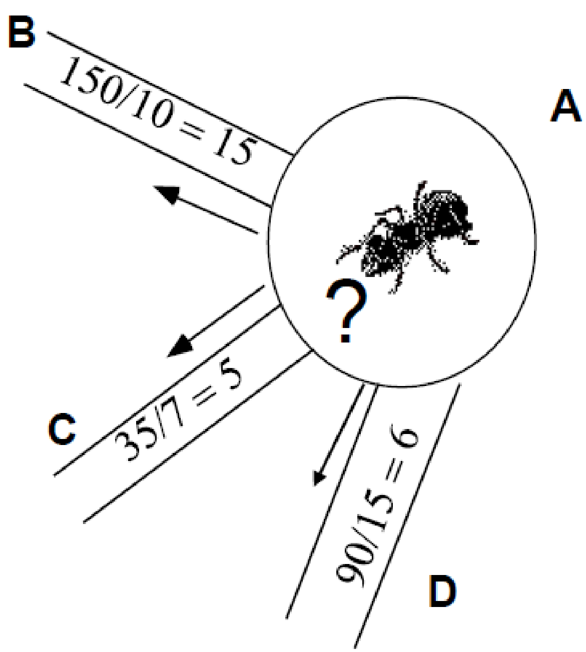
(c) The ants that have taken the short path have arrived earlier at the food source. Therefore, when returning, the probability to take again the short path is higher.

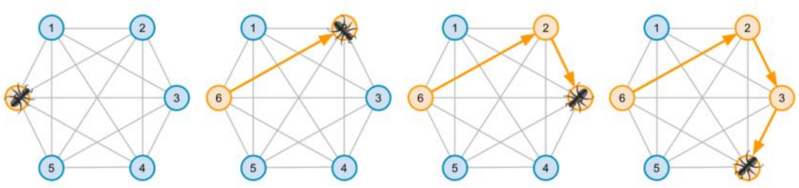


(d) The pheromone trail on the short path receives, in probability, a stronger reinforcement, and the probability to take this path grows. Finally, due to the evaporation of the pheromone on the long path, the whole colony will, in probability, use the short path.

ANT COLONY OPTIMIZATION

- Ants are almost always artificial (do not use real ants!)
- Each artificial ant represents a solution that is being incrementally constructed by adding opportunely defined solution components (i.e., according to a probabilistic transition rule)





ANT COLONY OPTIMIZATION

"Rules of the game"

The pheromone level of each solution component i is initialized to a uniform value $\tau_i = \tau^0 \ \forall \ i$; then, at each iteration:

- A population of N ants gets associated with N empty solutions
- For each ant, we incrementally build a solution; the probability of adding the next solution component i is

$$p_i = \begin{cases} \dfrac{\tau_i^\alpha / c_i^\beta}{\sum_{j \in J} \tau_j^\alpha / c_j^\beta} & , if \ i \in J \\ 0 & , o/w \end{cases}$$

α, $\beta \geq 1$ are scaling parameters

J is the set of solution components that "fit" at this point of the construction

- Once all solutions have been constructed, the pheromone values of each component i gets updated according to the formula

$$\tau_i \leftarrow (1-\rho)\, \tau_i + \frac{1}{N} \sum_{\substack{a=1: \\ i \in s_a}}^{N} F(s_a)$$

$F(s_a)$ is the solution quality function (high values for overall good solutions)

$\rho \in [0,1]$ is the evaporation rate

- Repeat

TRAJECTORY VS POPULATION BASED METHODS

Trajectory
(S-metaheuristics)

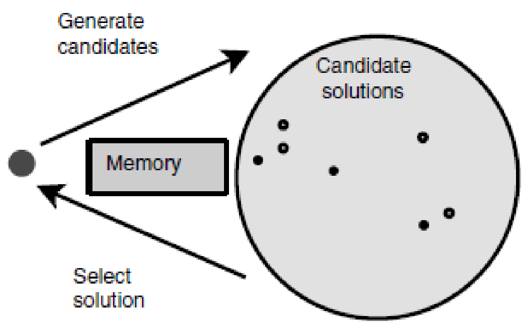


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  $s_{t+1} = \text{Select}(C(s_t))$;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.

Population-based
(P-metaheuristics)

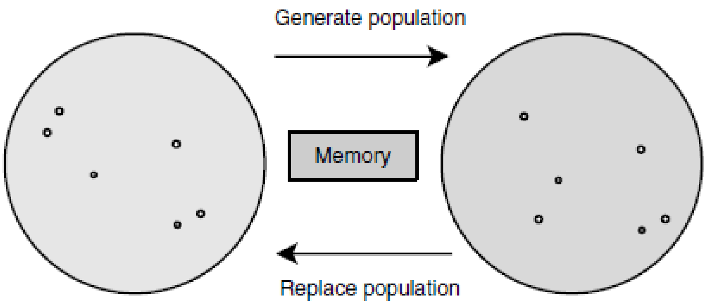


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P_t'); /* Generation a new population */
  $P_{t+1} = \text{Select-Population}(P_t \cup P_t')$; /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

TRAJECTORY METHODS

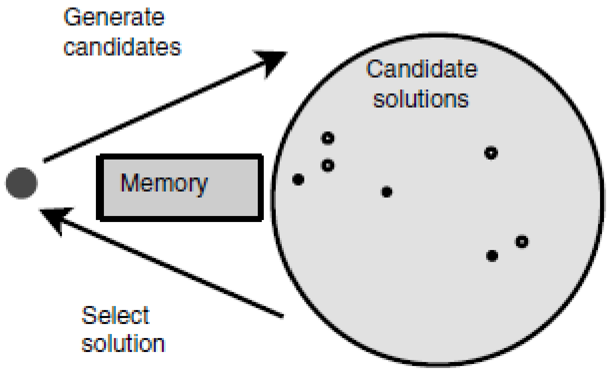
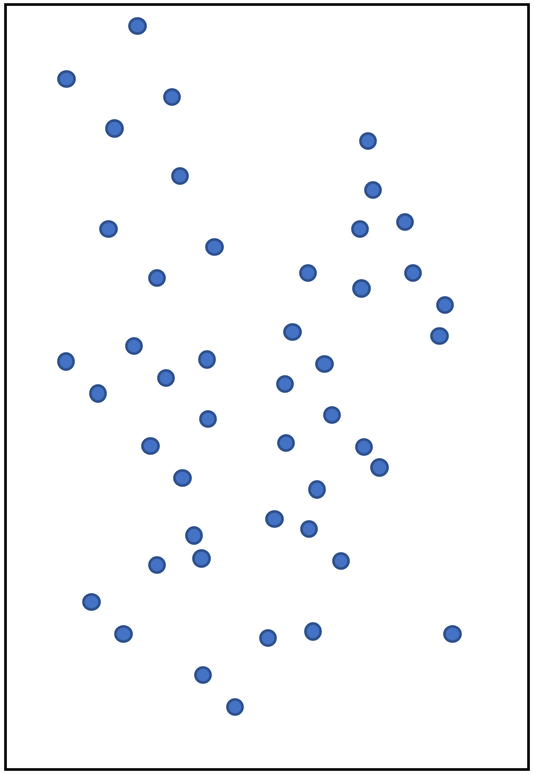


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  $s_{t+1} =$ Select($C(s_t)$) ;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.



TRAJECTORY METHODS

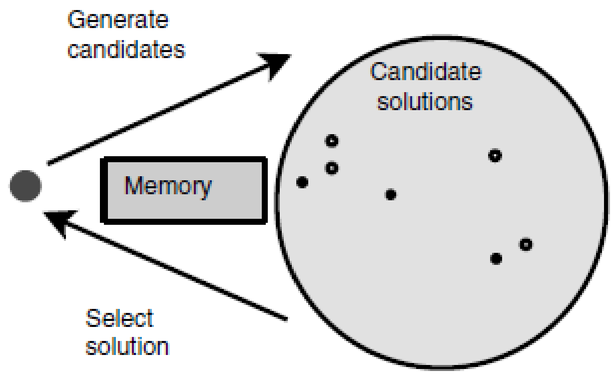
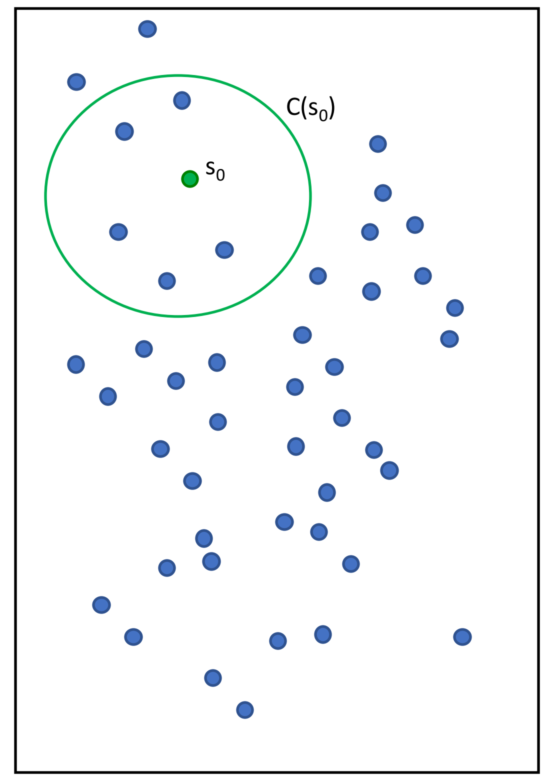


FIGURE 2.1 Main principles of single-based metaheuristics.



Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  s_{t+1} = Select($C(s_t)$) ;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.

TRAJECTORY METHODS

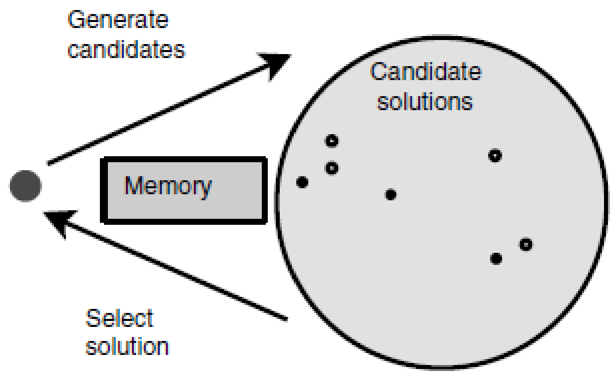
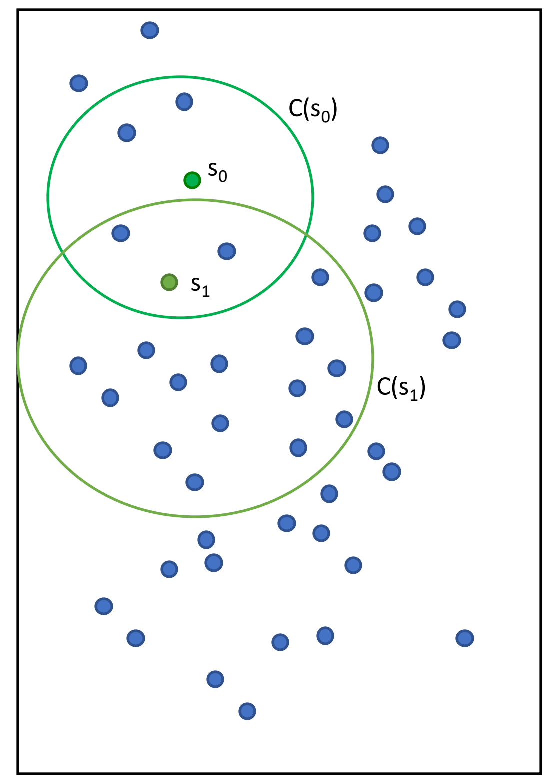


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
/* Generate candidate solutions (partial or complete neighborhood) from s_t */
Generate($C(s_t)$) ;
/* Select a solution from $C(s)$ to replace the current solution s_t */
$s_{t+1} = \text{Select}(C(s_t))$;
$t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.



TRAJECTORY METHODS

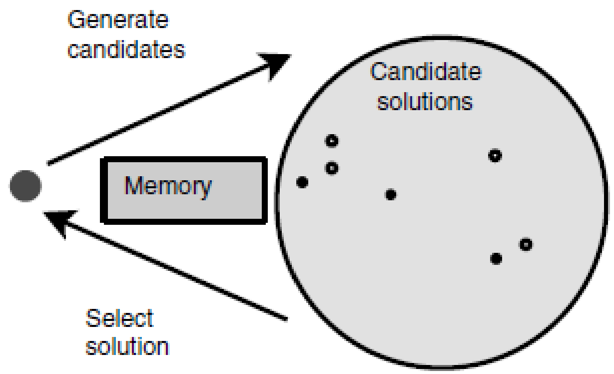
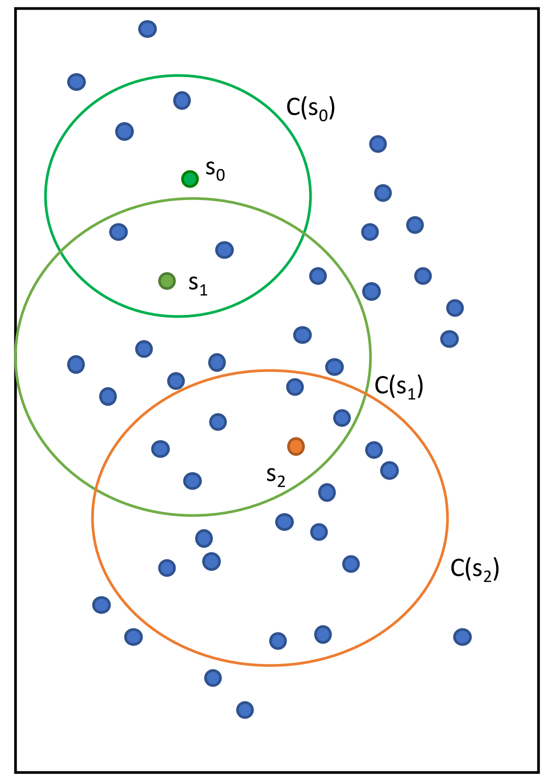


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  $s_{t+1} = \text{Select}(C(s_t))$;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.



Stop because of no improvement in region $C(s_2)$

TRAJECTORY METHODS

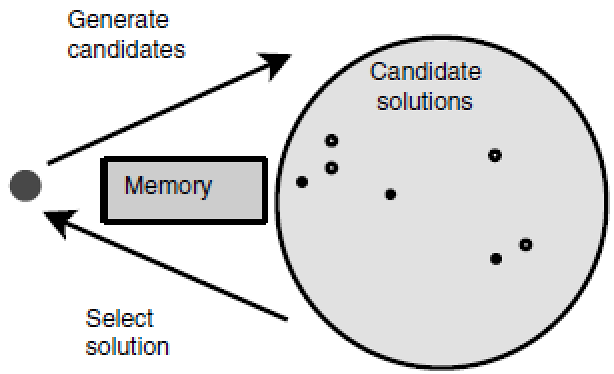
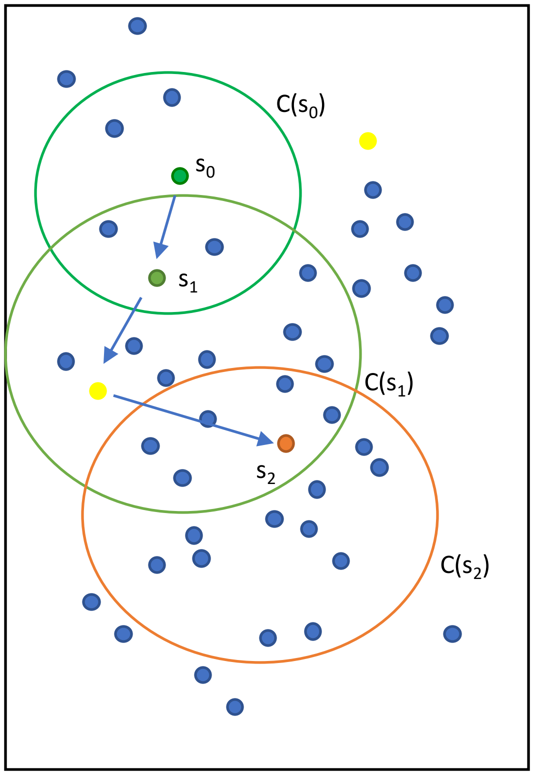


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
/* Generate candidate solutions (partial or complete neighborhood) from s_t */
Generate($C(s_t)$) ;
/* Select a solution from $C(s)$ to replace the current solution s_t */
$s_{t+1} = \text{Select}(C(s_t))$;
$t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.



Turns out two global optima in this problem, but none was identified

- One was missed during search of region $C(s_1)$
- One was far away from searched space

POPULATION-BASED METHODS

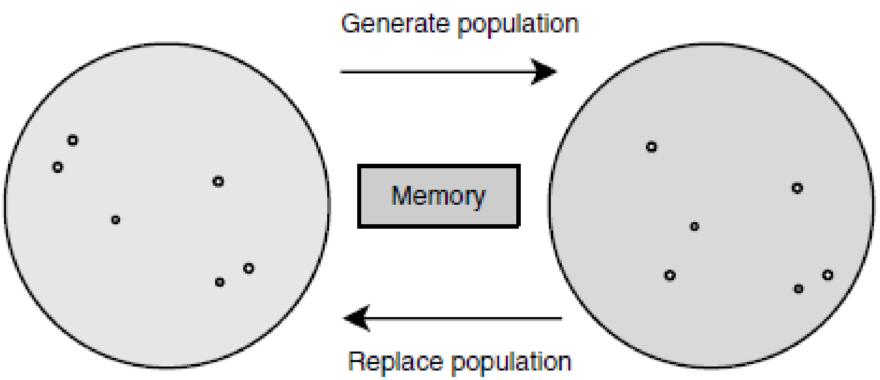
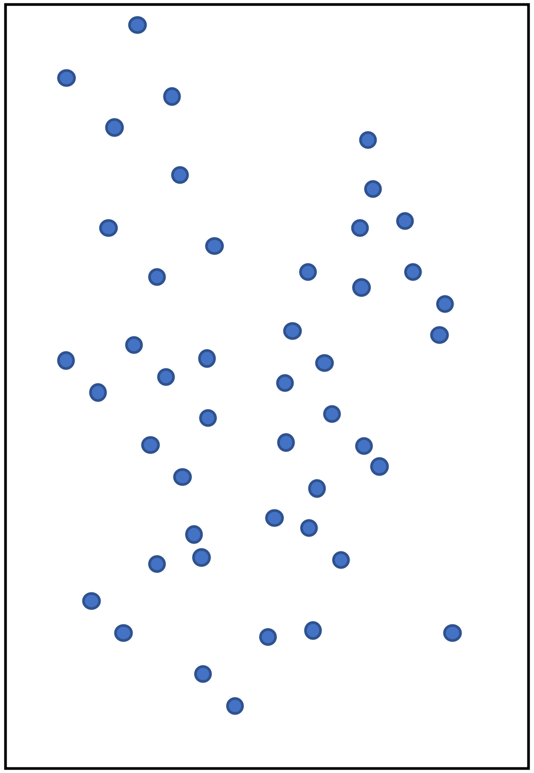


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.



POPULATION-BASED METHODS

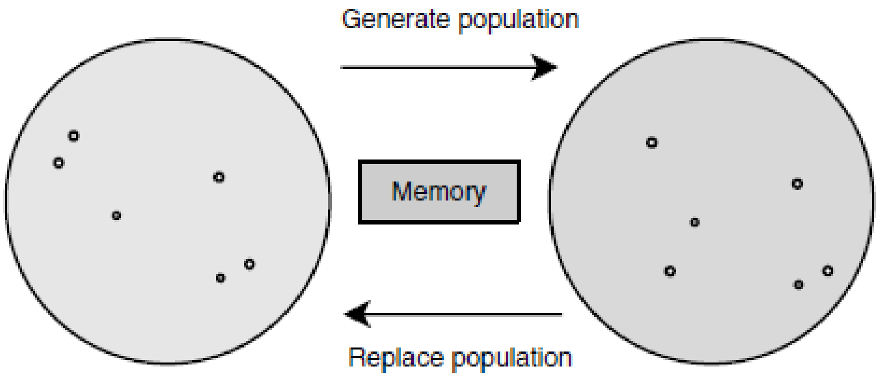
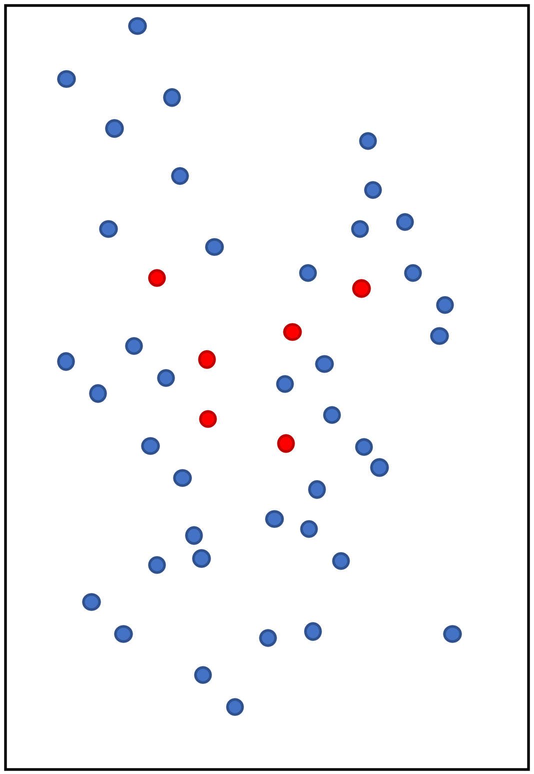


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

0^{th} population



POPULATION-BASED METHODS

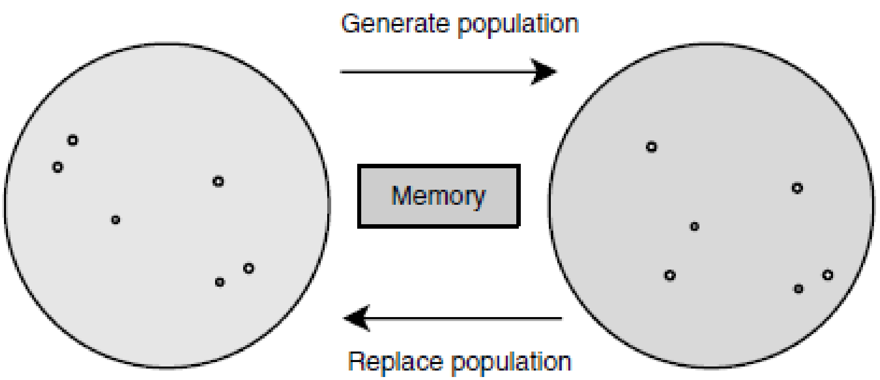
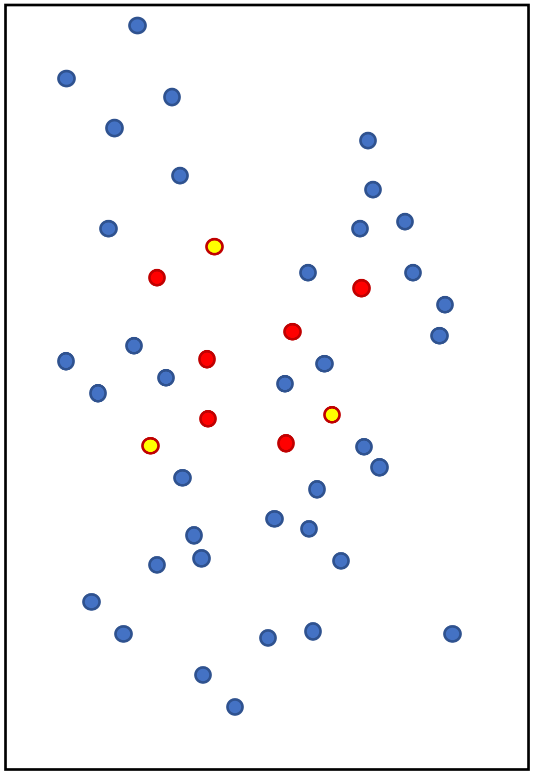


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

Get some new points



POPULATION-BASED METHODS

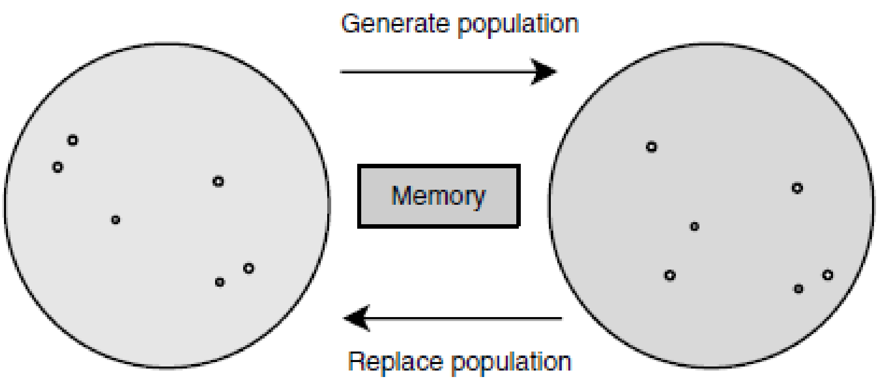
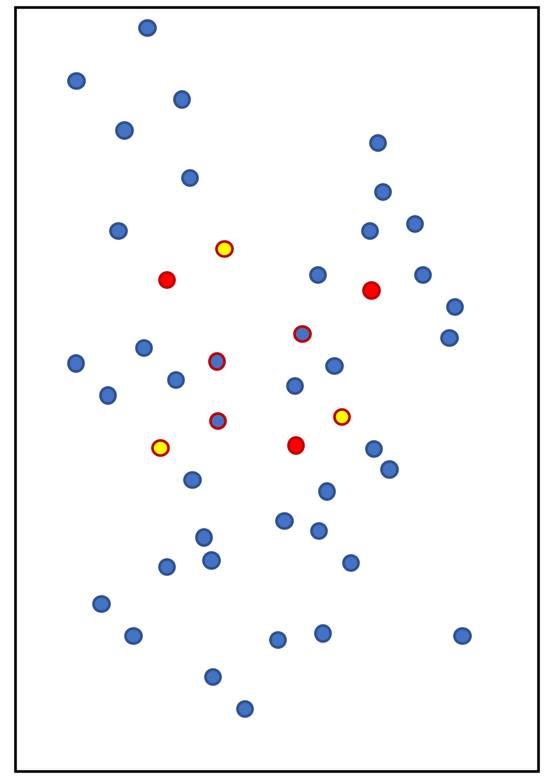


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P_t'); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P_t'$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

Drop some old points



POPULATION-BASED METHODS

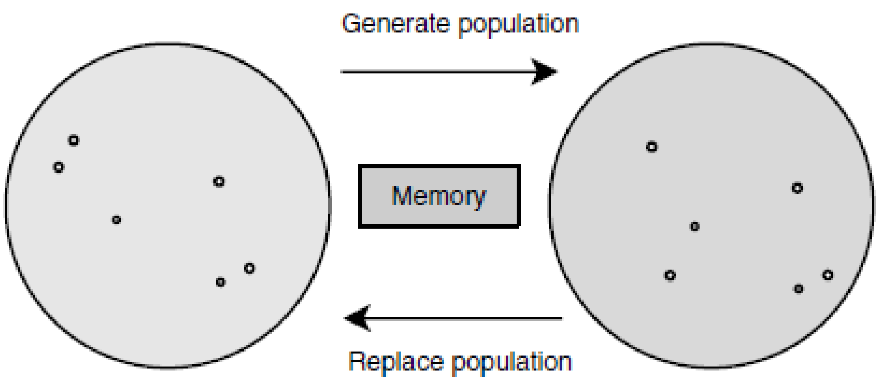
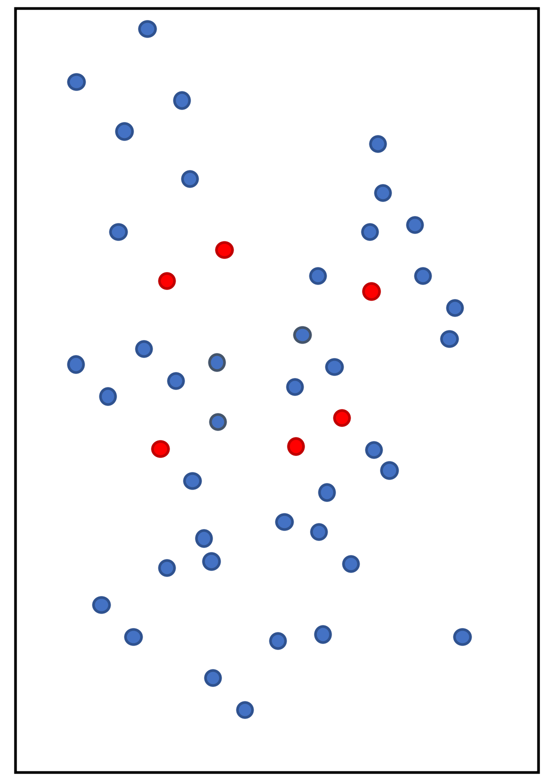


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

1st population



POPULATION-BASED METHODS

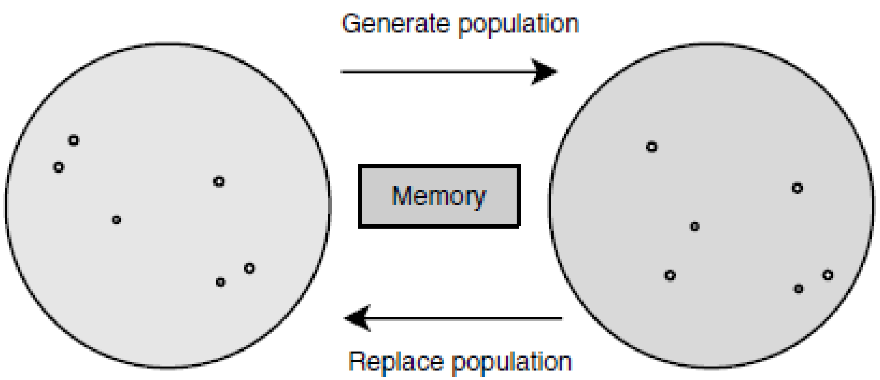
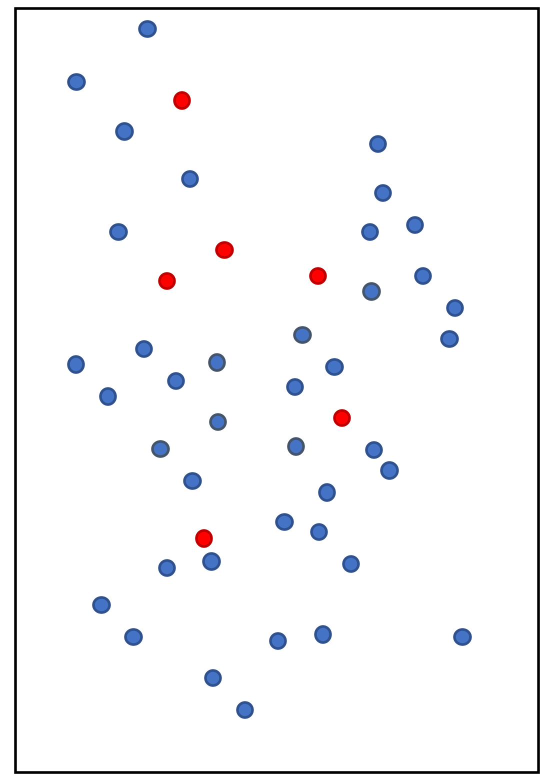


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

2nd population



POPULATION-BASED METHODS

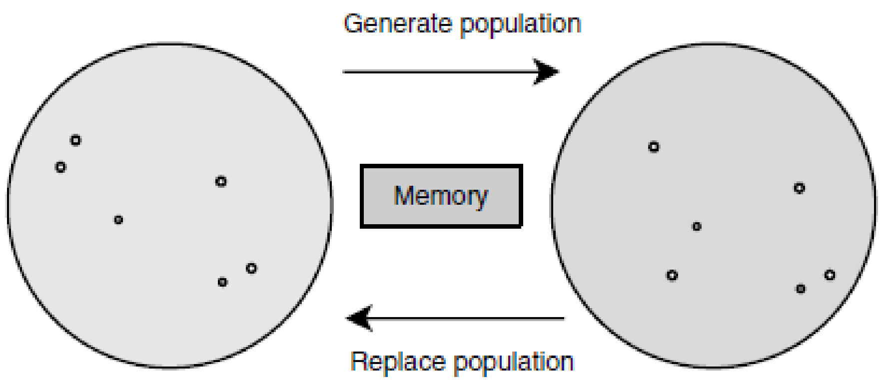
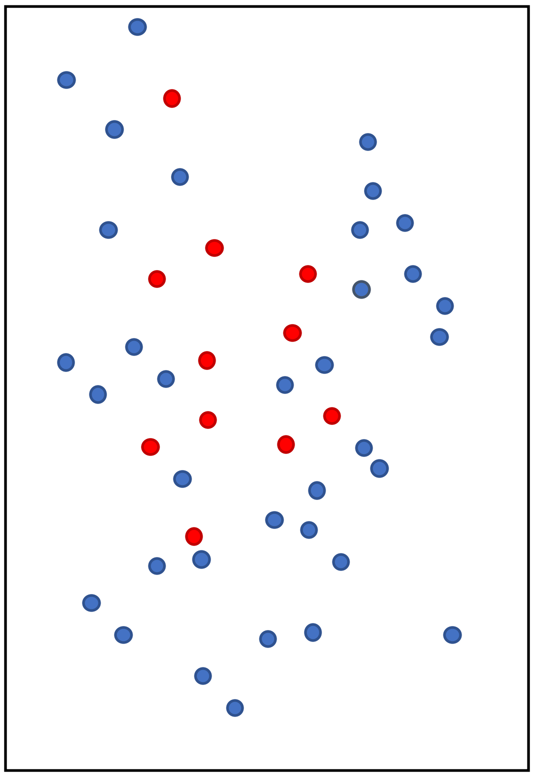


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P_t'); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P_t'$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

All points sampled



Again, optimum may or may not have been sampled

- Typically, the incumbent always remains in the population, so need only focus on last generation

TRAJECTORY BASED METHODS

Trajectory
(S-metaheuristics)

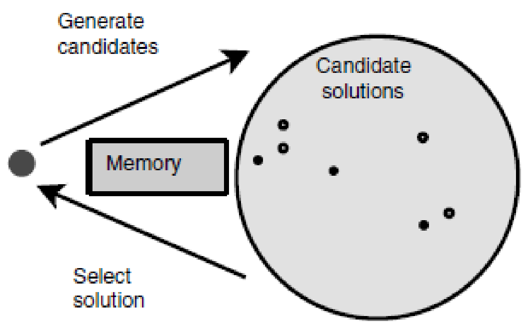
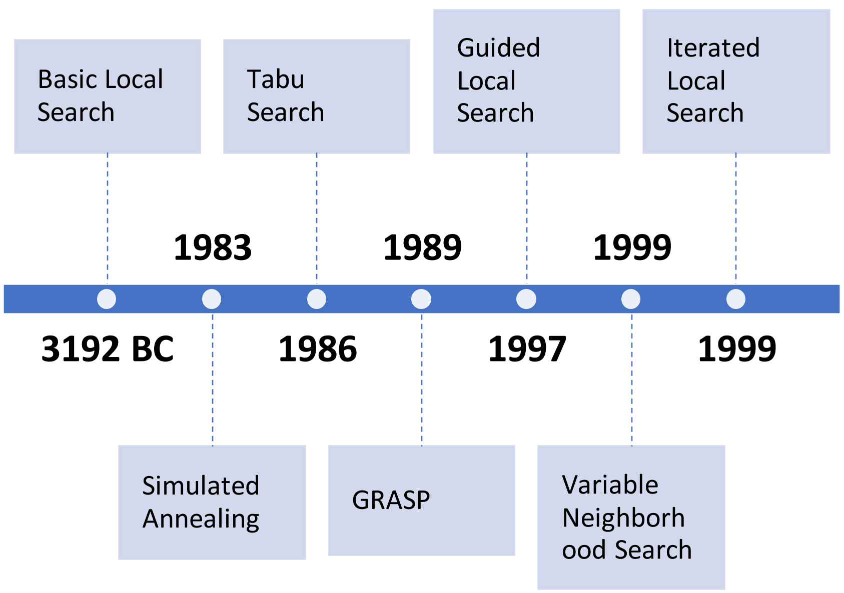


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  $s_{t+1} = \text{Select}(C(s_t))$;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.



POPULATION BASED

Population-based
(P-metaheuristics)

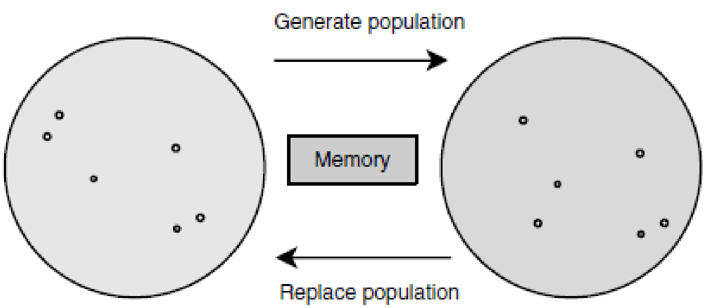


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

Evolutionary Computation

- Evolutionary Programming (1962)
- Evolutionary Strategies (1973)
- Genetic Algorithms (1975)
- Estimation of Distribution Algorithm (1996)

Swarm Intelligence

- Ant Colony Optimization (1992)
- Particle Swarm Optimization (1995)
- Honey-Bees Mating (2005)

Differential Evolution (1995)

Adaptive Memory Programming (1997)

Scatter Search/Path Relinking (1999)

INTENSIFY & DIVERSIFY

INTENSIFY & DIVERSIFY

Intensify

- to become stronger or more extreme; to become more intense

Diversify

- to change (something) so that it has more different kinds of people or things

INTENSIFY & DIVERSIFY

Every metaheuristic aspires to do both

Many have parameters to control this balance

Often, these parameters can be changed dynamically during search

Initially, focus is on diversification; later, on intensification (and cycles back)