VRPTW + CF

Rania Fathallah - Mariam Amdouni

Plan de la présentation

01 Problématique de VPRTW

02 Fondement théroique de CF

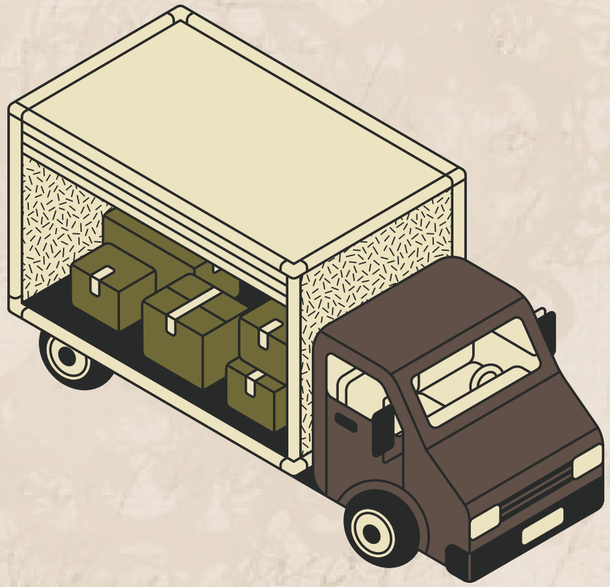
03 Méthodologie de résolution du VRPTW avec CF

04 Analyse

05 Performance sur quelque instance de Solomon’s Benchmark

01– Problématique

Problème VRPTW
VEHICLE RROUTING PROBLEM WITH TIME WINDOW



Définition du problème

Le problème de routage de véhicules avec fenêtres temporelles (VRPTW) peut être défini comme le choix des itinéraires pour un nombre de véhicules afin de desservir un groupe de clients dans des plages horaires spécifiques.

Les Principales contraintes

Contraintes

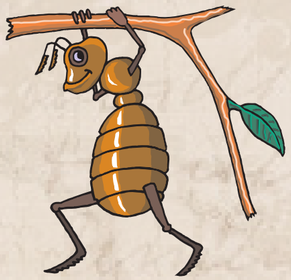
- la capacité des véhicules
- les fenêtres temporelles des clients

02- CF

FONDEMENT THÉORIQUE DE CF
ANT COLONY
COLONIE DE FOURMIS

Ant Colony :

L'Ant Colony Optimization est choisi pour résoudre le VRPTW en raison de sa capacité à trouver des solutions efficaces dans des problèmes d'optimisation combinatoire avec des contraintes temporelles.

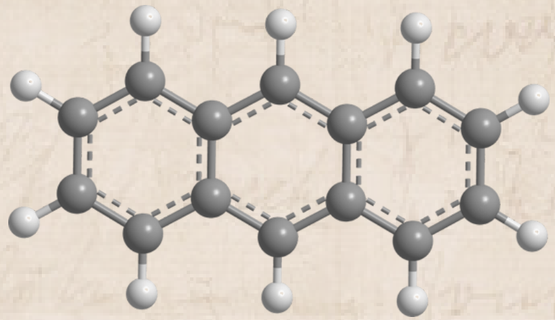


Ant Colony :Relation avec l’informatique

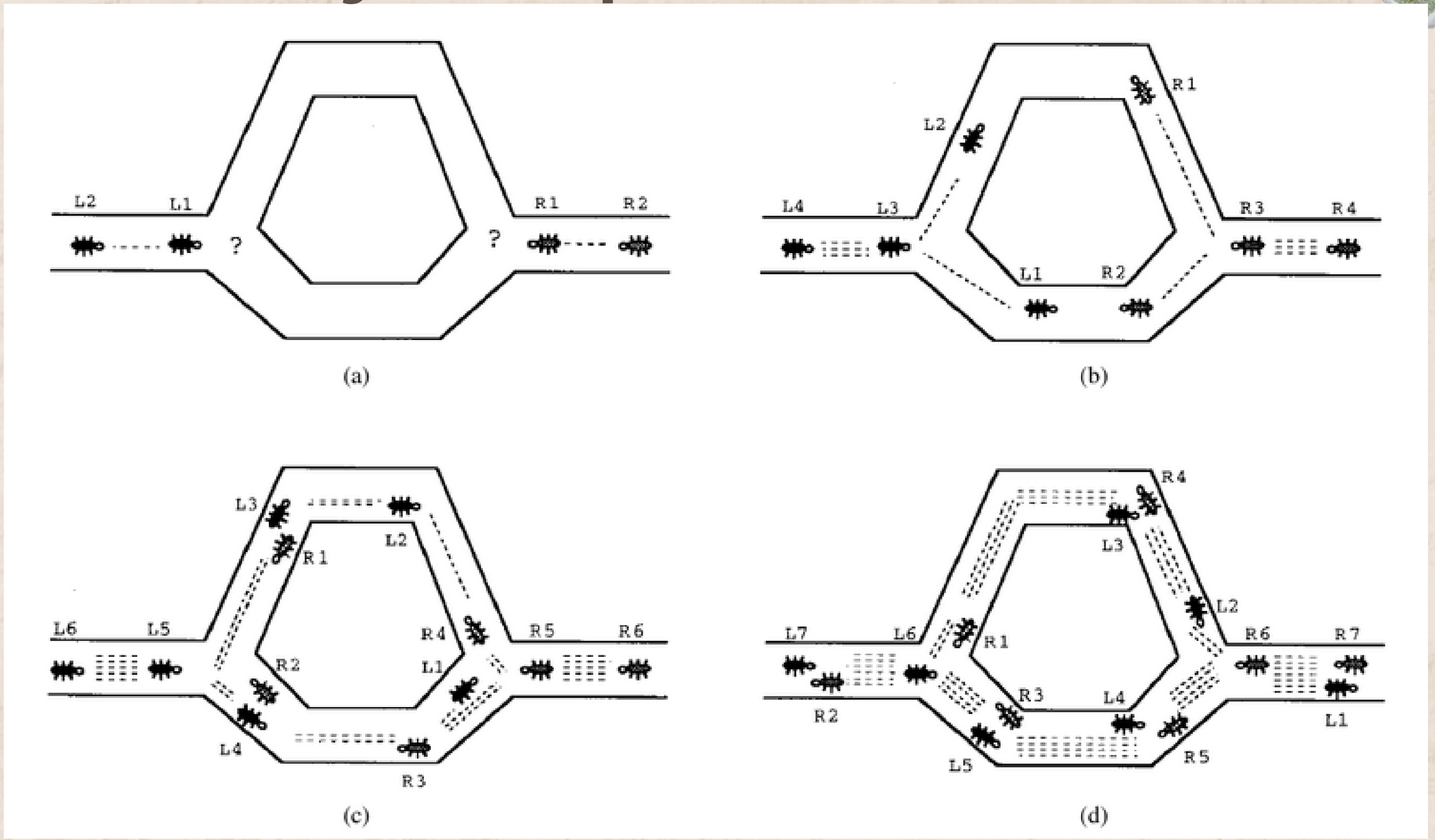
Les fourmis résolvent des problèmes complexes tels que celui de la recherche du plus court chemin.

Ant Colony :Comportement de la fourmi

Les fourmis utilisent des **phéromones**



Ant Colony :Comportement de la fourmi

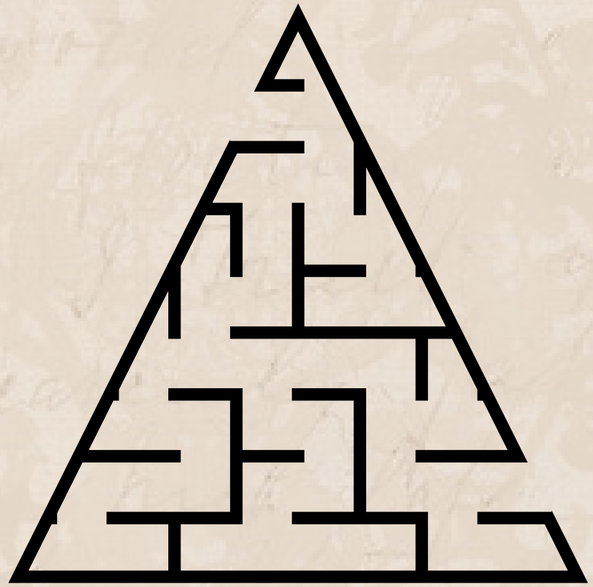


Ant Colony :Comportement de la fourmi

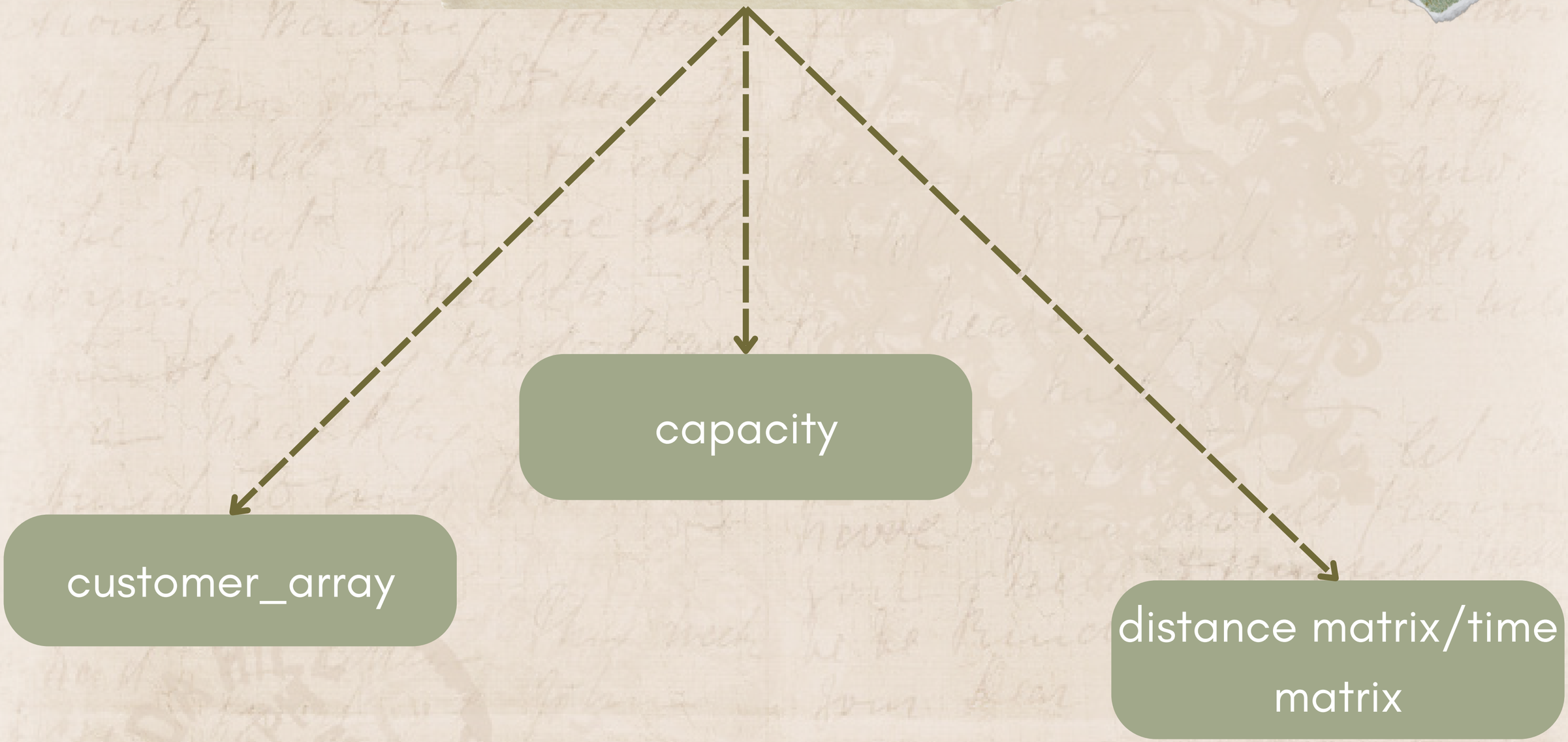
Le comportement des fourmis leur permet de trouver le chemin le plus court vers la nourriture en utilisant les pistes de phéromones, facilitant ainsi le choix optimal parmi plusieurs chemins marqués et constituant la base de méthodes ultérieures de navigation

03- Méthodologie

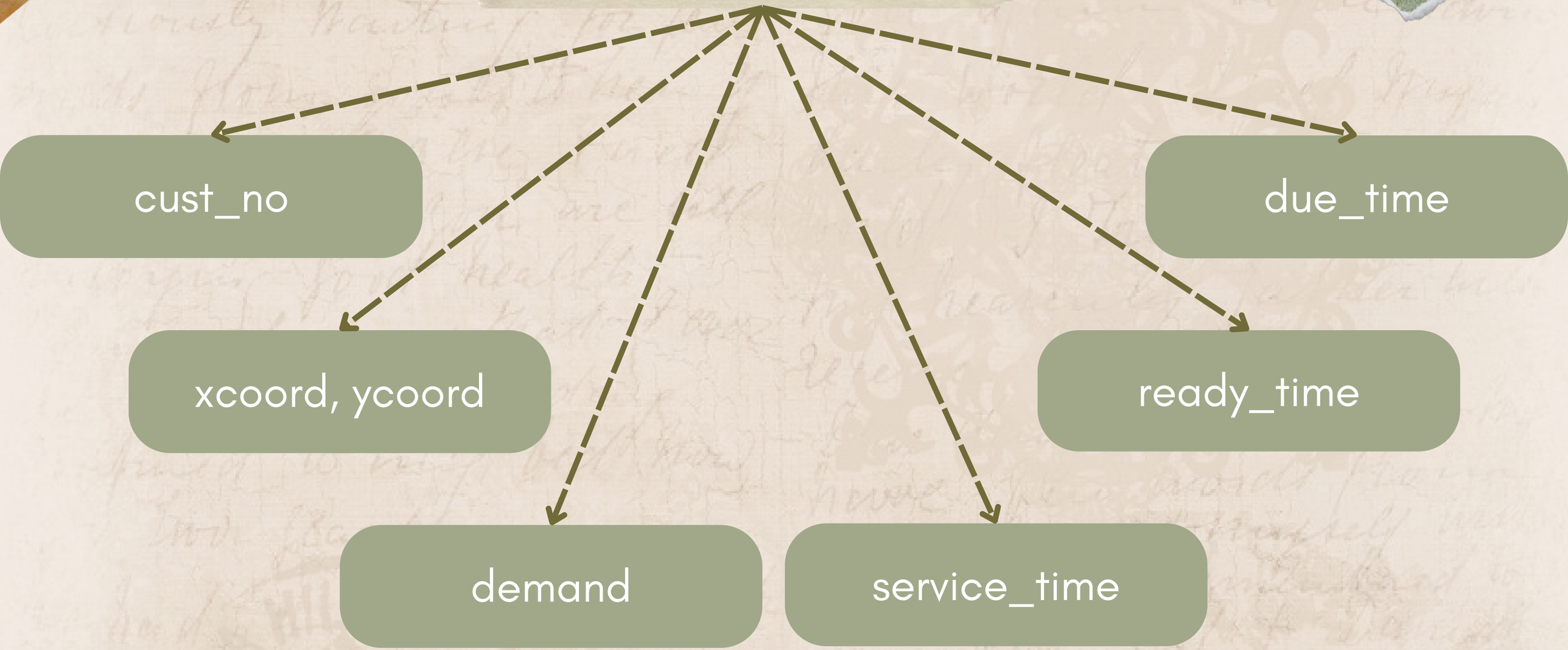
Méthodologie de résolution du VRPTW avec CF



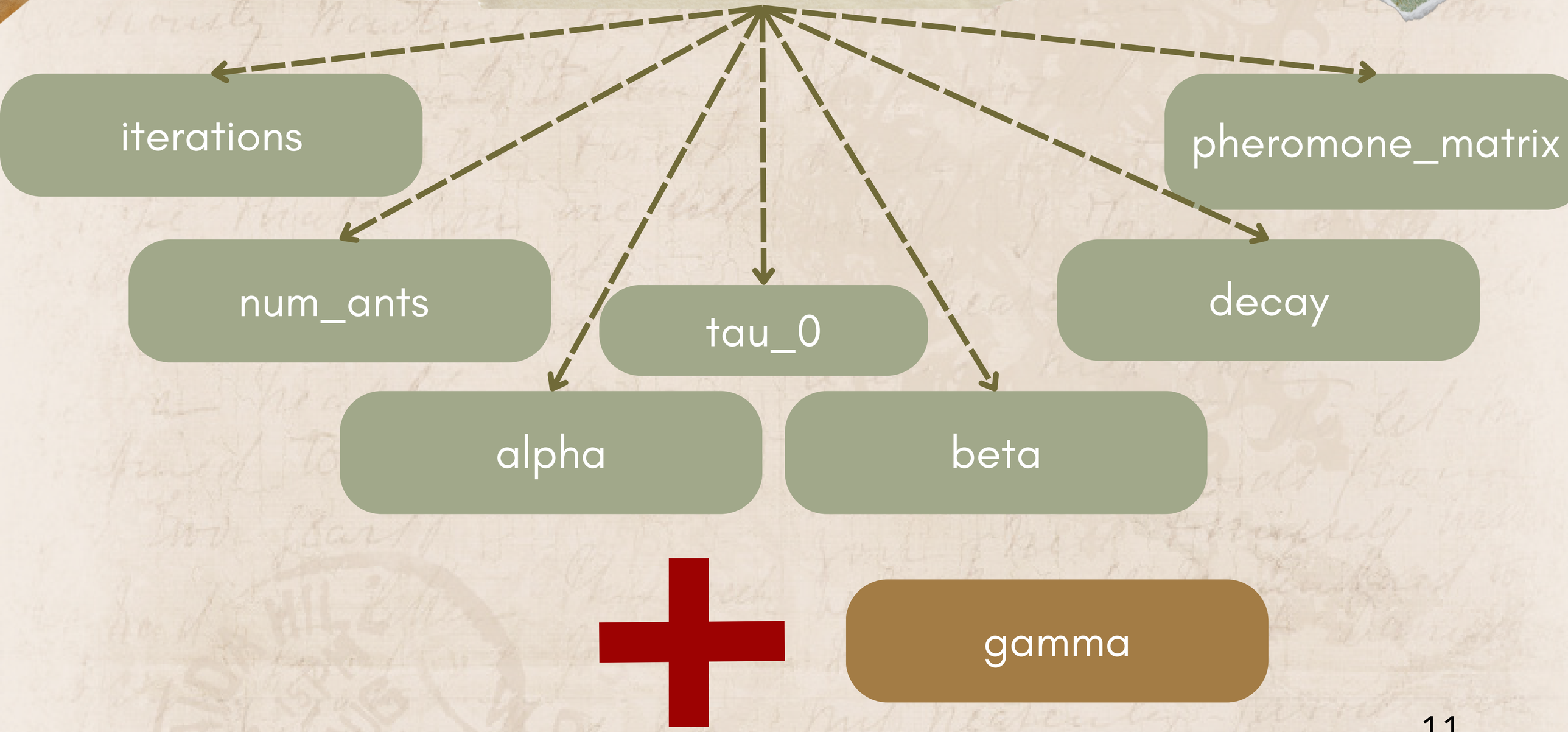
VPRTW



Customer


cust_no
due_time
xcoord, ycoord
ready_time
demand
service_time

ACS


iterations
pheromone_matrix
num_ants
tau_0
decay
alpha
beta
gamma

Fonction Main

```
Function MAIN
    best_solution ← NULL
    for I:= 1 to iterations:
        for J := 1 to num_ants:
            path, num_v, distance ← generate_solution()
            current_solution ← (path, num_v, distance))
            local_update_pheromones(path)
```

```
            if best_solution == NULL:
                best_solution ← current_solution
            else:
                best_solution ← the minimum between
                current_solution and best_solution depending on number
                of vehicule in first place then the distance
            end_if;
        end_for;
    global_update_pheromones(best_solution)
    end_for;
    return best_solution
```

update locale de pheromone

Function local_update_pheromones

pheromone_matrix I,j ←(1 - decay) . pheromone_matrix I,j + decay . Tau_0

Tau_0 = 1/(number_of_customers*nearest_neighbor_solution_length)

update globale de pheromone

Function global_update_pheromones

pheromone_matrix (I,j) ← (1 - decay) . pheromone_matrix (I,j) + decay . V

V = 0 if (I,j) is not in best_solution else = 1/distance_of_best_path

generation de solution

Function generate_solution

```
not_visited ← customer_array
time ← 0
num_v ← 1
capacity ← initial_capacity
solution ← [[],]
solution[0].append(depot)
not_visited.remove(depot)
possible ← not_visited
```

```
while len(not_visited) > 0 :
    possible, solution, time, num_v, capacity ← update_possible
    next_node ← choose_next_node(possible)
    not_visited.remove(next_node)
    capacity ← capacity - next_node.demand
    time = time + time_to_reach_nextNode_from_currentCustomer
    if time < next_node.ready_time :
        time ← next_node.ready_time + next_node.service_time
    else:
        time ← time + next_node.service_node
    end_if;
    solution[-1].append(next_node)
    end_while;
solution[-1].append(depot)
return(solution, num_V, calculate_path_length(solution))
```

update_possible

Function update_possible

```
can_be_served_by_vehicule ← False
possible ← not_visited
for customer in not_visited:
    # Check if due date is late
    if time_customer_is_reached > customer.due_date
        possible.remove(customer)
        continue
    else:
        can_be_served_by_vehicule ← True
    end_if;
    # Check if capacity exceeds the limit
    if capacity < customer.demand :
        possible.remove(customer)
    end_if;
end_for;
```

```
if len(possible) == 0 :
    solution[-1].append(depot)
    capacity ← initial_capacity
    time ← time + time_to_reach_depot
    if can_be_served_by_vehicule == False :
        time← 0
        num_V ← num_V + 1
        solution.append([depot,])
     end_if;
    solution, possible, time, num_v, capacity = update_possible()
end_if;
return solution, possible, time, num_V, capacity
```

Function choose_next_node

pheromone_values ← pheromone_matrix(current.cust_no)

visibility_values ← 0 if customerToReach_not_in_Possible else
(1/ distance_between_Current_and_customerToReach)

timeP_values ← 0 if customerToReach_not_in_Possible else 1 if
(customerToReach.ready_time <
time_after_reaching_customerToReach) else (1/-
((time_after_reaching_customerToReach) -
customerToReach.ready_time))

```
for c in customers_array :
        valueProb(c) ← pheromone_values(c)*alpha .
                              visibility_values(c)*beta . timeP_values*gamma
end_for;

total_probability ← sum(valueProb)
for c in customers_array :
        probabilities(c) ← valueProb(c)/total_probabilities
end_for;
next_node ← random_choice_depending_on_probabilities
return next_node
```

Procedure d’optimisation

improve solution

```
Function improve_solution(solution):
        # Apply tour improvement heuristic to the given solution
        solution ← optimize_routes(solution)
        # 2-opt heuristic for local search
        for i in range(len(solution)) :
              route(i)← two_opt(route(i))
         end_for;
        return solution
```

2-opt

```
Function two_opt (route):
        route_dist ← route_total_distance(route)
        improved_route← route
        while improved:
            ----------------------------
        end_while;
        return improved_route
```

```
for i in range(1, len(improved_route) - 3):
    for j in range(i+ 1, len(improved_route) - 2):
        new_route ← Swap edges (i, i+1) with (j, j+1)
        new_route_dist=route_total_distance(new_route)
        if (new_route_dist<route_dist) and new_route is valid:
            improved_route= new_route
            route_dist=new_route_dist
            improved= True
        end_if;
    end_for;
end_for;
```

Function optimize_routes(routes)**:**

sorted_routes ← Sort routes by their lengths

while improved**:**

for i **in** range(len(sorted_routes)-1)**:**

new_route1← sorted_routes(i)

Iterate over each customer in the small route

for customer **in** sorted_routes[i][1:-1]**:** # Excluding the depot

Try to insert the customer into other routes

for j **in** range(i+1, len(sorted_routes)):

new_route2 ← sorted_routes[j][:]

---

end_for; end_for; end_for;

end_while;

return sorted_routes

```
for k in range(1, len(new_route2)):
    if customer != depot:
        new_route2.insert(k, customer)
        # Check if the resulting route is valid
        if check_route_valid(new_route2):
            # Update the route with the inserted customer
            new_route1.remove(customer)
            sorted_routes[i] ← new_route1
            sorted_routes[j] ← new_route2
            improved← True
            customer_inserted ← True
            break          # Move to the next customer
```

```
                else:
                    # Revert the insertion if the route is not valid
                    new_route2.remove(customer)
                end_if;
            else :
                customer_inserted ← True
                break
            end_if;
    end_for;
    if customer_inserted:
        break          # Move to the next customer
    end_if;
```

04- Analyse

en changeant les paramètres :
alpha
beta
gamma
qu'est ce qu'on obtient ?

Analyse:

alpha	beta	gamma	Nombre de véhicules	Distances
1.5	2	2	21	2218.994138889913
0.5	2	2	22	2454.760569531627
0.5	0.5	2	26	2999.407832796353
0.5	0.5	0.5	27	2706.029993833985

Analyse:

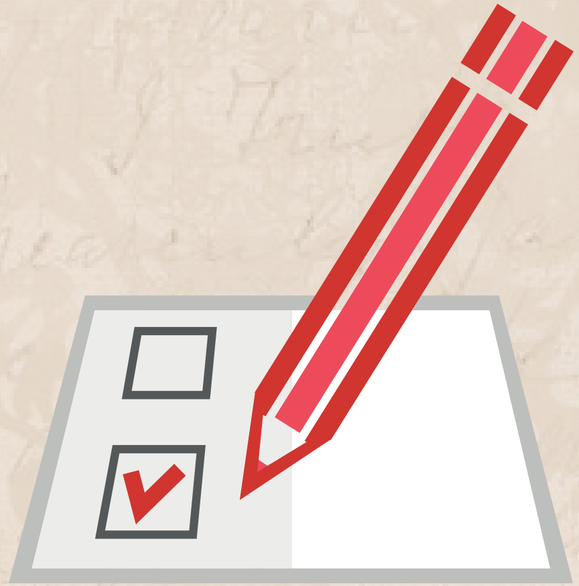
alpha	beta	gamma	Nombre de véhicules	Distances
2	1	0.5	22	2074.173013452977
1.5	3	2	20	2097.1843427051526
3	3	3	23	2499.9212638204167

Meilleure combinaison : alpha = 1.5 , beta = 3 ,gamma = 2

Interprétation

Une sélection judicieuse et une adaptation fine de ces paramètres peuvent conduire à des solutions de haute qualité dans divers contextes d'optimisation

05- Performance sur quelque instance de Solomon’s Benchmark



Instance	Nombre Vehicule	Distance	Nombre Vehicule (B)	Distance (B)
C101	10	828.93686	10	828.93664
C109	10	1180.52763	10	828.93664
C202	4	742.38762	3	591.55626
C205	3	591.55655	3	588.87566
R101	20	2051.47560	19	1650.79864
R108	11	1333.44542	9	960.87528
R207	3	1227.51542	2	890.60784
RC108	12	1467.68051	10	1139.82108
RC203	4	1581.5222	3	1049.62383

Merci Pour Votre Attention