POLYTECH TOURS
64 avenue Jean Portalis
37200 TOURS, FRANCE
Tél +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet Recherche Innovation

Logiciel d'optimisation de tournées

Etudiant :
JAMET Adrian, Étudiant en 5ème année cycle ingénieur

Tuteurs :
BILLAUT Jean-Charles, Maître de conférences, enseignant chercheur

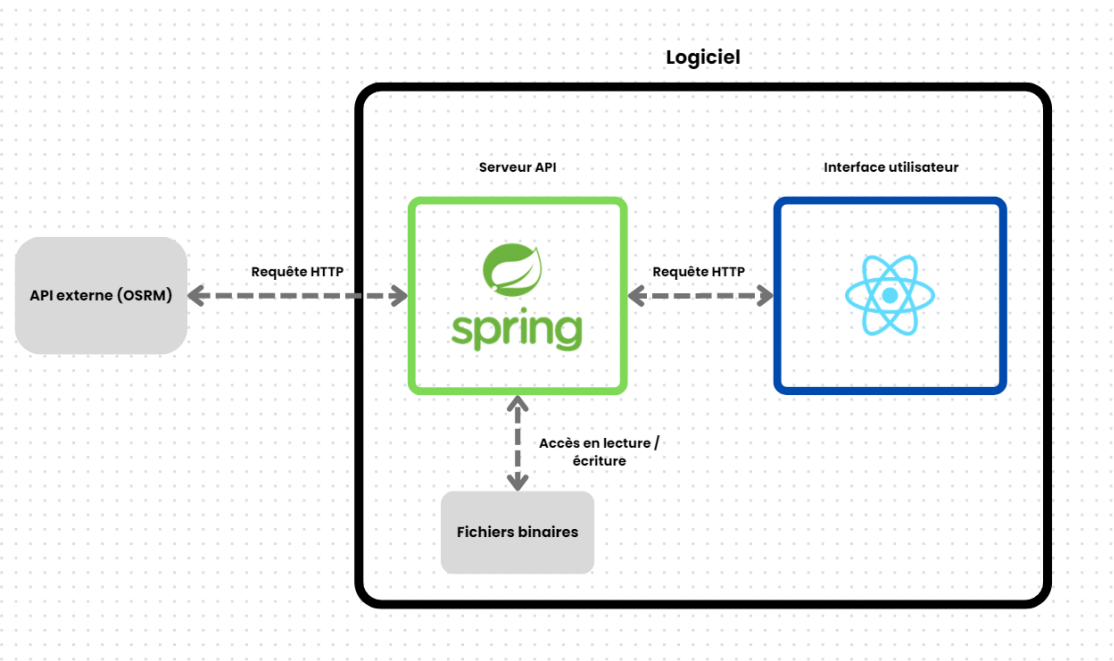
VINOT Marina, Maître de conférences, enseignante chercheuse

Table des matières

I. Contexte et objectifs

A. Historique du projet

L'entreprise Estivin avait initié le besoin d'acquérir un outil lui permettant d'optimiser ses tournées de livraison de fruits, légumes et produits maritimes. La première version du logiciel a été développée dans le cadre du projet collectif de 4A.
Cette première version intégrait un algorithme génétique pour la génération de tournées et une interface graphique permettant de gérer les variables pour les tournées, lancer l'algorithme, visualiser la solution, modifier les bons d'un véhicule à l'autre et enfin conserver la solution.
L'objectif était de pouvoir avoir accès à la solution proposée par l'algorithme afin d'éviter l'effet boîte noire. De plus, il était important que le projet n'ait pas besoin de serveur pour fonctionner afin d'éviter des coûts supplémentaires. Nous avions donc opté pour une solution utilisant Electron qui permettait d'englober une partie interface graphique avec des technologies Web actuel ainsi que le serveur backend (Spring Boot). Pour la persistance des données nous avions choisi des fichiers binaires ce qui permettait d'avoir aussi les données sur la machine de l'utilisateur la transformant ainsi en client-serveur.



Le projet s'est poursuivi lors de mon stage de 4a avec un objectif plus axé sur la recherche. L'objectif était d'analyser les résultats proposés par l'algorithme génétique en les confrontant à ceux d'un modèle linéaire sous CPLEX. Cela m'a permis d'identifier les points faibles de notre algorithme qui pouvait être coûteux en mémoire sur de grande instance et qui pouvait engendrer des solutions plus ou moins bonnes.

B. Nouveaux objectifs techniques

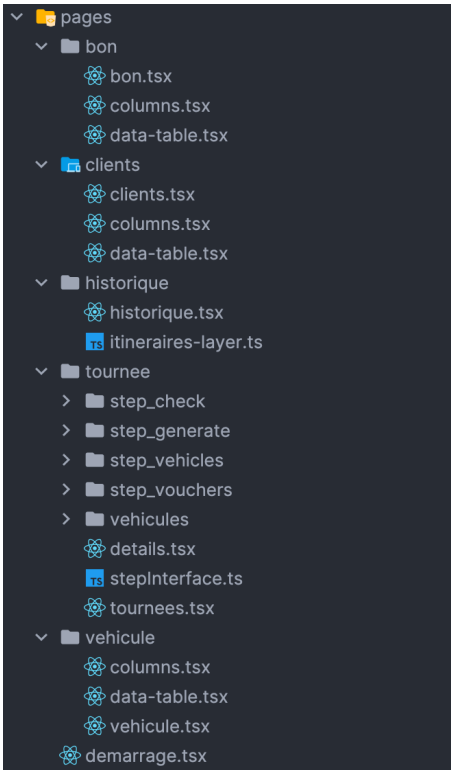
L'évolution du projet s'est orientée vers plusieurs axes d'amélioration. Premièrement, la refonte d'une partie de l'architecture front est devenue nécessaire pour améliorer la maintenabilité et la scalabilité de l'application. Cette migration de **React à NextJs** vise à moderniser la base de code et permet d'avoir des règles plus strictes en matière de programmation.
L'optimisation des performances de la carte qui permet de visualiser les tournées était un point important car la latence sur celle-ci impactait fortement l'expérience utilisateur. Le temps utilisé par l'algorithme pour générer une solution était aussi un point important pour la nouvelle version.
En plus de ces points il fallait corriger des bugs mineurs mais aussi améliorer l'éditeur de solution avec de nouvelles fonctionnalités comme la prévisualisation de la future position du bon à déplacer et pouvoir rechercher et naviguer directement vers un bon.

II. Évolution technique du projet

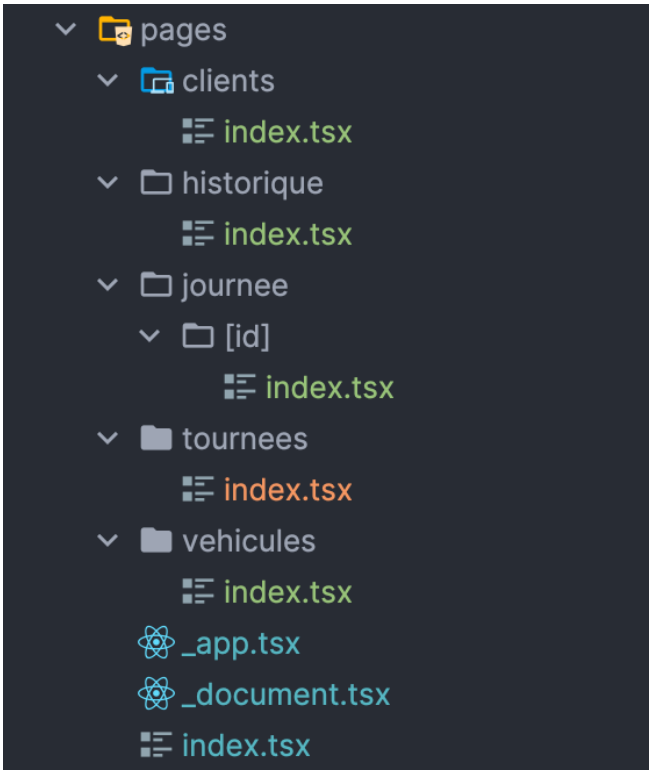
Migration vers Next.js

L'application React existante présentait des limitations en termes de performances et de maintenabilité. Next.js a été choisi pour sa capacité à gérer efficacement le routage, le rendu côté client et l'optimisation automatique des ressources.
Le nouveau système de routage basé sur les fichiers de Next.js a considérablement simplifié la navigation dans l'application. Cette approche a rendu la structure du code plus intuitive avec une organisation claire où chaque route correspond à un fichier spécifique dans l'arborescence du projet.
Le code splitting intégré assure que seul le code nécessaire est chargé pour chaque page.

Avant avec React



Après avec NextJS



A. Amélioration des performances

La principale amélioration des performances concerne la gestion des états. Initialement, l'application utilisait des useState qui rendaient difficile le maintien de la cohérence des données particulièrement pour les opérations complexes comme la gestion des tournées. L'introduction des reducers a permis de centraliser la logique de modification d'état et de mieux structurer les mises à jour des données.
Pour les composants gérant les listes de tournées et d'informations des véhicules, l'utilisation des reducers a simplifié la gestion des états CRUD.

III. Développement algorithmique

Analyse de l'existant

L'algorithme initial était un algorithme génétique structuré en trois classes principales : Algorithme, Instance et Solution.
La classe Algorithme gérait la population de solutions et leur évolution tandis que la classe Instance contenait les données du problème et Solution représentait une solution individuelle.
Dans cette version, l'initialisation des solutions était directement intégrée dans la classe Solution. Les bons étaient d'abord placés dans un ordre aléatoire puis répartis entre les véhicules en fonction de leurs capacités. Cette méthode présentait plusieurs limitations notamment le manque de contrôle sur la qualité des solutions initiales. Avec l'approche aléatoire, l'algorithme ne garantissait pas des solutions de départ pertinentes. De plus, l'algorithme ne prenait pas en compte les contraintes temporelles lors de l'initialisation. La convergence de l'algorithme dépendait donc de la qualité de l'initialisation.
De surcroit, l'évaluation des solutions était fortement couplée à leur représentation ce qui rendait difficile la modification ou l'ajout de nouvelles contraintes.

A. Conception du nouvel algorithme

La refonte de l'algorithme s'est orientée vers une heuristique.

```
Définir Fonction runAlgorithm() → Retourne Journee
    i ← 0

    bestSolution ← Initialiser la solution avec la méthode des plus proches
voisins

    currentSolution ← Copie(bestSolution)

    Tant que i ≤ NB_ITERATIONS Faire
        worstVehicles ← currentSolution.getWorstVehicles(lambdaTruck)
        worstVehicle ← Sélectionner un véhicule aléatoire de worstVehicles

        lstCustomer ← currentSolution.getCustomersForVehicle(worstVehicle)

        Si lstCustomer est vide Alors
            i ← i + 1
            Continuer
        FinSi

        customer ← Sélectionner un client aléatoire de lstCustomer
        customerIndex ← Index de customer dans lstCustomer
        Retirer customer de lstCustomer

        x ← Nombre aléatoire entre 0 et 1

        Si x < lambdaInsert Alors
            Faire
                newPosition ← Nombre aléatoire entre 0 et (taille de lstCustomer +
1)
            Tant que newPosition = customerIndex et taille de lstCustomer > 1

            Insérer customer à newPosition dans lstCustomer
            Mettre à jour worstVehicle avec lstCustomer
        Sinon
            Faire
                newVehicle ← Sélectionner un véhicule aléatoire différent de
worstVehicle
            Tant que newVehicle = worstVehicle

            newVehicleCustomers ←
currentSolution.getCustomersForVehicle(newVehicle)
            insertPosition ← Nombre aléatoire entre 0 et (taille de
newVehicleCustomers + 1)
            Insérer customer à insertPosition dans newVehicleCustomers

            Mettre à jour worstVehicle avec lstCustomer
            Mettre à jour newVehicle avec newVehicleCustomers
        FinSi
```

```
        currentSolution.evaluate()

        Si currentSolution est meilleure que bestSolution Alors
            bestSolution ← Copie de currentSolution
        Sinon
            currentSolution ← Copie de bestSolution
        FinSi

        i ← i + 1

    FinTantQue

    Retourner convertSolutionToDay(bestSolution)

Fin Fonction
```

B. Tests et optimisations

Le processus de validation et d'amélioration s'est déroulé en plusieurs phases. Dans un premier temps, il était essentiel de vérifier le bon fonctionnement de l'algorithme ce qui a nécessité une révision approfondie du modèle linéaire pour identifier et corriger des erreurs.

Une fois ces corrections apportées, j'ai dû procéder au lancement d'une instance de l'algorithme. Par la suite, j'ai dû forcer l'exécution de cette instance par le modèle linéaire sur CPLEX. Les multiples exécutions m'ont permis d'analyser les variations d'évaluation entre l'algorithme et le modèle, ce qui m'a permis de corriger les erreurs d'évaluation qui empêchaient d'obtenir les mêmes résultats des deux côtés pour la fonction objectif.

Une fois les résultats validés des deux côtés, j'ai dû observer les différences de solution proposées par l'algorithme par rapport à la solution optimale, proposée par CPLEX. J'ai donc dressé des tableaux de résultats sur trois instances différentes avec plusieurs itérations de l'algorithme pour analyser les différences de résultats et pour pouvoir constater à quel point l'algorithme donnait des solutions éloignées de la solution optimale.

Ces 3 instances ont été formés de la manière suivante :

- Une instance utilisant un camion 19 tonnes avec un véhicule léger
- Une instance utilisant deux camions moyen 12 et 13 tonnes
- Une instance utilisant un camion 12 tonnes avec un véhicule léger

L'ensemble des résultats a été répertorié dans différents tableaux au sein d'un document Excel. Le premier tableau présente les paramètres lambda (sélection aléatoire du pire

véhicule, chance d'effectuer un changement entre véhicule ou au sein d'un même véhicule) ainsi que le coût de la solution de base.

S0 :	Lambda Insert	Lamba WT
1352948.1	0,5	0,5

Tableau du coût de la solution initiale

On retrouve ensuite le tableau des coûts pour chaque partie de la fonction objectif.

Description	Coût
Livraison	492,4
Équité	11
Poids	0
Longue Journée	0
Retard	0
Avance	168
Cout total	671,4

Tableau des coûts

Le tableau des affectations avec 1 si le bon est dans le véhicule sinon 0.

ID	Nom	FL 380 NZ	DZ 285 WA
1	Ma Boulangerie C.	1	0
2	JLC Modern Tradit	1	0
3	Paineau Alain	0	1
4	Mars Fruit	1	0
5	Boulangerie Paul	0	1
6	Super U Monts	0	1
7	Clinique Vontes	0	1
8	Carrefour Grange	0	1

Tableau des affectations.

Enfin le tableau des heures d'arrivées pour chaque bon.

ID	Nom	Arrivée	Camion
1	Ma Boulangerie C.	05h43	FL 380 NZ
2	JLC Modern Tradit	09h02	FL 380 NZ
3	Paineau Alain	06h21	DZ 285 WA
4	Mars Fruit	07h14	FL 380 NZ
5	Boulangerie Paul	07h53	DZ 285 WA
6	Super U Monts	08h20	DZ 285 WA
7	Clinique Vontes	09h24	DZ 285 WA
8	Carrefour Grange	08h50	DZ 285 WA

Tableau des heures d'arrivées.

En ce qui concerne les itérations de l'algorithme un tableau supplémentaire a été rajouté afin de voir si certaines opérations permettaient d'améliorer considérablement la solution.

Historique Solutic	Valeur de	Action
38309.516	0,901282	Transfert
24425.566	0,455329	Changement de position
18660.066	0,92053	Transfert
11825.15	0,76562	Transfert
10809.45	0,326984	Changement de position

Pour les résultats choisis *(voir la feuille de calcul en annexe)* j'ai sélectionné des itérations dont les résultats étaient relativement différents.

D'après les résultats obtenus, on peut voir que les opérations de transfert dans la plupart des cas améliorent grandement la solution. On remarque aussi que le coût de la solution initiale est très élevée. L'étape suivante a donc été de trouver un meilleur moyen d'initialiser la solution de base.

En effet, lors de l'élaboration de l'algorithme, il a été décidé d'initialiser la solution avec la méthode des plus proches voisins. Il a donc fallu revoir le code pour pouvoir tester différentes manières d'initialiser la solution. J'ai donc choisi d'utiliser deux patrons de conception : le patron Stratégie et le patron Fabrique. Cela m'a permis de refactoriser le code de la classe Solution afin que celle-ci ne contienne que la logique d'évaluation.

	NEAREST NEIGHBOR	**ALTERNATING TIME WINDOW**	**TIME WINDOW**
INSTANCE 1	1 352 948.1	22 586.0	1 158.72
INSTANCE 2	7 365.5	6422.4	14 997.75
INSTANCE 3	1 352 948.1	1 487 004.6	1 656 327.1

Tableau des coûts de la solution de base selon la méthode d'initialisation et les différentes instances.

En ce qui concerne la nouvelle manière d'initialiser, à la suite de tests approfondis, il est apparu que l'approche alternative basée sur les fenêtres temporelles (**AlternatingTimeWindow**) donnait le meilleur résultat sur l'instance 2. Cette stratégie fonctionne en deux phases principales :

- **Tri initial des bons de livraison** : Les bons sont triés selon deux critères hiérarchiques :
  L'heure de début de la fenêtre de livraison comme critère principal
  La durée de la fenêtre comme critère secondaire, privilégiant les fenêtres les plus courtes

- **Attribution alternée aux véhicules** : L'algorithme attribue les bons aux véhicules de manière cyclique tout en respectant les contraintes de capacité :

    - Si le véhicule courant peut accepter le bon, il lui est attribué
    - Si le véhicule est plein, l'algorithme cherche le prochain véhicule disponible
    - En dernier recours, le bon est attribué au véhicule le moins chargé

On peut voir aussi que la méthode **TimeWindow** est meilleure sur l'instance 1. Elle fonctionne de la manière suivante :

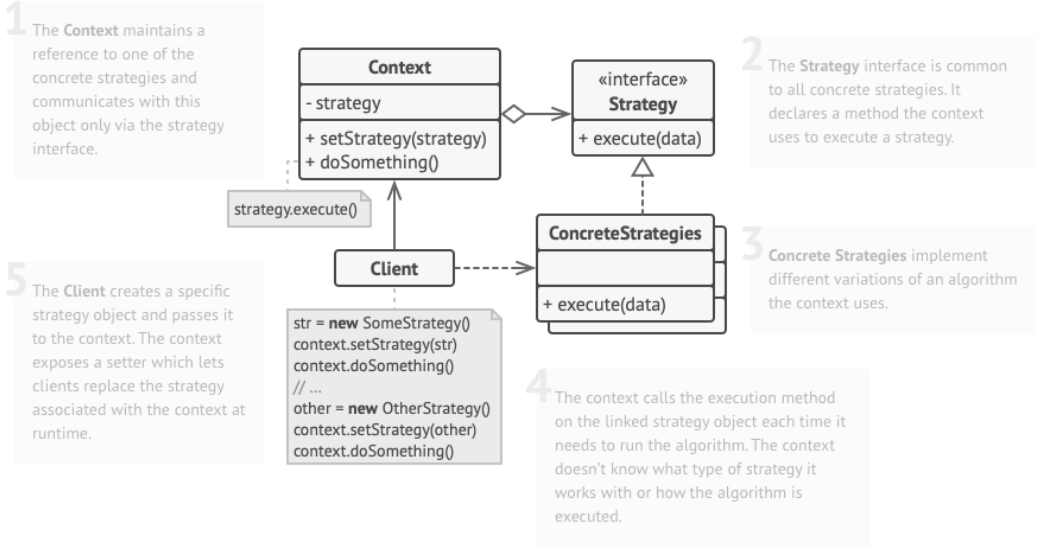
- **Les clients sont d'abord triés** selon leurs fenêtres de temps, en prenant en compte :
    - L'heure de début de la fenêtre de livraison
    - La durée de la fenêtre (pour privilégier les clients ayant moins de flexibilité)
    - L'adresse du client comme critère de départage
- Pour chaque client, l'algorithme **cherche le meilleur itinéraire possible** en calculant un score qui combine :
    - La déviation temporelle par rapport au centre de la fenêtre de livraison (pondération de 0.6)
    - L'impact sur la durée totale de la tournée (pondération de 0.4)
    - Si aucun itinéraire satisfaisant n'est trouvé (à cause des contraintes de capacité ou de temps) le client est assigné au véhicule le moins chargé.

La conclusion de ce test a été d'implémenter une vérification dans l'algorithme au moment de l'initialisation de la solution de base afin de vérifier quelle manière d'initialisée est la plus appropriée pour l'instance à tester.

```
    bestSolution ← MIN(Créer une Solution pour chaque mode d'initialisation,
comparer par coût total)
```

Modification de l'initialisation de l'algorithme

J'ai fait le choix d'utiliser le **patron Stratégie** pour qu'il devienne facile d'implémenter et de tester de nouvelles approches d'initialisation sans modifier le reste du code. Cette flexibilité a été particulièrement utile pendant la phase de développement et de test des différentes stratégies.



1. Test des lambdas

Ce test visait à évaluer si une combinaison de valeurs de *lambda_truck* (qui influence la sélection du véhicule à optimiser) et de *lambda_insert* (qui détermine le type de mouvement sur les bons de livraison) permettait d'atteindre systématiquement la solution optimale ou au mieux de toujours la trouver.

J'ai donc utilisé les 3 instances précédentes que j'ai exécuté 3 fois en testant différentes combinaisons de lambda. J'ai ensuite calculé des moyennes et d'autre indicateurs que j'ai classé dans un tableau.

lambda_truck	lambda_insert	cout_moyen	cout_min	cout_max	variance	ecart_type	mps_livraison_moy	equite_moyenne	retard_moyen	avance_moyenne
0,00	0,00	996,39	955,03	1 011,77	5,43E+02	23,30	493,69	23,20	37,00	442,50
0,00	0,10	883,06	765,58	978,75	1,19E+04	109,28	520,06	17,00	37,00	309,00
0,00	0,30	896,09	765,73	978,75	1,28E+04	113,28	515,15	19,60	0,00	361,33
0,00	0,50	968,30	926,50	978,75	5,46E+02	23,37	503,43	14,20	37,00	413,67
0,00	0,70	915,25	765,73	978,75	7,67E+03	87,57	519,45	12,80	74,00	309,00
0,00	0,90	925,70	765,73	978,75	8,51E+03	92,24	511,76	15,60	37,00	361,33
0,00	1,00	1 158,72	1 158,72	1 158,72	0,00E+00	0,00	483,05	78,00	0,00	597,67
0,10	0,00	958,41	807,83	1 012,37	7,65E+03	87,49	514,04	17,20	37,00	390,17
0,10	0,10	968,30	926,50	978,75	5,46E+02	23,37	503,43	14,20	37,00	413,67
0,10	0,30	947,40	926,50	978,75	8,19E+02	28,62	518,80	8,60	111,00	309,00
0,10	0,50	978,75	978,75	978,75	0,00E+00	0,00	495,75	17,00	0,00	466,00
0,10	0,70	896,06	765,58	978,75	1,28E+04	113,32	515,12	19,60	0,00	361,33
0,10	0,90	916,79	773,47	978,75	7,10E+03	84,27	521,99	11,80	74,00	309,00
0,10	1,00	1 158,72	1 158,72	1 158,72	0,00E+00	0,00	483,05	78,00	0,00	597,67
0,30	0,00	976,38	955,03	1 011,77	8,60E+02	29,33	510,35	17,20	111,00	337,83
0,30	0,10	957,85	926,50	978,75	8,19E+02	28,62	511,12	11,40	74,00	361,33
0,30	0,30	978,75	978,75	978,75	0,00E+00	0,00	495,75	17,00	0,00	466,00
0,30	0,50	925,67	765,58	978,75	8,52E+03	92,31	511,73	15,60	37,00	361,33

Extrait du tableau des résultats

L'objectif était de rechercher la combinaison donnant un coût minimum et le coût moyen le moins élevé. J'ai pu remarquer qu'une combinaison de valeur donnait toujours le coût minimum.

1,00	0,50	855,29	670,98	944,00	1,61E+04	126,76

Instance 1 (lambda truck 1 lambda insert à 0,5)

1,00	0,50	645,41	570,17	765,73	6,56E+03	80,99

Instance 2 (lambda truck 1 lambda insert à 0,5)

1,00	0,50	1 098 481,00	719 836,30	1 351 119,20	1,19E+11	345 653,76

Instance 3 (lambda truck 1 lambda insert à 0,5)

IV. Nouvelle version de l’algorithme

Le paramétrage des lambdas a permis de réduire la variabilité des solutions. En revanche, j'ai pu remarquer que l'algorithme stagnait sur certaines solutions. L'algorithme se trouvait coincé dans des optimums locaux à cause du choix de toujours repartir de la meilleure solution. Deux choix s'offraient alors :

- Le premier était d'accepter de repartir de n'importe quelle solution puis de vérifier si celle-ci était meilleure que la meilleure solution sauvegardée.
- Le second était d'agir sur les paramètres de l'aléatoire pour essayer de diriger davantage les choix afin de ne pas toujours faire le même type d'opération.

Ces observations m'ont conduit à chercher des algorithmes ou des procédés permettant de réaliser l’un des choix. J’ai donc trouvé un compromis avec le recuit simulé. Cette métaheuristique présente l'avantage de pouvoir accepter temporairement des solutions moins bonnes permettant ainsi d'explorer plus largement l'espace des solutions et d'échapper aux optimums locaux.

Le choix du recuit simulé s'est également justifié par sa capacité à gérer les plateaux, ces zones de l'espace des solutions où plusieurs configurations présentent des coûts très similaires.

Le recuit simulé tire son inspiration d'un processus physique utilisé en métallurgie où un métal est chauffé à haute température puis refroidi progressivement pour améliorer ses propriétés. Dans le cas de l’algorithme, cela permet d’accepter des solutions dégradés au début (haute température) pour devenir progressivement plus précis (refroidissement).

Dans la version de l’algorithme utilisant le recuit simulé, la température initiale est fixée à 100.0 et diminue progressivement selon un taux de refroidissement de 0.995. À chaque itération, l'algorithme évalue une modification de la solution courante. Si cette modification améliore la solution, elle est automatiquement acceptée. Dans le cas contraire, elle peut être acceptée selon une probabilité qui dépend de deux facteurs : l'ampleur de la dégradation et la température actuelle. Cette probabilité est donnée par la formule exp(-delta/T), où delta représente la dégradation du coût et T la température courante.

```
Définir Fonction SimulatedAnnealingMultiStart() → Retourne Journee
    NB_ITERATIONS ← 300000
    INITIAL_TEMPERATURE ← 100.0
    FINAL_TEMPERATURE ← 0.1
    COOLING_RATE ← 0.995
    NB_RESTARTS ← 100

    bestGlobalSolution ← NULL
    bestGlobalCost ← ∞

    Pour start ← 1 à NB_RESTARTS Faire
        currentSolution ← SimulatedAnnealing()

        Si currentSolution.totalCost < bestGlobalCost Alors
            bestGlobalSolution ← currentSolution
            bestGlobalCost ← currentSolution.totalCost
        FinSi
    FinPour
    Retourner ConvertSolutionToSchedule(bestGlobalSolution)
Fin Fonction
```

```
Définir Fonction SimulatedAnnealing() → Retourne Solution
    temperature ← INITIAL_TEMPERATURE
    noImprovementIterations ← 0
    plateauCount ← 0
    bestSolution ← InitializeSolution()
    currentSolution ← Copie(bestSolution)
    previousCost ← currentSolution.totalCost
    i ← 0

    Tant que i ≤ NB_ITERATIONS et temperature > FINAL_TEMPERATURE Faire
      neighbors ← Générer des voisins (currentSolution, 5)
      bestNeighbor ← Récupérer le meilleur voisin

      delta ← bestNeighbor.getTotalCost() – currentSolution.getTotalCost()
      Si delta < 0 ou Nombre Aléatoire < Exponentielle(-delta / temperature)
            currentSolution ← Nouvelle Solution(bestNeighbor)

            Si currentSolution.getTotalCost() < bestSolution.getTotalCost()
                  bestSolution ← Nouvelle Solution(currentSolution)
                  noImprovementIterations ← 0
                  plateauCount ← 0
            Sinon
                  noImprovementIterations++
            FinSi
      FinSi
```

```
      Si |currentSolution.totalCost - previousCost| < 0.001 Alors
            plateauCount++
      Sinon
            plateauCount ← 0
      FinSi
      previousCost ← currentSolution.getTotalCost()
      Si noImprovementIterations > NB_ITERATIONS / 20 Alors
            temperature ← Min(INITIAL_TEMPERATURE, temperature
                              / COOLING_RATE * 1.5)
            noImprovementIterations ← 0
            currentSolution ← Appliquer plusieurs changement sur currentSolution
      Sinon
            temperature ← temperature * COOLING_RATE
      FinSi

      i++
    FinTantQue
    Retourner bestSolution
Fin Fonction
```

Le principal problème était la gestion des plateaux. Pour pouvoir empêcher la stagnation de l'algorithme j'ai mis en place une vérification pour savoir si l'algorithme se trouve dans un plateau.

```
Si |currentSolution.totalCost - previousCost| < 0.001 Alors
    plateauCount ← plateauCount + 1
Sinon
    plateauCount ← 0
FinSi
```

Lorsqu'un plateau est détecté, l'algorithme applique une perturbation plus forte à la solution courante. Cette perturbation consiste à effectuer plusieurs mouvements aléatoires consécutifs sur les bons permettant à l'algorithme de d'aller vers une région différente de l'espace des solutions.

Si aucune amélioration est détectée pendant un certain nombre d'itérations la température est légèrement augmentée offrant une nouvelle chance d'explorer d'autres régions de l'espace des solutions. Pour être sûr d'obtenir la meilleure solution, l'algorithme va être lancé plusieurs fois et seule la solution ayant le coût le plus bas sera retournée. Dans la situation actuelle cela permet de retourner toujours la solution optimale.

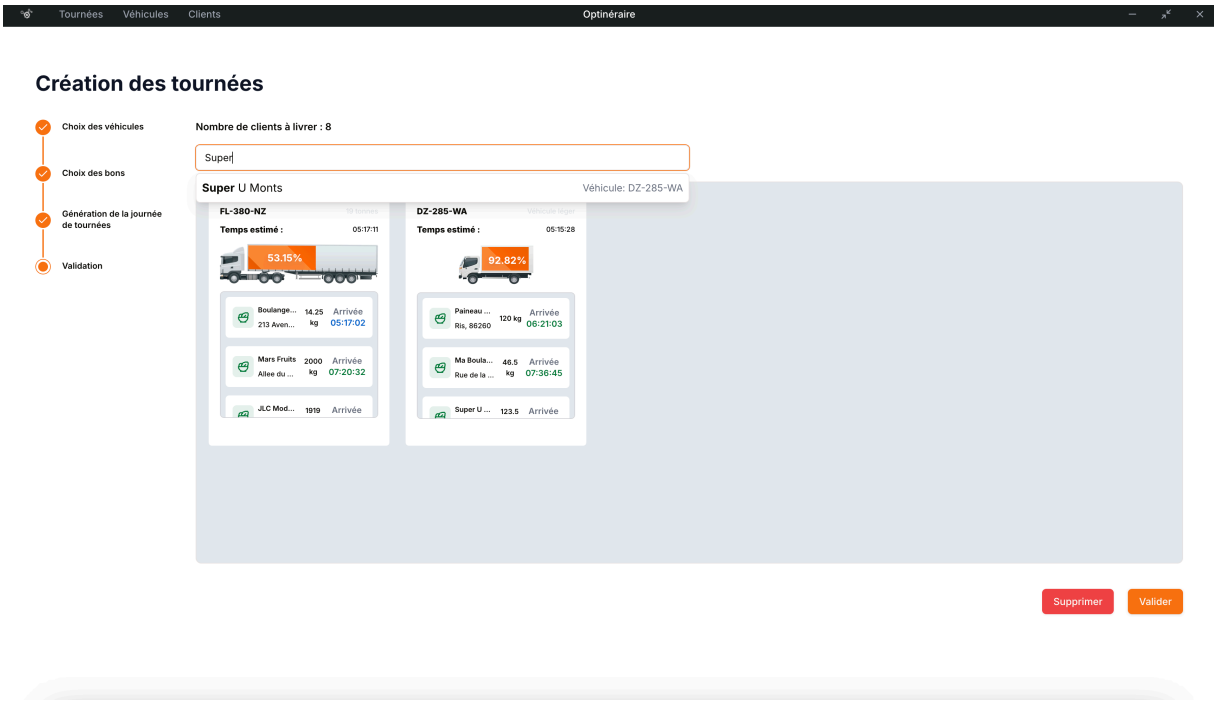
V. Améliorations de l'interface utilisateur

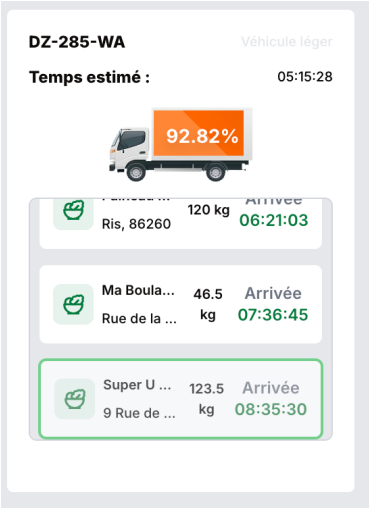
Nouvelles fonctionnalités

Le développement des nouvelles fonctionnalités s'est concentré sur l'amélioration de l'expérience utilisateur notamment sur la manipulation des tournées et la visualisation des informations.

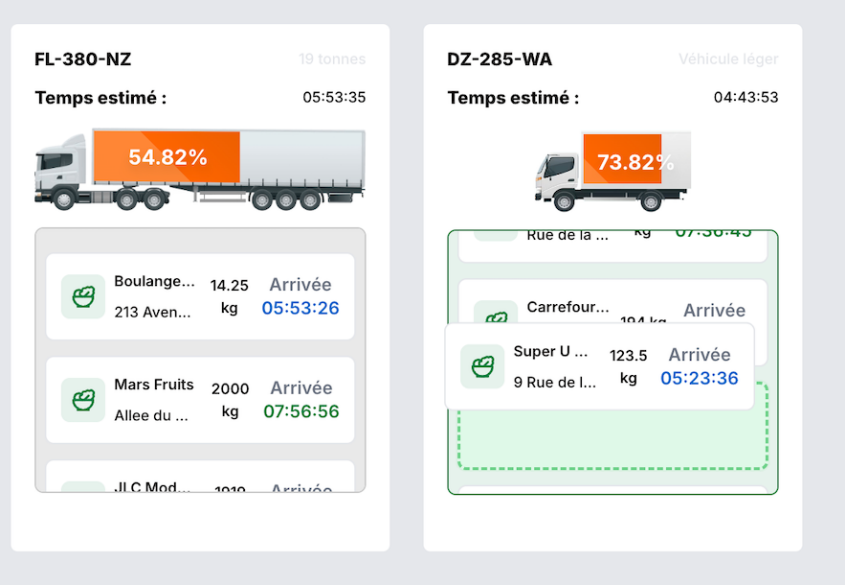
La première amélioration concerne la recherche et la localisation des bons de livraison. L'interface précédente obligeait les utilisateurs à parcourir manuellement l'ensemble des tournées pour retrouver un client spécifique ce qui s'avérait difficile lorsque la solution utilise une dizaine de véhicules et une centaine de bon. Pour résoudre ce problème, j'ai développé une barre de recherche qui permet non seulement de localiser instantanément un client, mais qui déclenche également un défilement automatique vers le véhicule concerné, avec une mise en surbrillance du bon recherché.

L'implémentation de cette fonctionnalité a nécessité une refonte du système de navigation dans les tournées, l'ajout d'un marquage visuel sur le bon et le développement du défilement vers le bon exact.





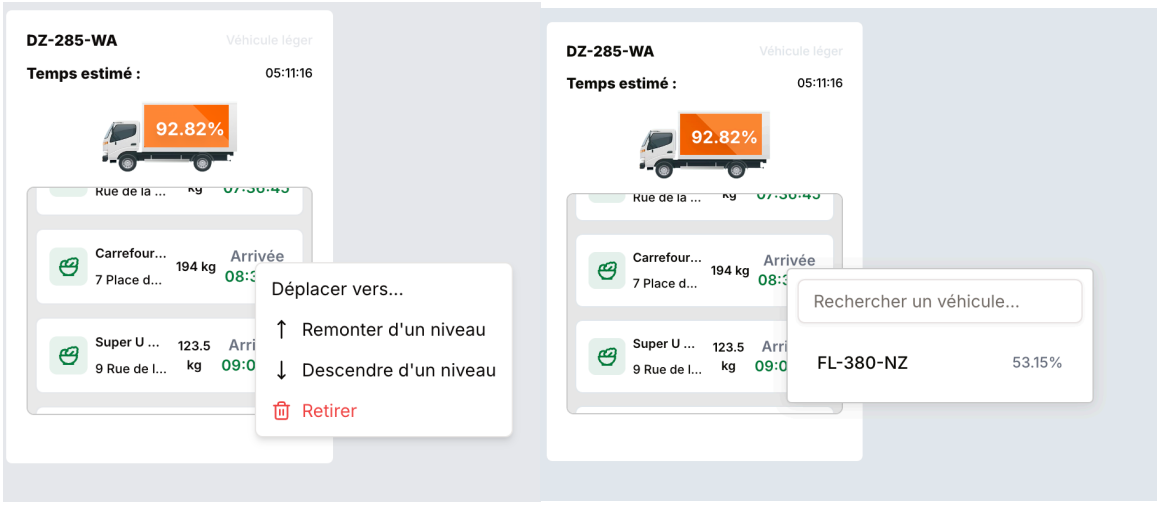
L'interaction avec les bons de livraison a également été repensée pour être plus intuitive. Le système de glisser-déposer a été enrichi d'une fonctionnalité de prévisualisation, permettant aux utilisateurs de voir exactement où leur bon sera positionné avant de le déposer. Pour implémenter cette fonctionnalité, j'ai dû développer un système de gestion des interactions utilisant la bibliothèque GSAP (GreenSock Animation Platform).



Le développement s'est articulé autour d'un hook (*fonction spéciale qui permet d'ajouter des fonctionnalités supplémentaires aux composants fonctionnels*) personnalisé React qui gère l'ensemble des interactions de glisser-déposer. Ce hook crée un placeholder qui s'affiche pendant le déplacement du bon afin de donner une prévisualisation du futur emplacement du bon. Le placeholder est un élément div avec une bordure en pointillés et une couleur de fond.

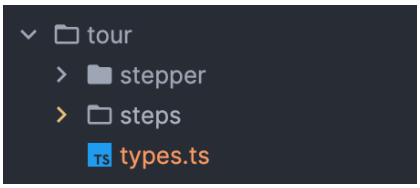
Le système de positionnement du placeholder a été compliqué notamment pour déduire la future place mais aussi lorsque le bon est inséré en premier ou dernière place. J'ai

implémenté une logique détection de position qui divise chaque zone de dépôt en sections. Cela permet de calculer précisément où insérer le placeholder en fonction de la position du curseur. Pour améliorer la précision j'ai créé des "zones de détection" autour de chaque position possible permettant une insertion plus naturelle et évitant les changements brusques de position.
La gestion du défilement automatique a également été une partie importante. Lorsqu'un utilisateur déplace un bon vers le haut ou le bas d'une zone de dépôt le système détecte la proximité avec les bords et déclenche un défilement automatique. Cela facilite le réordonnancement des bons dans les listes longues.
La manipulation des bons a été encore améliorée par l'ajout d'un menu contextuel. Ce menu offre désormais trois options principales : la recherche de destination pour un déplacement précis, le réordonnancement au sein d'un même véhicule et l'accès à des informations détaillées sur les bons.

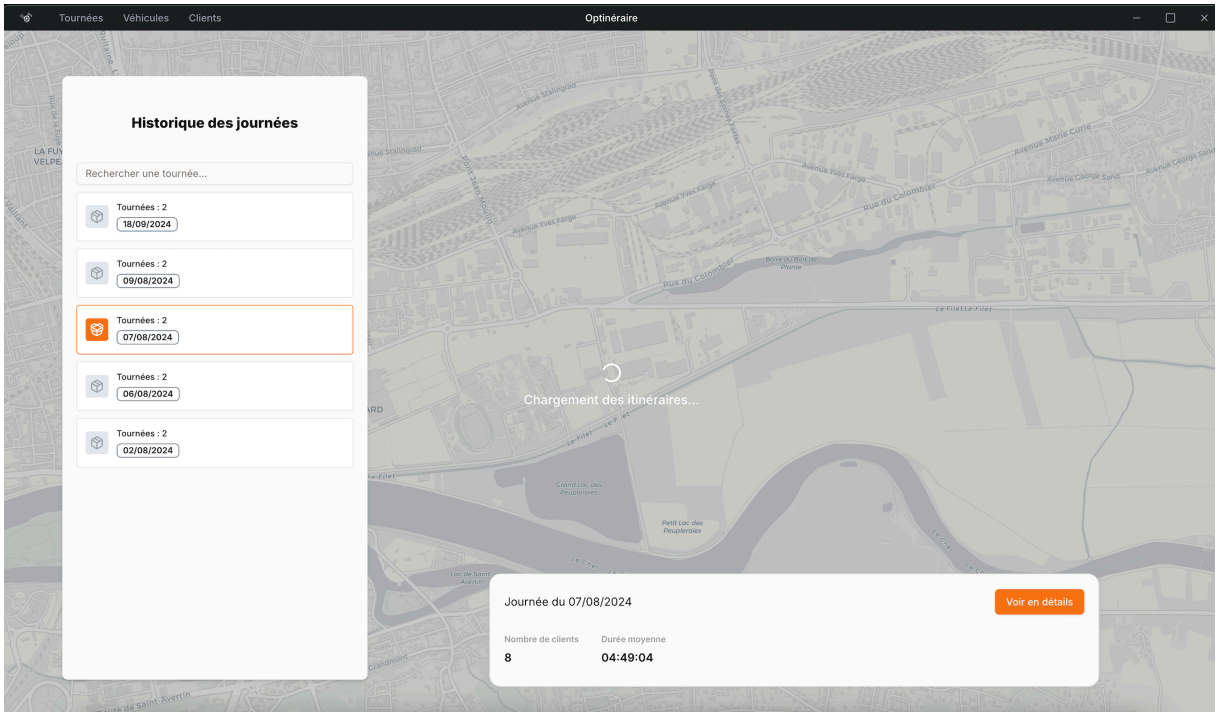


A. Corrections et optimisations

Le travail d'optimisation s'est concentré sur trois axes principaux : l'architecture de l'application, les performances graphiques, et l'expérience utilisateur globale.
La restructuration de la partie "étapes" de l'algorithme représente une amélioration architecturale majeure. Initialement, cette section était gérée par un composant monolithique qui générait des problèmes de chevauchement d'interface et compliquait la maintenance. La nouvelle architecture divise le processus en composants distincts, chacun gérant sa propre logique et son état. Cette approche modulaire a non seulement résolu les problèmes d'interface, mais a également simplifié le processus de développement et de débogage. J'ai aussi rajouté des animations entre les changements d'étapes afin d'avoir une meilleure expérience utilisateur.

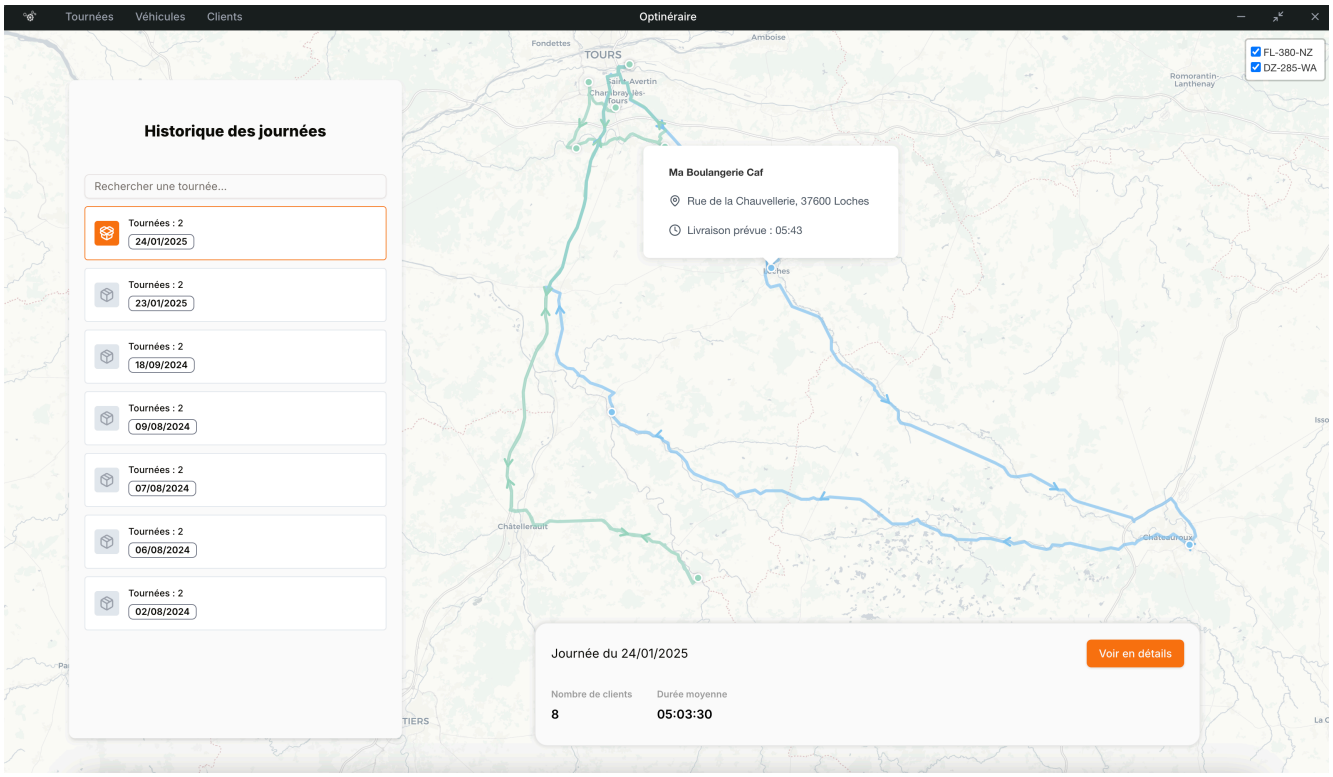


Les performances de l'affichage cartographique étaient à revoir. L'activation de l'accélération matérielle a constitué une première étape mais c'est la réécriture complète du système d'affichage des itinéraires qui a apporté les améliorations les plus significatives. Le nouveau code optimise la gestion de la mémoire et réduit les calculs redondants. Pour gérer les temps de chargement des itinéraires, j'ai implémenté un écran de chargement qui permet de faire charger correctement les itinéraires sur la carte.



Les problèmes de mémoire et de rendu qui survenaient lors des changements de page ont été résolus par l'implémentation d'un système de nettoyage des composants. Ce système assure une libération propre des ressources et prévient les erreurs d'affichage qui pouvaient perturber auparavant l'expérience utilisateur lorsque l'utilisateur changeait de page trop rapidement alors que les itinéraires étaient ajoutés à la carte.

Enfin, l'interface a bénéficié d'une refonte visuelle complète. Le système de couleurs des icônes a été revu pour améliorer la lisibilité et un code couleur a été mis en place dans la partie vérification des solutions pour identifier rapidement les avances et retards.



VI. Résultats et perspectives

Bilan des améliorations

Les améliorations apportées au projet ont permis d'appréhender davantage ce qu'est la recherche. Une des leçons importantes de ce projet a été l'importance cruciale de la phase de test et de validation. Cette phase est démoralisante car elle occupe beaucoup de temps durant lequel on ne développe pas.

La période consacrée aux tests a été particulièrement exigeante. Des journées entières ont été dédiées à l'analyse des résultats, à la comparaison des différentes approches et à l'ajustement des paramètres. Ce travail minutieux, bien qu'il ne se traduise que par quelques lignes d'analyse a été déterminant pour comprendre le comportement de l'algorithme et pour identifier les points d'amélioration.

A. Points d'amélioration

L'analyse du projet fait émerger plusieurs axes d'amélioration significatifs. Une des améliorations majeures concerne l'intégration du module OSRM (Open Source Routing Machine) directement dans le projet. Cette modification permettrait non seulement d'améliorer les données de distance en les rendant plus précises et mieux adaptées aux spécificités des véhicules de livraison. Cette évolution nécessiterait le développement d'un profil Lua spécifique pour les camions prenant en compte leurs contraintes particulières de circulation (rayon de braquage, tonnage, hauteur, etc...). Ce travail pourrait constituer à lui seul un projet.

Du point de vue du code, plusieurs aspects méritent d'être repensés. La nomenclature des routes dans les contrôleurs ne respecte pas les bonnes pratiques de nommage et pourrait compliquer la maintenance future du code. D'un point de vue plus global, il y a un mix d'anglais-français dans le code qui doit être corrigé afin d'éviter des confusions ou même de perdre du temps dans la recherche de fonction.

Une difficulté supplémentaire rencontrée concerne l'équilibre entre composantes aléatoires et déterministes dans l'algorithme. L'expérience a montré que l'aléatoire joue un rôle crucial dans la convergence vers de bonnes solutions en permettant d'explorer plus largement l'espace des possibilités. Cependant, il est apparu tout aussi important de guider cette exploration par des choix pragmatiques basés sur les caractéristiques du problème.

B. Perspectives

Ce projet a été riche en enseignements notamment sur la façon d'aborder un problème complexe d'optimisation. Il m'a permis de développer une approche plus structurée dans l'élaboration de solutions en apprenant à équilibrer les aspects théoriques et pratiques.

La méthodologie de développement adoptée alternant phases de conception, d'implémentation et de test s'est révélée efficace même si parfois frustrante. Cette expérience m'a appris l'importance de la patience et de la rigueur dans le développement d'algorithmes complexes où les résultats visibles ne sont pas toujours immédiats.