Une méthode de réparation locale pour des routes maritimes précises

1st Alexandre Coppé *Aix-Marseille Université*
LIS UMR 7020
France
alexandre.coppe@univ-amu.fr

2nd Nicolas Prcovic *Aix-Marseille Université*
LIS UMR 7020
France
nicolas.prcovic@univ-amu.fr

7 mai 2026

Abstract

Nous présentons un algorithme permettant de calculer un trajet maritime aussi précis que l'on veut. Contrairement aux approches classiques où les trajets générés sont des suites de transitions entre des lieux de passages prédéterminés par une grille rectangulaire donnée, les trajets que nous calculons ont des lieux de passages librement localisés de manière aussi précise géographiquement que l'on veut. Notre algorithme est basé sur une méthode de réparation locale de trajet, ce qui permet au calcul d'être très rapide. Nous vérifions l'efficacité de notre approche en réalisant des expérimentations à partir de données météorologiques réelles.

1 Introduction

Plus de 80% du volume des échanges commerciaux mondiaux se traite par la mer. En 2018, au sein du transport maritime, 40% des frais opérationnels étaient absorbés par le coût du carburant lors d'un voyage. Une petite amélioration, aussi minime soit-elle, peut avoir de grandes répercussions au niveau des coûts. Le routage des navires en fonction de la météo est donc un domaine qui génère un grand intérêt pour des raisons écologiques et économiques.

L'objectif de nos travaux consiste à trouver un trajet de navire entre deux ports, en arrivant avant une date donnée, en optimisant la consommation de carburant et en prenant en compte les contraintes environnementales, tout ceci dans un contexte qui varie dynamiquement au cours du trajet (courants, météo).

En plus de la géographie maritime, des lieux et dates de départ et d'arrivée, nous devons tenir compte des courants maritimes et de la météo. Ceux-ci nous sont données sous forme d'un pavage du globe découpé en cellules rectangulaires à l'intérieur desquelles le courant et le vent sont considérés comme étant uniformes (cf figure 1). Ces données nous permettent de calculer les différents coûts (temps, carburant) liés au trajet qui sont fonctions du vent et des courants à un moment donné. Ces prévisions ne sont valables que pour une période donnée (typiquement, pendant 6h) et, pour un trajet, il nous faut récupérer les prévisions pour les périodes incluses entre la date de départ et d'arrivée du navire.

Bien que la modélisation la plus réaliste d'un trajet de navire soit une courbe continue, la numérisation des données

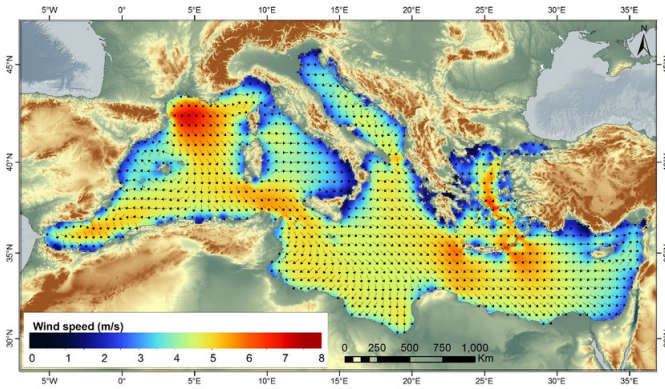

FIGURE 1 – Exemple d'une grille météo. Les flèches indiquent la direction du vent et les couleurs représentent la vitesse du vent.

implique leur discrétisation. L'espace et le temps numérisés seront donc un ensemble de lieux et de dates dont le nombre et la précision du repérage influeront sur l'efficacité des algorithmes et la qualité des solutions produites. Bien souvent, les lieux sont répartis uniformément sur une grille rectangulaire et forment un maillage couvrant toute la zone de déplacements possibles en se calquant sur la grille météo.

L'approche la plus classique est de représenter les lieux par un graphe dont les arcs relient les sommets correspondant aux lieux les plus proches (cf fig 2). Les arcs sont étiquetés par un vecteur de coûts (contenant au moins ceux du temps et du carburant utilisés pour joindre les deux lieux).

Des outils calculant des trajets dans ce contexte existent déjà et sont utilisés par des entreprises de routage maritimes. Cependant la localisation prédéterminée des lieux de passages sur une grille rectangulaire pêchent par plusieurs aspects :

— Les lieux de passages prédéterminés ne sont pas forcément bien placés. Idéalement, on voudrait pouvoir les placer avec une très grande précision.
— Les transitions (en ligne droite) entre lieux de passage se font selon quelques valeurs d'angles possibles (0°, 45°, 90°, etc). Ainsi, les trajectoires subissent un effet de crénelage, ce qui, outre l'aspect

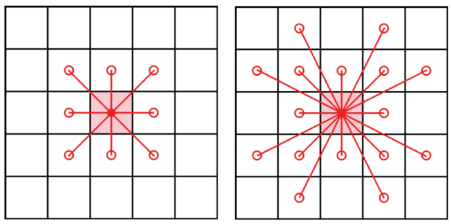

FIGURE 2 – Exemple de configurations à 8 et 16 voisins, où les sommets correspondent à des lieux uniformément distribués sur une grille rectangulaire, comme dans [Chauveau et al., 2017].

irréaliste du trajet, induit un allongement artificiel des distances parcourues.

— Les transitions (en ligne droite) entre lieux de passage sont limités à quelques voisins (typiquement 4 ou 8) à distance fixée. Or, il serait mieux que les distances entre lieux de passages d'une trajectoire soient très variables : longues si on est en pleine mer sur des courants et une météo constantes, courtes lorsqu'on traverse un bras de mer sinueux.

— Les données météo sont fiables dans un temps court mais imprécises dans un temps long. Une densité de lieux de passage plus élevée vers le début du trajet qu'à la fin est donc préférable, sachant qu'un trajet peut être recalculé plusieurs fois avant que le navire soit arrivé à destination.

Le plan de cet article le suivant. Nous commençons par donner une définition formelle du problème de recherche d'un trajet optimal dans un contexte multi-objectif et dynamique. Puis, nous présentons les algorithmes existants permettant de traiter ce problème. Ensuite, nous présentons notre approche en deux temps : d'abord nous décrivons notre méthode de réparation de trajet, ensuite, nous présentons différentes manières de générer rapidement un trajet initial destiné à être réparé ultérieurement. À chaque étape, nous présentons des expérimentations à partir de données réelles fournies par l'entreprise de transport maritime avec laquelle nous collaborons.

2 Recherche d'un chemin optimal dans un graphe

2.1 Définition formelle du problème

Considérons un graphe orienté et muni d'une fonction de coût c défini par : G = (N, A , c) où N = $\{x_1..., x_n\}$ est un ensemble fini de sommets et A $\subset$ N $\times$ N un ensemble fini de |A| arcs de la forme (x_i,x_j). À chaque arc a $\in$ A est associé un vecteur de fonction de coûts à valeurs dans $\mathbb{N}$ de la forme c(a) = (c_1(a), . . . , c_q (a)) où q est le nombre de valuations du graphe, et c_i la fonction qui attribue à un arc du graphe sa i^{eme} valuation. s $\in$ N est le sommet source du graphe (départ), et p $\in$ N est le sommet puits du graphe (arrivée). Une solution est un chemin partant du sommet source et finissant par le sommet puits.

On s'intéresse aux chemins optimaux entre deux sommets du graphe. Soit CH_{ij} l'ensemble des chemins de x_i vers x_j dans le graphe G. On définit un vecteur de coût C qui à un chemin ch_{ij} associe l'évaluation des coûts pour parcourir ce chemin. On cherche à identifier le (ou les) chemin(s) $ch_{ij} \in CH_{ij}$ tels qu'il n'existe pas de chemin ch'_{ij} $\in CH_{ij}$ vérifiant C(ch'_{ij}) $<$ C(ch_{ij}). L'opérateur $<$ compare des vecteurs, ce qui implique un ordre partiel sur le coût des chemins.

Nous n'indiquons pas ici comment sont déterminés précisément les coûts dans le contexte des trajets maritimes. À chaque sommet est associé un lieu géographique et le coût de chaque arc est calculé pour un trajet en ligne droite entre deux lieux de passage en fonctions des courants marins et des conditions météorologiques à la date où le trajet a lieu.

2.2 Les algorithmes de recherche de chemins optimaux

Il existe de nombreux algorithmes de recherche de chemins optimaux dans un graphe valué, qui se déclinent notamment selon s'ils sont mono ou multi-objectifs et si le(s) coût(s) des arêtes évoluent au cours du temps.

Tous ces algorithmes supposent que le trajet se fait à vitesse constante. La prise en compte d'un changement de vitesse ou de puissance du moteur au cours du trajet inclut une complexité encore plus grande qui est rarement pris en compte en pratique.

2.2.1 Algorithmes mono-objectifs

L'optimisation mono-objectif consiste à n'optimiser qu'un seul critère et permet d'obtenir une seule solution optimale. L'algorithme le plus connu est celui de Dijkstra [Dijkstra, 1959] qui utilise une approche gloutonne pour obtenir une solution optimale en temps $O(n \log n)$, où n est le nombre de sommets du graphe.

L'algorithme A* [Hart et al., 1968] est une variante de Dijkstra qui a été définie initialement pour traiter le cas où le graphe (trop grand voire infini) est défini en compréhension. Il évalue un chemin en construction non seulement en prenant en compte le coût du chemin déjà parcouru (comme Dijkstra) mais aussi une sous-évaluation heuristique du coût du chemin restant à parcourir.

En pratique A* permet de sélectionner plus rapidement le meilleur chemin et de diminuer notablement le temps de calcul pour obtenir le chemin le moins coûteux.

2.2.2 Algorithmes multi-objectifs

Les algorithmes multi-objectifs doivent optimiser plusieurs objectifs dont aucun n'est dominant.

Dans le cas qui nous occupe, nous pouvons avoir à prendre en compte le temps de trajet, le coût du carburant, l'impact environnemental, l'usure du navire, le confort des voyageurs (minimiser les changements de cap et de vitesse), etc. Dès lors que l'on n'a plus un seul objectif, on n'a plus unicité de la solution optimale (plus précisément : unicité du coût optimal, car à un coût peut correspondre plusieurs solutions équivalentes) mais un nombre potentiellement exponentiel de vecteurs de coûts dont aucun ne domine l'autre, qu'on appelle *Front de Pareto*.

Front de Pareto Lorsqu'on a plusieurs critères, on ne peut dire qu'une solution est meilleure qu'une autre que si elle l'est sur tous les critères. On dit alors qu'elle la *domine*. Certaines solutions sont incomparables : une solution peut être meilleure sur un critère et moins bonne sur un autre.

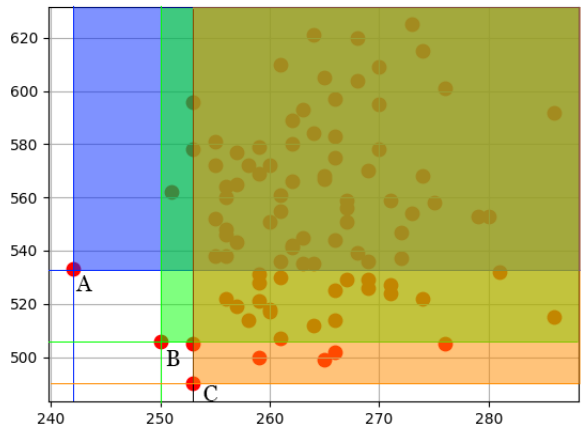

FIGURE 3 – Ensemble de solutions. On a en bleu, vert et orange les zones dominées par les sommets A, B et C. Ces sommets constituent un front de Pareto.

L'ensemble des solutions dominées par aucune autre solution s'appelle le *front de Pareto*. Une solution *pareto-optimale* a un vecteur de coût tel qu'il n'existe pas une solution alternative pour laquelle un élément du vecteur de coût serait meilleur.
Contrairement à sa version monocritère, un algorithme multicritère doit gérer plusieurs solutions optimales, potentiellement un nombre exponentiel. Un problème multi-objectif présente la difficulté qu'un algorithme le résolvant ne doit pas prendre en compte uniquement le coût de la meilleure solution pour éliminer d'autres candidats potentiels mais tous les vecteurs de coût du front de Pareto en cours de constitution. Le problème devient ainsi NP-difficile et donc potentiellement de complexité exponentielle.
MOA* [Stewart and White, 1991] (Multi-Objective A*) et NAMOA* (New Approach to MOA*) [Mandow and De La Cruz, 2008] sont des variantes généralistes de A* prenant en compte le multi-objectif.

Scalarisation La scalarisation consiste à combiner linéairement différents critères pour en former un seul. Si on a trois critères c_1, c_2 et c_3, la scalarisation consiste à définir la fonction C(c_1,c_2, c_3) = $\alpha_1.c_1 + \alpha_2.c_2 + \alpha_3.c_3$, où les α_i sont des coefficients à fixer qui déterminent l'importance relative de chaque coût c_i.
Cela permet de trouver une solution appartenant au front de Pareto (voir section 2.2.2). En choisissant adéquatement les coefficients (par recherche dichotomique), on peut obtenir toute l'enveloppe convexe d'un front de Pareto en un temps polynomial. Il nous manquera juste les solutions qui sont dans les "cavités" de l'enveloppe convexe.
La scalarisation a l'immense avantage de permettre une résolution rapide mais, comme elle ne trouve que les solutions de l'enveloppe convexe du front de Pareto, il faut que celle-ci contienne un "bon" sous-ensemble de solutions par rapport au contexte.
Dans la suite de l'article, nous appellerons *Dijkstra scalaire*, un algorithme de Dijkstra effectuant du multi-objectif en scalarisant les coûts.

Algorithmes dépendant du temps Lorsque le coût des arêtes d'un graphe change au cours du temps (par exemple à cause de la météo), la propriété qu'un sous-chemin d'un chemin optimal est optimal devient fausse (cf figure 4). On ne peut plus s'en servir pour éliminer des chemins partiels sous-optimaux. Il faut utiliser d'autres critères plus faibles.

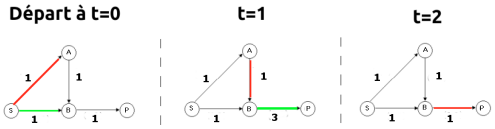

FIGURE 4 – Dans cet exemple de trajet temporel (t :0,1,2) toutes les arêtes ont un coût de 1 sauf B vers P au temps 1 qui coûte 3. Cela permet de mettre en avant que dans un cadre temporel même si au sommet B le trajet partiel vert domine le rouge le coût final du trajet rouge est meilleur.

Dans le contexte du routage maritime, nous pouvons citer l'algorithme de Venetti [Veneti, 2015] et NAMOA*-TD [Chauveau, 2018]. Pour permettre l'élimination de chemins partiels candidats, ces algorithmes ont deux critères :

- Le coût est supérieur à celui d'une solution déjà trouvée.
- Le coût est supérieur à celui d'un chemin partiel arrivant au même sommet à la même date.

NAMOA*-TD, une version "time-dependent" de NAMOA* s'avère beaucoup plus efficace en pratique que l'algorithme de Venetti. Ceci est notamment dû au fait que, contrairement à Venetti, NAMOA*-TD utilise une heuristique d'estimation du coût du chemin restant à parcourir. Cette heuristique s'avère être une évaluation suffisamment proche du réel pour détecter tôt les chemins candidats qui seront dominés.
Bien qu'apparaissant comme une méthode efficace dans le cadre du routage maritime international en permettant le calcul de trajets réalistes en moins de deux minutes, NAMOA*-TD a pourtant des limites qui l'empêchent d'être pleinement satisfaisant dans le contexte qui nous occupe. Le maillage rectangulaire et régulier qui détermine la forme des trajectoires n'est pas assez précis lors de la navigation proche des côtes ou dans des zones maritimes étroites (par exemple, la Manche) : la zone maritime peut être plus étroite que les mailles. La solution consistant à réduire la taille des mailles mène à une forte augmentation du temps de calcul. Par exemple, diviser la taille des mailles (rectangulaires) selon ses deux dimensions multiplie par quatre le nombre de sommets du graphe dans un contexte où la complexité temporelle des algorithmes est exponentielle en ce

nombre de sommets.
En pratique, l'outil utilisé par l'armateur que nous avons consulté diminue les tailles des mailles dans certaines zones où il était constaté a posteriori que c'était nécessaire (cf figure 5). L'inconvénient est que le graphe doit se construire

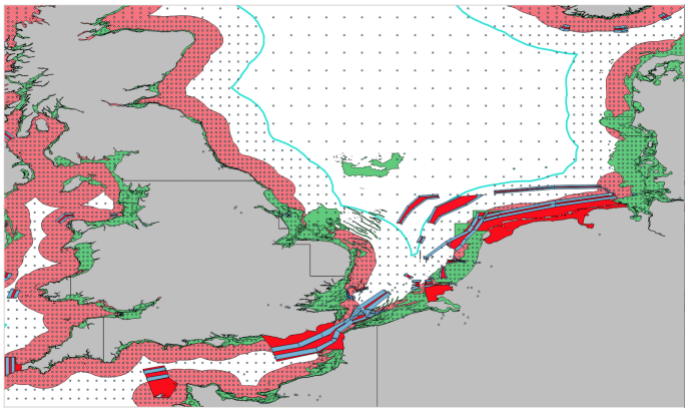

FIGURE 5 – Exemple de variation du maillage pour la Manche.

en quelque sorte "à la main" et qu'il est difficile de juger précisément de la taille des mailles en fonction des lieux. C'est pourquoi il existe d'autres méthodes qui génèrent le graphe en permettant aux lieux de passage associés aux sommets d'être localisés n'importe où dans un espace continu.

2.3 Recherche d'un chemin dans un espace continu

La recherche de trajectoire dans un espace continu est une problématique que l'on retrouve typiquement en robotique mobile. Les algorithmes de planification de trajectoire les plus utilisés prennent en compte un espace grand, complexe ou de haute dimension (ex : bras robotisés, drones, voitures autonomes, etc).
Comme notre objectif est de déterminer la trajectoire d'un navire sur l'océan, nous en faisons une présentation uniquement dans le cadre particulier d'un espace continu en 2D.
L'objectif de ces algorithmes est de trouver un chemin reliant une position de départ q_{start} à une position d'arrivée q_{goal}, sans heurter d'obstacles.

2.3.1 L'approche Probabilistic RoadMaps (PRM)

En substance, PRM [Kavraki et al., 1996] tire aléatoirement et uniformément des lieux dans l'espace 2D et essaie progressivement de les relier entre eux pour former un seul arbre. À chaque fois qu'un nouveau lieu est ajouté, on essaie de le relier au lieu le plus proche d'une composante connexe (un arbre) s'il se trouve à une distance inférieure à d (paramètre à fixer). Ainsi, progressivement, on relie entre eux les arbres du graphe en construction, en espérant que G sera connexe au bout de N itérations.
Une fois le graphe obtenu, il ne reste plus qu'à appliquer un algorithme de recherche de chemin dans un graphe.

2.3.2 L'approche Rapidly-exploring Random Trees (RRT)

RRT est une alternative à PRM, qui construit progressivement un arbre de lieux atteignables, en se développant rapidement vers les zones encore inexplorées.
Voici l'algorithme :

Algorithm 1 Algorithme RRT (Rapidly-Exploring Random Tree)

Initialisation : l'arbre T contient uniquement la racine q_{start}.
while le nombre d'itérations $< N_{max}$ **do**
  $q_{rand} \leftarrow$ point aléatoire dans l'espace 2D
  $q_{near} \leftarrow$ nœud de T le plus proche de q_{rand}
  $q_{new} \leftarrow$ point obtenu en avançant depuis q_{near} vers q_{rand} d'une distance maximale δ
  if le segment $[q_{near}, q_{new}]$ ne traverse aucun obstacle **then**
    ajouter q_{new} à l'arbre T
  end if
  if q_{new} est suffisamment proche de la cible **then**
    terminer
  end if
end while

Quand le nombre d'itérations tend vers l'infini, la probabilité de produire un chemin menant du lieu de départ au lieu cible tend vers 1. Cela s'explique par le fait que q_{rand} est tiré aléatoirement et uniformément sur tout l'espace et donc l'arbre s'accroît progressivement vers tous les lieux possibles de l'espace.
Au final, on obtient un arbre qui relient des lieux géographiques couvrant assez uniformément la totalité de l'espace 2D, en évitant les obstacles.

2.3.3 RRT*

RRT* est une variante de RRT qui garantit l'optimalité asymptotique de la longueur du trajet.
L'optimalité asymptotique signifie que la probabilité que le coût de la solution renvoyée converge vers la solution optimale (le coût minimal) s'approche de 1 lorsque le nombre d'échantillons approche l'infini.
Voici les éléments clés qui confèrent l'optimalité asymptotique à l'algorithme RRT :

- Le mécanisme d'optimisation (rewiring)
  Contrairement à RRT, qui ne garantit que l'exhaustivité probabiliste (la certitude de trouver un chemin si un chemin existe), RRT intègre un mécanisme d'optimisation en deux étapes pour affiner continuellement le chemin trouvé.
  RRT* étend RRT en introduisant un processus de recâblage de l'arbre (rewiring) pendant sa construction, permettant à l'algorithme de converger presque sûrement vers le chemin optimal pour chaque état du domaine de recherche.
  Les deux étapes d'optimisation clés sont :
  - Sélection du parent à coût minimal : lorsqu'un nouveau nœud (x_{new}) est ajouté, l'algorithme

ne le connecte pas simplement à son voisin le plus proche ($x_{nearest}$), mais il cherche le parent (x_{min}) parmi tous les nœuds voisins (X_{near}) qui permet de minimiser le coût total du chemin à partir de l'état initial jusqu'à x_{new}.

— Recâblage des voisins : l'algorithme vérifie si le nouveau nœud x_{new} peut offrir un chemin de coût inférieur à ses propres voisins. Si un voisin peut atteindre la racine via x_{new} à un coût inférieur à son chemin actuel, la connexion du voisin est mise à jour pour passer par x_{new}, et l'ancienne arête est supprimée pour maintenir la structure arborescente. Ce processus garantit que les sommets sont atteints par un chemin de coût minimal.

— Une stratégie d'échantillonnage efficace
Le succès de l'optimalité asymptotique repose sur une condition mathématique rigoureuse relative à la densité des connexions et au nombre d'échantillons (n). Pour garantir à la fois l'optimalité asymptotique et l'efficacité de la recherche, le rayon de connexion (r) ne doit pas être fixe (comme dans le RRT traditionnel) mais doit évoluer en fonction du nombre d'itérations n.

2.3.4 L'approche de Weather Routing Metaheuristic

Weather Routing Metaheuristic (WRM) [Grandcolas, 2022] constitue une approche originale du routage de navire qui ne fixe pas non plus a priori les lieux de passages et permet de générer des trajets dont les lieux de passage peuvent se trouver n'importe où sur la surface de navigation à un degré de précision aussi élevé que l'on souhaite.

Le graphe se détermine en générant aléatoirement n lieux dans une zone donnée et en créant une arête entre deux lieux si la distance les séparant est inférieure à une constante d donnée. C'est un peu comme le font PRM ou RRT, sauf que ces derniers ne rajoutent incrémentalement qu'un sommet à chaque fois pour maintenir une structure d'arbre, tandis que WRM crée une arête pour chaque sommet de distance inférieure à d, pour former un graphe. Le fait qu'on puisse choisir le nombre n à l'unité près permet de choisir précisément la taille de l'instance et donc le temps d'exécution de la méthode. Les coordonnées des lieux sont déterminés avec la précision que l'ont veut.

L'algorithme procède par itérations en cherchant un trajet dans le graphe, puis en sélectionnant la zone géographique proche des lieux du trajet trouvé, puis en retirant aléatoirement n lieux dans cette zone restreinte pour former un nouveau graphe. Au fur et à mesure, la zone de proximité se réduit de plus en plus, ce qui affine la trajectoire progressivement (cf figure 6). Dans WRM, c'est un algorithme incomplet mono-objectif minimisant la consommation de carburant qui est utilisé pour déterminer un nouveau trajet à chaque itération mais rien n'empêche l'utilisation d'une autre procédure mono ou multi-objectif, complète ou incomplète. Si la procédure est multi-objectif, elle génère plusieurs chemins à chaque itération et il faut en choisir un seul pour l'itération suivante.

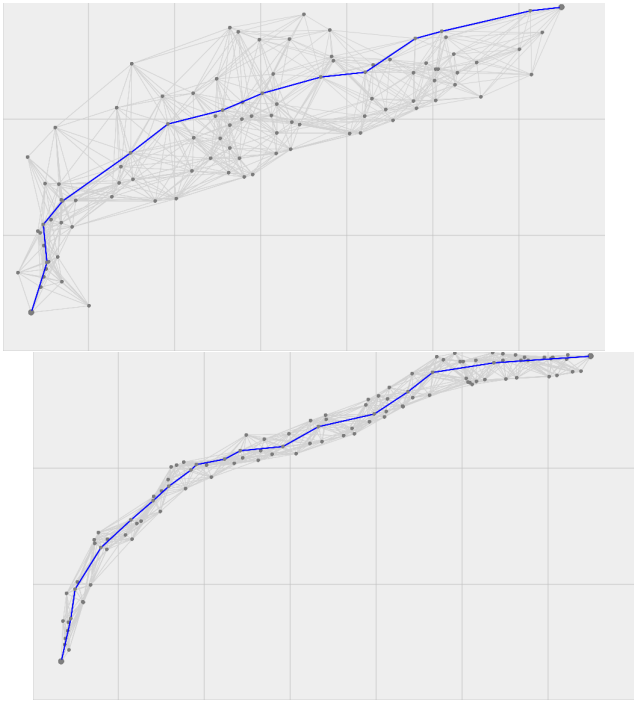

FIGURE 6 – Étape k (haut) et étape k + 1 (bas). La zone géographique des lieux possibles de l'étape $k+1$ est déterminée par la trajectoire trouvée à l'étape k.

Avec WRM, chaque trajet généré par une itération ne sert qu'à déterminer une zone dans laquelle le trajet devra se situer. Or, rien ne garantit que le prochain trajet trouvé dans le nouveau graphe dont les lieux des sommets sont choisis aléatoirement sera meilleur que le précédent. C'est pourquoi nous avons opté pour une méthode destinée à améliorer le trajet courant par réparation locale.

3 Réparation locale de trajet

Nous présentons ici l'idée fondamentale de notre approche. Nous supposons qu'un trajet a déjà été calculé, quelle que soit la méthode utilisée. Nous cherchons à améliorer le coût du trajet grâce à une méthode de réparation locale très rapide.

3.1 Déplacement des lieux de passage

L'idée très simple est de choisir un sommet et de modifier sa localisation géographique afin que le trajet soit globalement amélioré. La nouvelle localisation est choisie n'importe où à l'intérieur d'un disque qui contient le lieu actuel du sommet qu'on veut relocaliser. Le rayon du disque est la distance entre le sommet et le plus proche de ses deux voisins sur le trajet. La réparation est locale car on a uniquement modifié le lieu géographique d'un seul sommet ainsi que les arcs reliant le sommet à son antécédent et son successeur. Il y a plusieurs manières de sélectionner un sommet à modifier et un endroit où le relocaliser. La manière la plus courante utilisée dans les méthodes de réparation locale est de faire des choix aléatoires et de vérifier s'ils ont amélioré la solution. Voici le schéma d'algorithme que nous utilisons

et que nous appelons *Route-repair* :

Algorithm 2 Route-repair

Require: Trajet initial T, nombre d'itérations maximum N, fonction de coût f
Ensure: Trajet optimisé
1: $i \leftarrow 0$
2: **while** $i <$ N **do**
3: Sélectionner aléatoirement un sommet s dans T
4: Générer un nouveau point s' dans un disque centré sur s
5: $T' \leftarrow$ Trajet T avec s remplacé par s'
6: **if** $f(T') < f(T)$ **then**
7: $T \leftarrow T'$
8: **end if**
9: i $\leftarrow$ i $+1$
10: **end while**
11: **return** T

Le fait que le nouveau sommet puisse se trouver absolument n'importe où avec une précision arbitraire à l'intérieur du disque fait que tous les sommets du trajet final peuvent être plus finement placés. De même, les angles entre les transitions entre deux lieux ne sont plus limités à des valeurs fixes. Ainsi, l'effet de crénelage peut être très fortement réduit, ce qui diminue la longueur du trajet calculé.
Ce type de réparation a l'avantage d'être simple, rapide et de permettre souvent une amélioration non négligeable de la solution. Elle peut être utilisée en post-traitement de n'importe quel algorithme de recherche de trajet. Cependant, l'amélioration obtenue est forcément limitée : si le trajet de départ est éloignée géographiquement du trajet optimal, celui-ci ne sera probablement jamais atteint ou alors très progressivement en un temps très long. En effet, un sommet ne peut pas être déplacé loin de son lieu originel sinon cela accroît exagérément la distance totale parcourue et ne peut pas améliorer la solution. Si tous les sommets du trajet de départ sont éloignés des sommets du trajet optimal, la seule manière de les rapprocher tous de leur destination finale est de les bouger tous progressivement de façon à ce qu'ils restent proches les uns des autres.
Par rapport à WRM, qui lui aussi tire aléatoirement des lieux autour de ceux du trajet précédent, Route-repair génère les nouveaux lieux un par un et ne les intègre que quand ils améliorent le trajet. WRM génère aléatoirement tous les nouveaux lieux au début et trouve ensuite le meilleur trajet parmi eux.
En outre, il faut remarquer que si on veut que notre trajet approxime au mieux un trajet continu, nous ne devrions pas fixer a priori le nombre de lieux de passage mais permettre à notre algorithme d'en déterminer le nombre utile. Il faut aussi faire en sorte que la distance entre deux lieux du trajet ne soit pas identique comme cela l'est artificiellement dans un maillage, où il peut par exemple exister une succession de lieux de passage en ligne droite alors qu'il suffirait de relier directement les extrémités de cette ligne droite. Il nous faut un moyen pour permettre à la distance entre deux lieux de passage d'être aussi variable que l'ont veut, ce qui n'est pas réalisable en pratique quand on répare un trajet issu d'un maillage régulier.
Nous allons donc voir comment modifier le nombre de lieux de passage du trajet et leur distance.

3.2 Changement du nombre de lieux de passage

Pour ajuster le nombre de lieux grâce à une méthode de réparation, il n'est nécessaire que de prévoir comment ajouter un lieu et comment en supprimer un.
Pour ajouter un lieu, nous proposons de choisir une arête du trajet au hasard, de choisir aléatoirement un lieu dans le disque dont l'arête est le diamètre, puis de l'intégrer effectivement au trajet si le coût du trajet devient meilleur. Insérer un lieu de passage peut permettre de lisser un trajet quand on doit changer drastiquement de direction, par exemple quand on emprunte un canal sinueux.
Pour supprimer un lieu, nous proposons de vérifier systématiquement si l'angle entre deux arêtes consécutives du trajet est proche de 180°. Si c'est le cas, il est clair que le sommet intermédiaire peut être supprimé sans que le coût du chemin ne soit sensiblement altéré.
Afin que notre trajet ne finisse pas par contenir trop de sommets, il faut en ajouter avec parcimonie et en supprimer autant que l'on peut. Notre méthode de réparation peut donc être complétée comme indiqué par l'algorithme 2.

3.3 Expérimentations

Nous voulons savoir expérimentalement si notre méthode de réparation peut améliorer sensiblement une solution déjà produite par un autre algorithme. Le cas le plus intéressant est celui où un algorithme complet permet d'obtenir toutes les solutions Pareto-optimales pour un graphe donné, ce qui est le cas de NAMOA*-TD. Notre méthode de réparation locale ne peut améliorer les solutions que parce qu'on change les lieux de passage.
Nous voulons aussi expérimenter l'impact de chaque aspect de la réparation : déplacement, ajout et suppression de lieu. C'est pourquoi nous avons testé trois types de réparations : celle où on ne s'autorise qu'à déplacer des lieux, celle où on s'autorise en plus à en supprimer et celle où on s'autorise à la fois des suppressions et des ajouts.
Les données des courants et météo sont issus de fichiers GRIB correspondant à des données réelles. Les algorithmes ont été testé sur une plage météo de 10 jours en prenant plusieurs trajets différents commençant à différents horaires. Nous avons utilisé un ordinateur 64 bits cadencé à 2,1 Ghz et ayant 192 Go de RAM.
Nous n'avons pour l'instant considéré que deux critères : le temps et le carburant consommé. Le modèle de consommation nous a été fourni par un armateur et ne constitue qu'une approximation pour un seul type de ses navires.
La Figure 8 présente l'ensemble des solutions Pareto-optimales obtenues pour une instance d'exemple (trajet de Salvador, Brésil, à Luanda, Angola), ainsi que les versions réparées de ces routes. Le diamètre du disque à l'intérieur duquel un point de passage est repositionné est égal à la distance entre ses deux points voisins. Le nombre d'itérations

Algorithm 3 Route-repair (avec ajout et suppression de sommets

Require: Trajet initial T, nombre d'itérations max N, seuil sans amélioration k, fonction de coût f
Ensure: Trajet optimisé T

$i \leftarrow 0$
$n_{\text{sans\_amelioration}} \leftarrow 0$
while $i < N$ **do**
  $T_{\text{courant}} \leftarrow T$
  for all sommet s_i de T sauf les extrémités **do**
    if s_i est approximativement aligné avec ses deux voisins **then**
      Supprimer s_i de T
    end if
  end for
  $n_{\text{sans\_amelioration}} \leftarrow n_{\text{sans\_amelioration}} + 1$
  if $n_{\text{sans\_amelioration}} \geq k$ **then**
    Sélectionner une arête (s_i, s_{i+1}) aléatoirement dans T
    Choisir aléatoirement un lieu s dans le disque dont (s_i, s_{i+1}) est le diamètre
    Insérer s dans le trajet entre s_i et s_{i+1} pour obtenir T'
    if $f(T') < f(T)$ **then**
      $T \leftarrow T'$
      $n_{\text{sans\_amelioration}} \leftarrow 0$
    end if
  else
    Sélectionner un sommet s aléatoirement
    Générer un point s' autour de s (dans un disque)
    Construire T' en remplaçant s par s'
    if $f(T') < f(T)$ **then**
      $T \leftarrow T'$
      $n_{\text{sans\_amelioration}} \leftarrow 0$
    end if
  end if
  $i \leftarrow i + 1$
end while
return T

NAMOA*-TD (1897s)	Après Route-repair (+16 s)	Économie
5864	5648	216
5819	5651	168
5772	5649	123
5740	5653	87
5740	5668	72
5739	5647	92
5734	5661	73
5734	5658	76
5733	5639	94

TABLE 1 – Consommation de carburant (tonnes) : Namoa*-TD vs Namoa*-TD + Route-repair

(N) est fixé à 2000. Le Tableau 1 montre que la méthode Route-repair peut générer des routes améliorées permettant d'économiser environ 1% de carburant, avec un temps de calcul total de 16 secondes (soit 2 secondes par route), tandis que les routes originales ont été calculées en 30 minutes.

Nous constatons qu'il est rapide d'obtenir des trajets plus précis, plus rapides et permettant des réductions de consommation de carburant non négligeables.

Le fait de pouvoir supprimer des lieux constitue un gain net par rapport au seul déplacement des lieux. Le fait de pouvoir ajouter des lieux n'a pas permis d'améliorer sensiblement les solutions sur les instances que nous avons testées. En effet, celles dont nous disposions consistent à relier des ports qui ne sont pas séparés par des zones terrestres ayant besoin d'être contournées.

L'intérêt d'une méthode de réparation locale est de générer des solutions beaucoup plus rapidement qu'une méthode complète tout en garantissant d'obtenir des solutions suffisamment bonnes. Dans un contexte où on a besoin d'obtenir des solutions dans un temps arbitrairement limité, nous ne pouvons pas nous permettre d'utiliser un algorithme complet comme NAMOA*-TD. Dans la suite de l'article, nous étudions la possibilité d'obtenir des trajets encore plus rapidement en utilisant des méthodes de calcul de trajets initiaux très rapides destinés à être rapidement réparés par notre méthode.

4 Calculs rapides de trajets

Pour que le calcul d'un trajet initial (à réparer) soit rapide, il faut que l'algorithme utilisé soit de complexité polynomiale. Nous choisissons l'algorithme de Dijkstra scalarisé. Pour la localisation des sommets du graphe, nous expérimentons quatre méthodes :

— La localisation le long d'une grille régulière, les sommets ayant huit voisins (cf à gauche de la figure 1). Cette méthode est rapide, la répartition des lieux est homogène, mais le nombre de sommets du graphe ne peut pas être fixé comme on veut.
— Les lieux sont choisis aléatoirement. On choisit le nombre de sommets que l'on veut. Les sommets sont reliés si leur lieux sont en deça d'une distance fixée. La méthode est très rapide mais il y a un risque de clusters de points. Si la distance fixée est trop petite, le graphe risque de ne pas être connexe. Si la distance fixée est trop grande, les sommets ont trop de voisins et la combinatoire de recherche du meilleur trajet devient trop élevée.
— Les lieux sont choisis aléatoirement. Les sommets sont reliés entre eux grâce à une triangulation de Delaunay [Delaunay, 1934], ce qui garantit la connexité du graphe tout en limitant fortement le nombre d'arêtes. Le risque de clusters de lieux perdure.
— Les lieux sont choisis aléatoirement selon la technique du Poisson Disk Sampling [Bridson, 2007]. Contrairement à un tirage aléatoire pur (qui peut donner des amas de points ou des trous), le Pois-

NAMOA*-TD	Dijkstra scalaire			
meilleure route	Grille	Aléatoire	Delaunay	Poisson
5733	5732	5624	5928	5818
1897s	0.61s	23.68s	1.37s	0.17s
Après Route-repair				
5639	5661	5619	5835	5694
+16s	+1.77s	+2.11s	+2.39s	+1.59s

TABLE 2 – Consommation de carburant et temps de calcul : NAMOA*-TD vs Dijkstra scalaire, avant et après Route-repair.

son Disk Sampling garantit que deux points ne peuvent être à moins d'une distance minimale l'un de l'autre, ce qui produit une répartition plus homogène, tout en conservant un aspect aléatoire. Les temps de calcul sont plus longs.

4.1 Expérimentations

Les résultats de test suivants sont basés sur la même instance d'une route allant de Salvador, Brésil, à Luanda, Angola. D'autres expériences menées pour différentes dates et différentes paires de ports ont produit des résultats qualitativement similaires.

Tous les tests ont été réalisés en utilisant la même graine aléatoire afin d'assurer la reproductibilité et une comparaison équitable entre les configurations des algorithmes. Les distances sont exprimées en degrés. En raison de la nature stochastique des algorithmes, nous avons effectué cinq exécutions indépendantes et rapporté les valeurs moyennes à la fois du coût et du temps d'exécution (voir la table 2 et la figure 9).

Concernant le temps de calcul de la route initiale, par construction, un graphe généré aléatoirement contient significativement plus d'arêtes qu'une grille, tandis que les méthodes de Delaunay et de Poisson Disk Sampling produisent des graphes avec beaucoup moins d'arêtes. De plus, le Poisson Disk Sampling génère moins de sommets que les autres méthodes (voir Figure 7). Par conséquent, puisque le temps d'exécution de Dijkstra scalaire dépend fortement du nombre d'arêtes, le temps nécessaire pour calculer la solution initiale à partir d'un graphe aléatoire est considérablement plus long que pour les trois autres méthodes. Cependant, après l'application de Route-repair, les routes dérivées de la grille, du graphe aléatoire et du Poisson Disk Sampling sont de qualité comparable à celles obtenues en utilisant NAMOA*-TD. En revanche, les routes dérivées des graphes de Delaunay sont clairement de qualité inférieure, en raison de la faible qualité des routes initiales.

Nous observons que les meilleures approches — celles qui produisent des solutions de haute qualité en un temps court, comparables à celles obtenues avec NAMOA*-TD — sont basées sur l'algorithme de Dijkstra scalaire appliqué à une grille ou à un graphe généré par échantillonnage de Poisson Disk. L'approche fondée sur un graphe aléatoire est plus lente, tandis que celle basée sur un graphe de Delaunay produit des solutions de basse qualité.

4.2 Lissage de trajet par courbe de Bézier

La réparation de trajet permet aussi d'obtenir des trajectoires plus lisses. Or, il existe une technique de lissage de courbe, dite *courbes de Bézier* à laquelle il nous faut nous comparer. Dans le cadre de trajets maritimes, l'algorithme Bai-RRT* [Tie Xu, 2025] détermine une combinaison de courbes de Bézier simples et cubiques pour rendre les trajectoires moins anguleuses, tout en garantissant de passer par les lieux de passage (les points de contrôle de la courbe de Bézier). Or, bien que ce traitement permette d'obtenir des trajectoires plus lisses, cela n'aide pas à en optimiser le coût (le coût calculé peut même augmenter) car les points de contrôle ne bougent pas. Or, c'est bien cette possibilité de bouger les lieux de passage qui permet de réduire les coûts.

5 Conclusion et Perspectives

Nous avons proposé une méthode rapide pour générer des routes maritimes précises dans un contexte dynamique de conditions météorologiques et de courants dépendant du temps.

Notre méthode de réparation Route-repair permet de repositionner avec précision les points de passage qui ne sont pas idéalement placés, de supprimer les points de passage inutiles (lorsqu'ils sont alignés) et d'en ajouter de nouveaux afin d'adoucir les changements de direction.

Route-repair peut servir d'étape de post-traitement pour tout algorithme déjà implémenté dans un contexte industriel. Il est simple à programmer et permet souvent d'apporter une amélioration non négligeable aux solutions en un temps arbitrairement court.

Il peut également réparer des routes sous-optimales générées très rapidement, bien que les routes finales obtenues ne soient assorties d'aucune garantie de proximité avec l'optimalité — même si cela se produit parfois.

La simplicité de Route-repair ouvre la voie à de nombreuses améliorations potentielles. Fondamentalement, elle se comporte comme une descente de gradient pouvant converger vers un optimum local. Toutefois, de nombreux algorithmes existent pour échapper aux optima locaux et pourraient être adaptés à notre problème. Une autre limite actuelle est que nous ne gérons pas les portions d'itinéraire obligatoires du type canaux ou chenaux, où les points de passage ne sont pas modifiables, ni la présence d'obstacles qui empêchent certains points d'être librement déplaçables et certaines arêtes d'exister.

Références

[Bridson, 2007] Bridson, R. (2007). Fast poisson disk sampling in arbitrary dimensions. *SIGGRAPH '07 : ACM SIGGRAPH 2007 sketches*.

[Chauveau, 2018] Chauveau, E. (2018). Optimisation des routes maritimes : un système de résolution multicritère et dépendant du temps.

[Chauveau et al., 2017] Chauveau, E., Jégou, P., and Prcovic, N. (2017). Weather routing optimization : A new shortest path al-

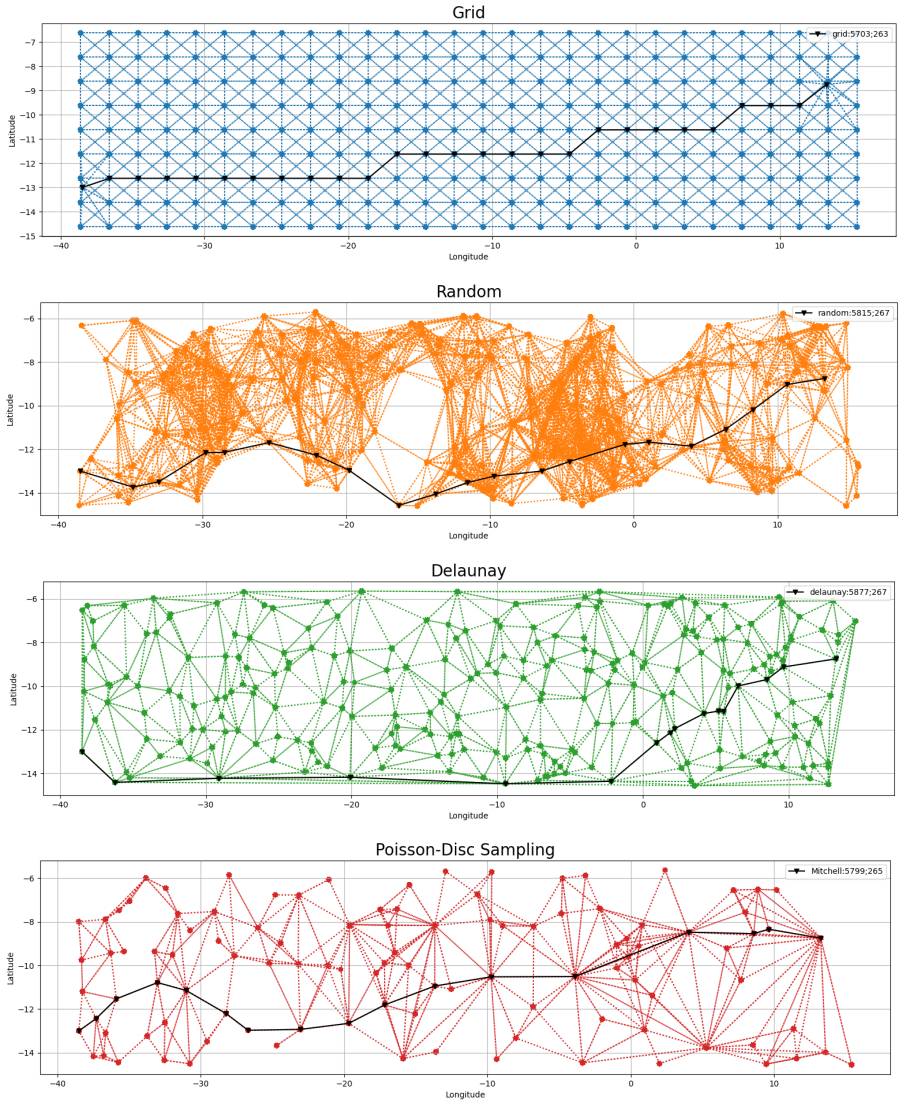

FIGURE 7 – Les quatre types de graphes générés (grille, aléatoire, Delaunay et Poisson Disk Sampling) avec un exemple de route.

gorithm. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pages 687–694. IEEE Computer Society.

[Delaunay, 1934] Delaunay, B. (1934). Sur la sphère vide. *Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et na*, 1934(6) :793–800.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1) :269–271.

[Grandcolas, 2022] Grandcolas, S. (2022). A metaheuristic algorithm for ship weather routing.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2) :100–107.

[Kavraki et al., 1996] Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4) :566–580.

[Mandow and De La Cruz, 2008] Mandow, L. and De La Cruz, J. L. P. (2008). Multiobjective a* search with consistent heuristics. *J. ACM, Vol.57(5)*.

[Stewart and White, 1991] Stewart, B. S. and White, III, C. C. (1991). Multiobjective a*. *J. ACM, Vol.38(4) :p.775-814.*

[Tie Xu, 2025] Tie Xu, Jun Ma, P. Q. Q. H. (2025). An improved rrt* algorithm based on adaptive informed sample strategy for coastal ship path planning.

[Veneti, 2015] Veneti, A., K. C. e. P. G. (2015). Continuous and discrete time label setting algorithms for the time dependent bicriteria shortest path problem. *Computing Society Conference, p.62-73. 8.*

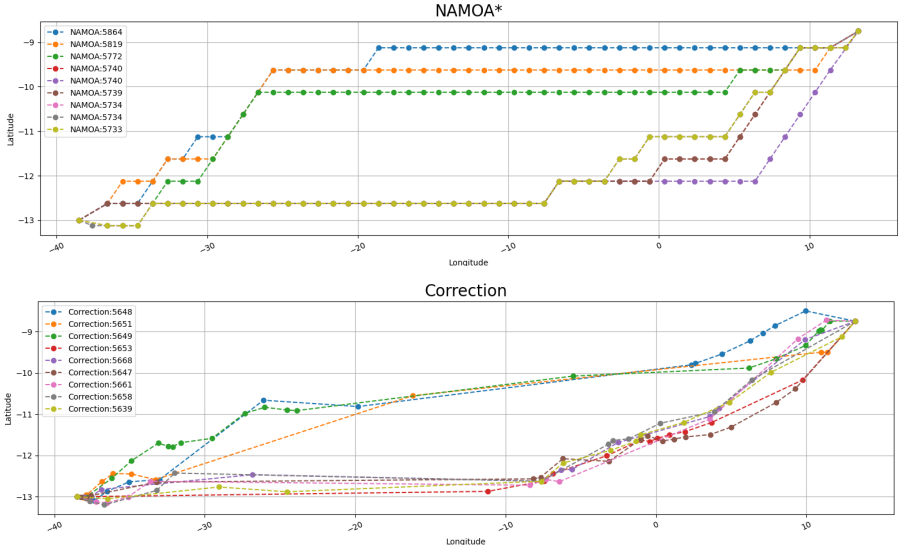

FIGURE 8 – Exemple de solutions Pareto-optimales obtenues en utilisant NAMOA*-TD (en haut) et leur réparation de route (en bas).

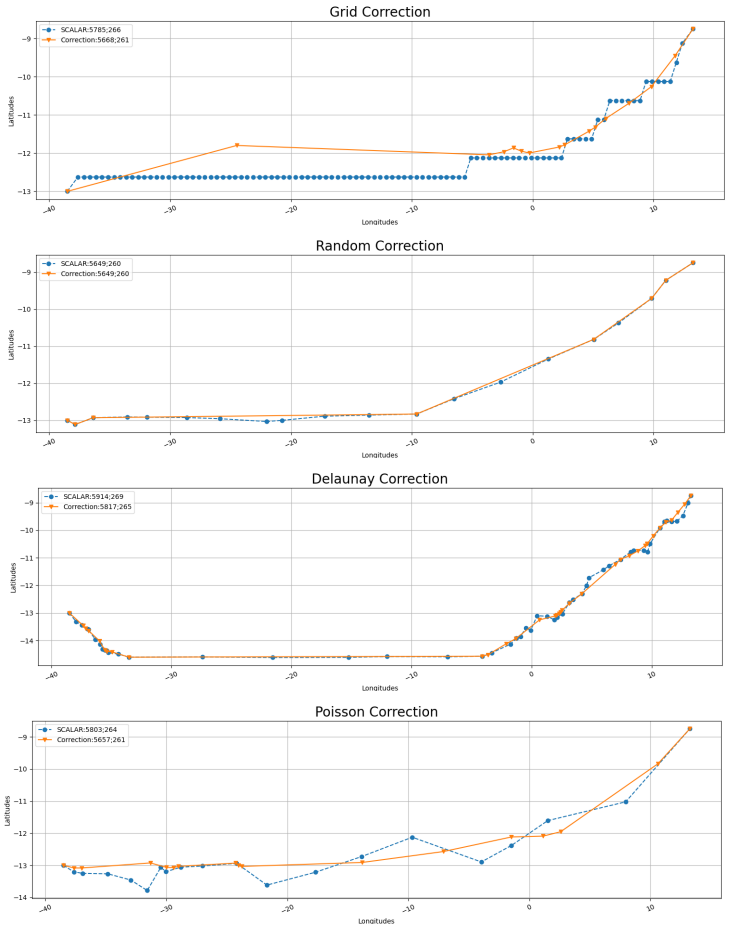

FIGURE 9 – Les routes réparées obtenues à partir des routes initiales de quatre types de graphes (grille, aléatoire, Delaunay et Poisson Disk Sampling).