VECTORTSP : *un problème de voyageur de commerce avec contraintes d'accélération* ‡

Arnaud Casteigts[1] et Mathieu Raffinot[1] et Jason Schoeters[1]

[1] *Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France*

Nous nous intéressons à une nouvelle version du TSP Euclidien, appelée VECTORTSP (ou VTSP) dans laquelle un véhicule peut se déplacer par rapport à un ensemble de contraintes physiques inspirées du jeu de papier et crayon RACETRACK (aussi appelé VECTOR RACER). Comparée à d'autres versions du TSP avec contraintes physiques, comme le DUBINS TSP, l'esprit de ce modèle est que (1) la vitesse n'est pas bornée, et (2) l'inertie dépend de la vitesse. Dans cet article, nous montrons plusieurs faits intéressants, comme le fait qu'un ordre de visite optimal pour le problème ETSP peut être sous-optimal pour VTSP. De même, une trajectoire optimale pour VTSP peut comporter des croisements. Par ailleurs, nous démontrons que VTSP est *NP*-difficile, et dans l'autre sens, nous présentons une réduction naturelle vers GROUPTSP. Enfin, nous présentons un algorithme pour ce problème qui trouve souvent un meilleur ordre de visite des villes que l'ordre optimal pour ETSP, nous permettant ainsi de quantifier la différence entre les deux problèmes et de légitimer l'étude de VTSP. Notre algorithme repose sur une approche de type 2-opt utilisant un oracle pour calculer la trajectoire physique optimale correspondant à un ordre de visite donné. Cet oracle lui-mêmee est basé sur une version originale de l'algorithme A* dont la vocation est de traiter des solutions multipoints.

Mots-clefs : voyageur de commerce, planification de mouvements, contraintes d'accélération, problème d'optimization

Le problème de la visite d'un ensemble donné de sites et du retour au site initial tout en minimisant le coût total est connu sous le nom de problème du voyageur de commerce (dit TSP). Une instance de ce problème est souvent donnée sous la forme d'un graphe dont les sommets représentent les sites à visiter, les *villes*), et dont les poids sur les arrêtes représentent le coût du déplacement d'une ville à l'autre. En 1972, Karp a démontré que le problème du CYCLE HAMILTONIEN est NP-difficile, ce qui implique que la version originale de TSP (où une seule visite par ville est autorisée) est NP-difficile. En fait, le TSP en général est même inapproximable (si P $\neq$ NP). Du côté positif, bien que l'énumération de toutes les solutions possibles prenne un temps factoriel, Held et Karp ont montré que le problème peut être résolu en temps "seulement" $O(n^2 2^n)$, ce qui reste à ce jour le meilleur temps connu.

De nombreux cas particuliers du TSP ont été étudiés, motivés par des applications diverses. Les plus classiques sont METRIC TSP, où les coûts doivent satisfaire l'inégalité triangulaire, et EUCLIDEAN TSP (ETSP), où les villes ont des coordonnées dans le plan et les coûts correspondent aux distances euclidiennes. Bien que ces versions soient toujours NP-difficiles, il est bien connu que la première devient approximable à un facteur constant près et la seconde admet un schéma d'approximation (PTAS). Certaines versions du TSP ont aussi été proposées pour prendre en compte l'inertie des véhicules. La plus connue est le DUBINS TSP (DTSP) [LNFF07], qui limite la courbure d'une trajectoire par un rayon fixe. On peut également citer le DOUBLE INTEGRATOR TSP [SBF09], qui considère un modèle d'accélération en temps continu.

Dans cet article, nous présentons une nouvelle version du TSP appelée VECTORTSP, qui revient à utiliser comme modèle de contrainte physique un ensemble de règles discrètes très simples issues d'un jeu de papier et crayon connu sous le nom de RACETRACK (ou VECTOR RACER, entre autres). Dans une chronique recréative du Scientific American de 1973 [Gar73], Martin Gardner a présenté ce jeu, où chaque joueur contrôle la vélocité (vitesse et direction) de son véhicule. À chaque tour, la vélocité peut être modifiée d'au plus une unité dans chaque dimension. Le jeu est en fait un problème d'optimisation de la planification des

† La version anglaise de cet article a été présentée à ALGOSENSORS 2020 [CRS20]. Nous référons le ou la lectrice à cette version pour la liste complète des références bibliographiques, majoritairement omises dans ce résumé.

‡ Ce travail a été soutenu par le projet ANR ESTATE (ANR-16-CE25-0009-03).

mouvements, le paramètre naturel à minimiser étant le nombre de tours nécessaires pour qu'un véhicule atteigne l'arrivée. Dans [HM10], Holzer *et al.* montre que RACETRACK est en *NL*, et que vérifier si une stratégie donnée pour Racetrack est gagnante est *P*-dur. Dans [BBF+18], Bekos *et al.* étudient RACETRACK sur une simple piste rectangulaire de taille $L \times H$ d'une certaine largeur W, et montrent qu'on peut résoudre RACETRACK en temps $O(W^5)$, solution qu'ils généralisent ensuite pour des pistes de type Indianapolis.

Dans cet article, nous tentons de motiver l'étude du VECTORTSP en démontrant un certain nombre de résultats qui illustrent sa spécificité. Par manque de place, la plupart des preuves sont omises. Elles peuvent être trouvées dans la version officielle de cet article [CRS20], ainsi qu'une généralisation de certaines propriétés en dimensions supérieures.

1 Notions de base et formalisation

Les contraintes d'accélération RACETRACK peuvent être énoncées comme suit : dans le plan $\mathbb{Z}^2$, la *configuration* de l'entité mobile est de la forme (x, y, dx, dy), où (x, y) représente sa position, et (dx, dy) représente son vecteur vitesse. Initialement, au temps 0, la configuration du véhicule contient la position de départ (x_0, y_0) et le vecteur nul $(0, 0)$. Ensuite, à chaque pas de temps discret i, la composante dx (*resp.* dy) peut être modifiée au maximum d'une unité, de sorte que $dx_i = dx_{i-1}$ ou $dx_i = dx_{i-1} \pm 1$ (*resp.* $dy_i = dy_{i-1}$ ou $dy_i = dy_{i-1} \pm 1$). L'emplacement du véhicule au moment i est alors donné par $x_i = x_{i-1} + dx_i$ et $y_i = y_{i-1} + dy_i$. Une suite de configurations respectant ces contraintes est appelée *trajectoire*. La longueur d'une trajectoire est le nombre de configurations qui la composent. Un exemple est donné à la Figure 1.

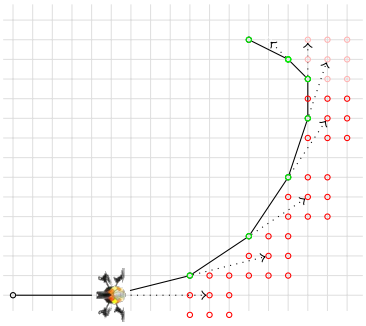

FIGURE 1: Exemple de trajectoire respectant les contraintes RACETRACK.

On peut définir une notion de *graphe des configurations*, dont les sommets correspondent à toutes les configurations possibles et les arcs indiquent quelles configurations peuvent se succéder directement. Une trajectoire devient alors un chemin dans ce graphe.

Remarque 1 (Folklore)**.** *Un espace délimité de taille $L \times L$ engendre un graphe des configurations de taille $O(L^3)$. Une trajectoire optimale d'une configuration donnée à une autre peut être trouvée en temps polynomial en effectuant un parcours en largeur (BFS) dans le graphe des configurations.*

Nous envisageons les paramètres suivants : (a) vitesse de visite ν : vitesse maximale à laquelle une ville est visitée, motivée par les arrêts de bus, la livraison, *etc.* ; (b) distance de visite α : distance maximale à laquelle une ville est visitée, similaire à TSP avec un voisinage ; (c) complétion du vecteur β : si les villes peuvent être visitées entre les configurations ($\beta = false$) ou non ($\beta = true$). Ce dernier paramètre peut représenter des cycles d'activités de réseaux de capteurs économisant la batterie, par exemple. Sauf indication contraire, les paramètres par défaut considérés dans ce document sont $\nu = \infty$, $\alpha = 0$ et $\beta = false$.

VECTORTSP peut maintenant être défini. Intuitivement, il s'agit de trouver une trajectoire de longueur minimum qui visite un ensemble de villes donné. Plus formellement :

Definition 1. VECTORTSP

> ***Input :*** *Un ensemble de n villes $P \subseteq \mathbb{Z}^2$, une ville de départ $p_0 = (x_0, y_0) \in P$, les paramètres ν, α et β, et la plus grande distance entre les villes L encodée en unaire.*
> ***Output :*** *Une trajectoire de longueur minimale $T = (c_0, \ldots, c_k)$ visitant toutes les villes de P, avec $c_0 = c_k = (x_0, y_0, 0, 0)$.*

Le rôle du paramètre L est de garantir que la longueur de la trajectoire optimale est polynomialement limitée par la taille de l'entrée. Sans cela, même une instance comportant seulement deux villes pourrait s'avérer difficile en raison de la seule distance qui les sépare. On peut définir la version décisionnelle de manière analogue (voir version complète de l'article).

FIGURE 2: Exemple de trajectoire réalisant un tour optimal pour ETSP (à gauche) et une trajectoire optimale pour VTSP (à droite), correspondant à des ordres de visites différents (avec un croisement pour VTSP).

Lemme 1 ([CRS20])**.** *Une solution doit contenir au moins $\sqrt{L}$ configurations et une solution optimale au plus $O(L^2)$ configurations. De plus, la partie utile du graphe des configurations comporte $O(L^4)$ configurations.*

On peut déjà remarquer quelques spécificités du problème VTSP vis à vis de ETSP. Par exemple, la ville de départ a un impact sur la trajectoire optimale. Une autre différence, plus fondamentale, est que :

Remarque 2. *Un ordre de visite* ETSP *optimal peut induire une trajectoire sous-optimale pour* VTSP*. De même, une trajectoire optimale pour VTSP peut contenir des croisements, ce qui est impossible pour ETSP.*

Ces propriétés sont illustrées à la Figure 2.

2 Complexité algorithmique

Nous présentons ici un résumé des principaux résultats de complexité algorithmique liés à VECTORTSP.

Théoreme 1 ([CRS20])**.** VECTORTSP *est NP-difficile.*

L'idée de preuve est de réduire EXACTCOVER à VECTORTSP en temps polynomial. Cette réduction est inspirée de la réduction de Papadimitriou pour montrer que ETSP est NP-difficile. Une limitation du résultat est que nous exigeons la vitesse de visite $v = 0$ ou $v = 1$ dans l'instance VECTORTSP produite, ce qui laisse la question suivante ouverte : VECTORTSP est-il aussi NP-difficile sans limitation de vitesse ?

Nous montrons aussi que VECTORTSP peut se réduire assez naturellement à GROUP TSP. L'idée est de considérer chaque configuration visitant une ville et de l'ajouter comme une ville propre à l'instance GROUP TSP créée. Chaque ville de l'instance VECTORTSP originale correspond alors à un groupe de villes dans l'instance GROUP TSP. Les arcs entre les villes créées sont ajoutés avec un poids, qui correspond à la longueur de la trajectoire RACETRACK la plus courte entre les deux configurations correspondantes.

3 Algorithmes

Nous proposons une heuristique appelée `flipVTSP`, qui est composée de deux parties : (1) l'exploration des ordres de visite possibles, et (2) l'implémentation d'un oracle calculant une trajectoire optimale pour un ordre de visite donné. La première partie est basée sur l'heuristique *flip* (également connue sous le nom de *2-opt*), communément utilisée pour ETSP. Elle est particulièrement adaptée à VECTORTSP pour deux raisons : elle peut fonctionner sans coûts sur les arêtes (qui ne sont pas donnés dans VECTORTSP car ils dépendent de l'ordre lui-même), et elle explore les ordres de visite qu'il y ait ou non des croisements.

L'heuristique commence par un ordre de visite initial arbitraire, qui est évalué par l'oracle de trajectoire. Les flips possibles (suppression de deux segments et re-connexion de leurs extrémités) dans cet ordre sont tous testés en utilisant à nouveau l'oracle de manière répétée. Si un meilleur ordre de visite est trouvé, l'algorithme récurse sur ce nouvel ordre de visite, jusqu'à ce que l'on obtienne un ordre correspondant à un minimum local (dans l'espace des ordres), appelé ordre 2-optimal (le "2" faisant ici référence au nombre de segments échangés, et non à un facteur d'approximation).

Lemme 2. *Une trajectoire correspondant à un ordre 2-optimal peut être calculée en temps $O(n^2L^2\tau)$, où τ est la complexité en temps de l'oracle de trajectoire.*

Notre implémentation de l'oracle de trajectoire est basée sur l'algorithme A^* que nous adaptons à la visite de plusieurs points dans un ordre *donné* en tenant compte des contraintes RACETRACK.

Pour cela, nous utilisons une fonction d'estimation des coûts que notre adaptation de A^* utilise pour se guider efficacement dans le graphe des configurations. Cette fonction d'estimation calcule les coûts correspondant aux trajectoires unidimensionnelles visitant les villes selon chaque dimension prise séparément (voir Figure 3). Elle renvoie le maximum entre les coûts obtenus pour chaque dimension, ce coût étant par définition inférieur au coût réel (multi-dimensionnel) de la trajectoire, ce qui est (connu pour être) une condition nécessaire et suffisante pour que A^* trouve l'optimum.

Lemme 3. *L'oracle de la trajectoire s'excécute en temps* $\widetilde{O}(n^2L^4)$.

Théoreme 2. *Une trajectoire 2-optimale peut être calculée en temps polynomial, plus précisément en temps* $\widetilde{O}(n^4L^6)$.

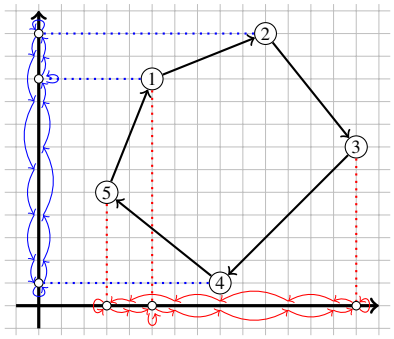

FIGURE 3: Projection des villes à visiter dans chaque dimension.

4 Résultats expérimentaux

Nous avons fait des expériences pour mesurer la différence entre ETSP et VTSP, ainsi que pour valider notre heuristique `flipVTSP`. Grâce à l'utilisation du solveur *Concorde*, nous pouvons quantifier dans quelle mesure un ordre de visite ETSP optimal peut être amélioré grâce à l'utilisation de `flipVTSP`. Ces

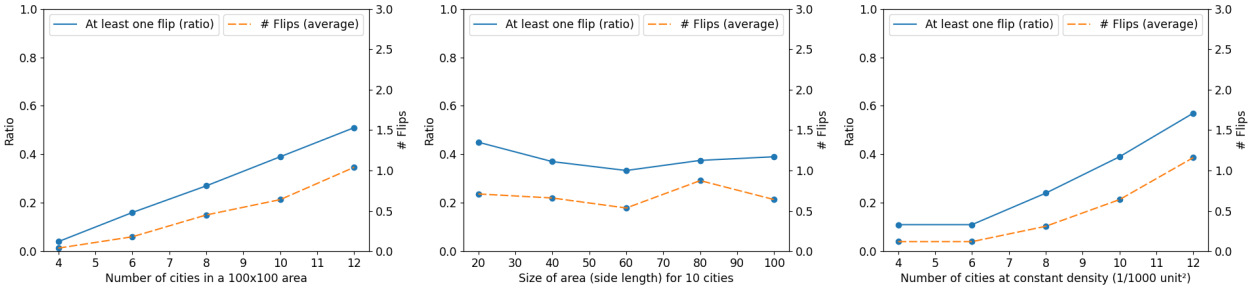

FIGURE 4: En variant le nombre de villes (à gauche), la taille de la zone (au milieu) et les deux (à droite), ces graphiques montrent respectivement la probabilité que notre heuristique effectue au moins un flip à partir d'une solution ETSP optimale (bleu), et le nombre moyen de flips effectués (orange), sur 100 instances.

résultats (voir Figure 4) suggèrent notamment qu'un ordre de visite optimal pour ETSP a moins de chances d'être optimal pour VTSP à mesure que le nombre de villes augmente. De ce fait, le problème VTSP semble mériter d'être étudié comme un problème indépendant d'ETSP, dont les solutions sont propres, y compris en ce qui concerne les ordres de visites (et non seulement leur conversion en trajectoire).

Références

[BBF+18] Michael A Bekos, Till Bruckdorfer, Henry Förster, Michael Kaufmann, Simon Poschenrieder, and Thomas Stüber. Algorithms and insights for racetrack. *Theoretical Computer Science*, 2018.

[CRS20] Arnaud Casteigts, Mathieu Raffinot, and Jason Schoeters. Vectortsp : A traveling salesperson problem with racetrack-like acceleration constraints. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 45–59. Springer, 2020.

[Gar73] M Gardner. Sim, chomp and race track-new games for intellect (and not for lady luck). *Sc. Ame.*, 228(1) :108–115, 1973.

[HM10] Markus Holzer and Pierre McKenzie. The computational complexity of racetrack. pages 260–271, 2010.

[LNFF07] Jerome Le Ny, Emilio Frazzoli, and Eric Feron. The curvature-constrained trav. sal. problem for high point densities. In *2007 46th IEEE Conference on Decision and Control*, pages 5985–5990. IEEE, 2007.

[SBF09] Ketan Savla, Francesco Bullo, and Emilio Frazzoli. Traveling salesperson problems for a double integrator. *IEEE Transactions on Automatic Control*, 54(4) :788–793, 2009.