VECTORTSP : *un problème de voyageur de commerce avec des 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 une entité mobile 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) il n'y a pas de limitations de vitesse, et (2) l'inertie dépend de la vitesse actuelle. Ce modèle est donc plus proche des modèles typiquement considérés dans des problèmes de planification de mouvements, et est ici appliqué à la visite de n villes dans un ordre non-predéterminé. Nous présentons ce problème en discutant des différences fondamentales avec d'autres versions de TSP. En particulier, un ordre de visite optimal pour ETSP peut ne pas être optimal pour VTSP. Nous démontrons que VECTORTSP est *NP*-difficile, et dans l'autre sens, que VECTORTSP se réduit vers GROUPTSP en temps polynomial. D'un point de vue algorithmique, nous proposons un schéma interactif entre un algorithme haut niveau et un oracle de trajectoire. Le premier permet de calculer l'ordre de visite, tandis que le second calcule le coût (ou la trajectoire) pour un ordre de visite donné. Nous présentons des algorithmes pour les deux, et nous démontrons et quantifions expérimentalement que cette approche trouve souvent une meilleure solution que la trajectoire optimale résultant d'un ordre de visite optimal ETSP.

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

1 Introduction

The problem of visiting a given set of places and returning to the starting point, while minimizing the total cost, is known as the Traveling Salesperson Problem (TSP, for short). An instance of the problem can be given as a graph whose vertices represent the places to visit (often referred to as *cities*) and weights on the edges represent the cost of moving from one city to another. Karp proved in 1972 that the HAMILTONIAN CYCLE problem is *NP*-hard, which implies that TSP is *NP*-hard [Kar72]. TSP was subsequently shown to be inapproximable (unless $P = NP$) by Orponen and Manilla in 1990 [OM90]. On the positive side, while the trivial algorithm has a factorial running time (essentially, evaluating all permutations of the visit order), Held and Karp presented a dynamic programming algorithm [HK62] running in time $O(n^2 2^n)$, which as of today remains the fastest known exact algorithm.

Many special cases of TSP are considered in the literature, motivated by applications in various areas, such as vehicle routing, computer wiring, stock cutting, and DNA reconstruction. We will go over the most relevant ones *w.r.t.* this paper. In EUCLIDEAN TSP (ETSP, for short), the cities are points in the plane, and weights are the Euclidean distance between cities. This problem, although still *NP*-hard (proven by Papadimitriou [Pap77] and Garey *et al.* [GGJ76]), was shown to admit a polynomial-time approximation scheme (PTAS) by Arora [Aro96] and Mitchell [Mit99]. DUBINS TSP (DTSP), which is also *NP*-hard (proven by Le Ny *et al.* [LNFF07]), accounts for inertia by bounding the curvature of a trajectory by a fixed radius. This may represent for example a vehicle with wheels of bounded turning angles, such as cars. Savla *et al.* [SBF09] introduce the DOUBLE INTEGRATOR TSP, which considers a continuous-time model of acceleration for the visiting vehicle, and give some probabilistic results on the stochastic version of the problem.

† La version anglaise de ce travail a été publiée à ALGOSENSORS 2020 [CRS20].

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

The aim of this paper is to introduce and study a version of TSP which may accurately represent a visiting vehicle with acceleration, *i.e.* which can use sharp turns at low speed, but experiences inertia forces at high speeds. In contrast to Savla *et al.*'s continuous model (see [SBF09]) which is more related to control theory, we focus on a discrete model of acceleration which is naturally prone to algorithmic investigation.

In a recreative column of the *Scientific American* in 1973 [Gar73], Martin Gardner presented a paper-and-pencil game known as RACETRACK (as well as Vector Racer, Graph Racer, Vector Rally, *etc.*). The game is played by each player controlling its respective vehicle's speed, which, each round, may be modified in a simple and discrete manner. The game is in fact a motion planning optimization problem, with the natural parameter to minimize being the amount of rounds needed for a mobile entity to arrive at the finish area. In [HM10], Holzer *et al.* show RACETRACK is in *NL*, and verifying if a given strategy for Racetrack is winning is *P*-hard which contradicts the popular "conjecture" stating that all interesting and fun games are *NP*-hard. In [BBF$^+$18], Bekos *et al.* are study RACETRACK in the Indianapolis track, a simple rectangular track of size $L \times H$ of some width W, and show that one can solve RACETRACK in time $O(W^5)$, which they generalize for Indianapolis-like tracks. They also consider heuristics for "limited view" vehicles in these tracks.

In this paper, we take the model of acceleration used in RACETRACK and apply it to the visiting vehicle in a TSP setting, creating the VECTORTSP.

Due to space limitations, we will present our results only for two dimensions. Also, some proofs may be omitted or replaced by a proof idea.

2 Preliminaries

The RACETRACK acceleration constraints can be stated as follows. In the plane $\mathbb{Z}^2$, the mobile entity's *configuration* is of the form (x, y, dx, dy), where (x, y) represents its position, and (dx, dy) represents its velocity vector. Initially, at time 0, the entity's configuration contains the starting position (x_0, y_0) and the null vector $(0,0)$. Now, every discrete time step i, component dx (*resp.* dy) may be modified by at most one unit, so that $dx_i = dx_{i-1}$ or $dx_i = dx_{i-1} \pm 1$ (*resp.* $dy_i = dy_{i-1}$ or $dy_i = dy_{i-1} \pm 1$). The entity's location at time i is then given by $x_i = x_{i-1} + dx_i$ and $y_i = y_{i-1} + dy_i$. A sequence of configurations obeying the above constraints is called a *trajectory*. The length of a trajectory is the number of configurations it is composed of. An example of a RACETRACK trajectory is given in Figure 1.

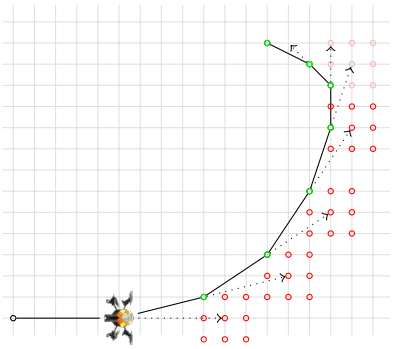

FIGURE 1: Example of a RACETRACK trajectory. Due to a significant initial velocity to the right, it takes a few rounds for the mobile entity to effectively turn its left.

The configuration space is defined as the directed graph with vertices corresponding to all possible configurations the mobile entity may attain, and arcs from one vertex to another if the mobile entity can change its configuration correspondingly in one time step. Note that a trajectory is a path in the configuration space.

Fact 1 (Folklore)**.** *A bounded space of size $L \times L$ results in a configuration space of size $O(L^3)$.*

Corollary 1 (Folklore)**.** RACETRACK *in a space of size $L \times L$ can be solved in polynomial time, through a breadth-first search in the configuration space.*

Informally, VECTORTSP is the problem of finding a minimum length trajectory which visits a given set of cities. Determining whether a city is visited is controlled by the following parameters :

— Visit speed ν : maximum speed at which a city is visited, motivated by bus stops, delivery, *etc.* ;
— Visit distance α : maximum distance at which a city is visited, similar to TSP with neighborhood ;
— Vector completion β : whether cities can be visited in-between configurations ($\beta = false$) or not ($\beta = true$). This may represent settings of sensor networks optimizing battery consumption.

The *default settings* considered in this paper (unless stated otherwise) are $\nu = \infty$, $\alpha = 0$ and $\beta = false$. VECTORTSP can now defined as follows.

Definition 1. VECTORTSP

> ***Input :*** *A set of n cities $P \subseteq \mathbb{Z}^2$, a starting city $p_0 = (x_0, y_0) \in P$, parameters ν, α and β, and the largest distance between cities L encoded in unary.*
> ***Output :*** *A minimum length trajectory $T = (c_0, \ldots, c_k)$ visiting all cities in P, with $c_0 = c_k = (x_0, y_0, 0, 0)$.*

The role of parameter L is to guarantee that the length of the optimal trajectory is polynomially bounded in the size of the input. Without it, an instance of even two cities could be artificially hard due to the sole distance between them. The decision version of the problem can be defined in a natural manner.

Fact 2. *The starting city has an impact on the optimal trajectory's length.*

Fact 3. *An optimal* ETSP *visit order may not be optimal for* VTSP.

Fact 4. *An optimal trajectory may self-cross.*

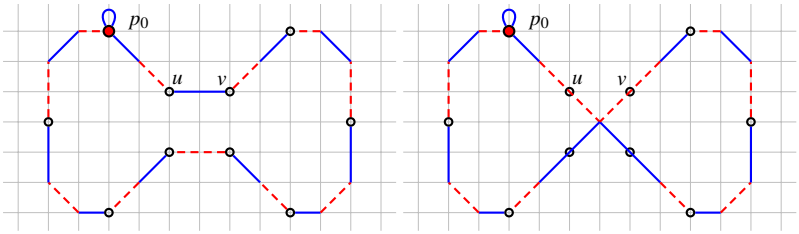

FIGURE 2: Example with optimal trajectory (in alternating red/blue) obeying an optimal ETSP visit order, versus crossing optimal trajectory.

These facts are illustrated through an example (see Figure 2).

Lemma 1. *A solution must contain at least $\sqrt{L}$ configurations, and at most $O(L^2)$ configurations.*

Lemma 2. *The configuration space can be bounded by $O(L^4)$ vertices.*

3 Computational complexity

Theorem 1. VECTORTSP *is NP-hard.*

The proof idea is to reduce EXACTCOVER to VECTORTSP in polynomial time. This reduction is heavily based on Papadimitriou's (lengthy) reduction used to show ETSP is *NP*-hard. The only inconvenience is that we require the visit speed $\nu = 0$ or $\nu = 1$ in the obtained VECTORTSP instance, which leaves the open question :

Open question 1. *Is* VECTORTSP *NP-hard in the default settings ?*

Lemma 3. VECTORTSP *reduces to* GROUP TSP *in polynomial time.*

Corollary 2. VECTORTSP *reduces to* ASYMMETRIC TSP *in polynomial time.*

Corollary 3. VECTORTSP *reduces to* SYMMETRIC TSP *in polynomial time.*

The proof idea here is to consider each configuration visiting a city, and adding it as a proper city to the created GROUP TSP instance. Each city in the original VECTORTSP instance then corresponds to a group of cities in GROUP TSP. Arcs between created cities are added with a weight, which corresponds to the shortest path's length between the two configurations in the configuration space. The corollaries are due to [NB93] and [KP80].

4 Algorithms

We propose a heuristic which we call FLIPVTSP which is composed of two main parts : the high-level exploration of visit orders, and a trajectory oracle.

The high-level part is based on the *flip* heuristic (also known as *2-opt*), commonly used for ETSP. It is surprisingly suitable for VECTORTSP for two reasons. First, it can function without edge costs (which aren't given in VECTORTSP and cannot be computed independently of a visit order). Secondly, it explores self-crossing visit orders, which may be optimal (see Fact 4). The heuristic starts with an initial visit order,

which is evaluated through the trajectory oracle, and then "flipped" in every possible way and re-evaluated. If a better visit order is found, the heuristic repeats on this visit order, or else it returns the best solution found. This results in a local optimum, known as a 2-optimal trajectory.

Lemma 4. *A 2-optimal trajectory can be computed in time $O(n^2L^2\tau)$, where τ is the time complexity of the trajectory oracle.*

The trajectory oracle is based on the A^* algorithm. As opposed to the standard A^*, we adapt it to the visit of multiple points in a given order, in the RACETRACK setting. For this, we use a cost estimation function which our adaptation of A^* uses to guide itself efficiently through the configuration space. This estimation function computes costs of corresponding one-dimensional projections of the cities (see Figure 3), and returns the maximum between the two obtained costs.

Lemma 5. *The cost estimation for any configuration can be computed in time $O(n)$.*

Lemma 6. *The trajectory oracle runs in time $\widetilde{O}(n^2L^4)$.*

Theorem 2. *A 2-optimal trajectory can be computed in polynomial time, more precisely in time $\widetilde{O}(n^4L^6)$.*

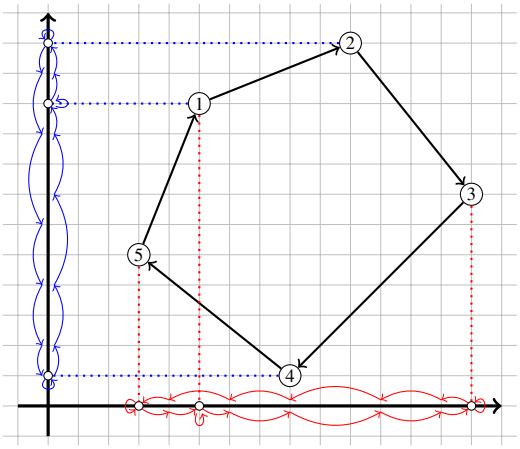

FIGURE 3: Projecting the cities to visit in each dimension, and solving the resulting one-dimensional instances.

5 Experiments and conclusion

We conduct experiments to quantify the difference between ETSP and VTSP, as well as to validate our flipVTSP heuristic. We note, through the use of the `Concorde` TSP solver, how likely it is for an optimal ETSP visit order to be improved for VTSP, through the use of flipVTSP. See Figure 4 for some measures when varying different parameters. These experimental results suggest that an optimal ETSP visit order becomes less likely to be optimal for VTSP as the number of cities increases. In fact, the plots may under-estimate the impact of VTSP since our heuristic, already resulting in a local optimum, was modified to optimize practical running time in these experiments, which potentially resulted in a worse quality of solution.

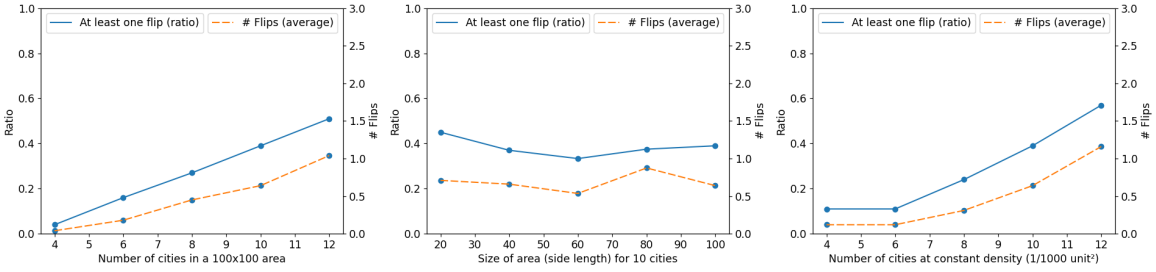

FIGURE 4: Varying the number of cities (left), size of the area (middle), and both (right). The plots show the likelyhood of at least one flip and the average number of flips (over 100 instances).

As a conclusion, we introduced a new version of `TSP`, in which controlling acceleration of a visiting vehicle plays a key role. We showed some preliminary results which already distinguish the problem from the well-known `Euclidean TSP`. We showed *NP*-completeness for the problem and presented multiple reductions. As algorithmic results, we proposed a heuristic using a combination of the flip heuristic with the A^* algorithm. Finally, we presented some experimental results to validate said heuristic and quantify the difference between `ETSP` and `VTSP`.

Références

[Aro96] Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 2–11. IEEE, 1996.

[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). *Scientific American*, 228(1) :108–115, 1973.

[GGJ76] Michael R Garey, Ronald L Graham, and David S Johnson. Some np-complete geometric problems. pages 10–22, 1976.

[HK62] Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1) :196–210, 1962.

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

[Kar72] Richard M Karp. Reducibility among combinatorial problems. pages 85–103, 1972.

[KP80] Paris-C Kanellakis and Christos H Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28(5) :1086–1099, 1980.

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

[Mit99] Joseph SB Mitchell. Guillotine subdivisions approximate polygonal subdivisions : A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM Journal on computing*, 28(4) :1298–1309, 1999.

[NB93] Charles E Noon and James C Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR : Information Systems and Operational Research*, 31(1) :39–44, 1993.

[OM90] Pekka Orponen and Heikki Mannila. On approximation preserving reductions : Complete problems and robust measures (revised version). *Department of Computer Science, University of Helsinki*, 1990.

[Pap77] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3) :237–244, 1977.

[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.