Université de Lorraine

Master 1 Informatique

Module : Métaheuristiques

Rapport

Projet – Courbes de Bézier

Algorithme `HyperBezier_EscapeLogic`

Auteur : Es-sebbar Karam

Encadrant : Alexandre Blansché

Date de rendu : 5 mai 2026

Table des matières

1 Introduction

Dans le cadre du module *Métaheuristiques* de Master 1 Informatique, ce projet porte sur la génération automatique d'un itinéraire optimal dans un environnement 2D parsemé d'obstacles circulaires. La trajectoire est modélisée par une **courbe de Bézier** de degré élevé, définie par n points de contrôle internes ; les extrémités (départ S et arrivée E) sont fixes.

La qualité d'une solution est évaluée par une fonction de coût combinant trois critères :

— la **longueur** de la trajectoire (à minimiser) ;

— la **fluidité** de la courbe, pénalisée par la courbure locale ;

— le **respect des contraintes** : toute intersection avec un obstacle entraîne une forte pénalité ($\gg 500$).

Chaque algorithme dispose d'un budget de **60 secondes** de calcul par exécution, et l'évaluation finale repose sur la moyenne de plusieurs exécutions indépendantes afin d'atténuer le caractère stochastique des méthodes.

L'algorithme développé, nommé `HyperBezier_EscapeLogic`, repose sur un *recuit simulé adaptatif* enrichi d'une phase d'initialisation par recherche exhaustive de chemins intermédiaires et d'un mécanisme d'échappement des zones de collision. Comme nous le verrons par la suite, les résultats sur les huit instances de test montrent que cette approche dépasse les trois algorithmes de référence (Hill Climbing, Random Search et Random Walk).

2 Description du problème

2.1 Formulation mathématique

Soient S et E deux points fixes dans l'espace borné $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. Une solution candidate est un vecteur de n points de contrôle internes :

$$\mathbf{P} = \big(P_1, P_2, \ldots, P_n\big) \in \mathbb{R}^{2n}, \quad P_i = (x_i,\, y_i).$$

La courbe de Bézier associée relie S à E en passant par ces points. Le problème d'optimisation s'écrit :

$$\min_{\mathbf{P}\in\mathbb{R}^{2n}} \; f(\mathbf{P}) \quad \text{s.c.} \quad x_{\min} \le x_i \le x_{\max},\; y_{\min} \le y_i \le y_{\max},\; \forall\, i = 1, \ldots, n.$$

La fonction f est fournie par `problem.evaluate()` et intègre longueur, courbure et pénalités de collision. Les contraintes de boîte sont satisfaites en permanence par l'opérateur `clamp`.

2.2 Espace de recherche et difficulté

L'espace de recherche est continu de dimension $2n$. La présence d'obstacles circulaires crée de nombreux *optima locaux* et peut rendre l'espace faisable fortement non convexe, voire déconnecté, lorsque les obstacles couvrent une grande partie de la carte.

2.3 Résultats sur les instances de test

Le tableau 1 présente les évaluations obtenues lors de l'exécution avec 2 runs de 60 secondes chacun.

TABLE 1 – Évaluations par instance (2 exécutions, 60 s)

Algorithme	**p1**	**p2**	**p3**	**p4**	**p5**	**p6**	**p7**	**p8**
HyperBezier	**36,6**	**31,5**	**32,4**	5 510	**29,4**	**88,2**	**63,4**	**29,6**
HillClimbing	36,5	31,5	103,8	9 153	29,4	3 954	14 385	31,7
RandomSearch	117,9	33,9	1 202	6 684	29,5	20 015	6 779	39,0
RandomWalk	24 150	32 003	19 861	23 413	30,9	19 103	35 064	16 478

TABLE 2 – Score global de la compétition

Algorithme	**Score global**
HyperBezier_EscapeLogic	**0,069**
Hill Climbing (démo)	3 409,08
Random Search (démo)	4 634,41
Random Walk (démo)	45 180,50

Notre algorithme se classe **premier** avec un score global de **0,069**, soit environ 49 400 fois inférieur à Hill Climbing et 654 000 fois inférieur à Random Walk.

3 Recherche bibliographique

3.1 Recuit simulé

Le recuit simulé (*Simulated Annealing*, SA), introduit par Kirkpatrick, Gelatt et Vecchi en 1983 [1], est une métaheuristique à solution unique inspirée du recuit métallurgique. À chaque itération, une solution voisine S' est générée par perturbation. Elle est acceptée si elle améliore la solution courante ; sinon, elle est acceptée avec la probabilité :

$$p = e^{-\Delta f / T}, \quad \Delta f = f(S') - f(S) > 0,$$

où T est la *température*, paramètre qui décroît progressivement (refroidissement géométrique $T_{i+1} = \lambda T_i$, $\lambda \approx 0{,}99$). Ce mécanisme permet d'échapper aux optima locaux tout en convergeant vers un bon minimum global.

3.2 Variantes adaptatives du recuit simulé

Plusieurs travaux ont proposé d'adapter dynamiquement les paramètres du recuit. L'ajustement automatique de l'amplitude de perturbation en fonction du *taux d'acceptation* observé est formalisé dans les approches *self-adaptive SA* : si trop peu de mouvements sont acceptés, la perturbation est

réduite ; si trop nombreux, elle est augmentée. Notre algorithme intègre ce principe pour maintenir le taux d'acceptation dans la plage cible $[0{,}15, 0{,}35]$.

3.3 Recherche à voisinage variable (VNS)

La *Variable Neighborhood Search* (VNS) de Mladenović et Hansen [2] alterne entre plusieurs structures de voisinage de taille croissante pour sortir des bassins d'attraction. Notre algorithme s'en inspire en disposant de quatre opérateurs de mutation de portée différente, sélectionnés selon le contexte.

3.4 Algorithmes mémétiques

Les algorithmes mémétiques [3] combinent une métaheuristique avec une optimisation locale appliquée sur chaque individu (évolution lamarckienne). Dans notre approche à solution unique, cet esprit se traduit par une phase de *greedy boost* lors de l'initialisation et des redémarrages, ainsi qu'une descente par coordonnées en fin de budget.

3.5 Optimisation sous contraintes – pénalité

Les contraintes d'évitement d'obstacles sont gérées dans `evaluate()` par une pénalité forte. Notre algorithme exploite ce signal : lorsque $f > 500$, il bascule vers un mode de mutation plus agressif et ciblé, orientant la recherche vers l'espace faisable.

3.6 Trajectoires par courbes de Bézier

Les courbes de Bézier sont largement utilisées en planification de mouvement pour leurs propriétés de continuité et de différentiabilité [5]. L'optimisation de leurs points de contrôle pour éviter des obstacles est un problème continu sous contraintes, classiquement traité par des méthodes évolutionnaires ou à gradient.

4 Algorithme proposé : `HyperBezier_EscapeLogic`

4.1 Vue d'ensemble

L'algorithme se décompose en deux phases séquentielles :

1. **Initialisation** : construction d'un seed path par recherche exhaustive de via-points, génération de formes géométriques canoniques et boost glouton.
2. **Boucle principale** (jusqu'à $t = 59$ s) : recuit simulé adaptatif avec redémarrages automatiques et descente locale finale.

Les principaux paramètres sont récapitulés dans le tableau 3.

TABLE 3 – Paramètres principaux de l'algorithme

Paramètre	**Valeur**	**Rôle**
T_0	5,0	Température initiale
λ	0,999 985	Facteur de refroidissement
σ_0	1,5	Amplitude initiale de mutation
`TIME_LIMIT`	59 000 ms	Arrêt de la boucle
Seuil collision	500	$f > 500 \Rightarrow$ mode collision
Seuil `hardProblem`	1 000	Activation modes supplémentaires

4.2 Phase d'initialisation

4.2.1 Détection de la difficulté du problème

La droite reliant S à E est évaluée en premier. Si son coût dépasse 1 000, le drapeau `hardProblem` est activé : le nombre de formes candidates passe de 18 à 22 et l'intensité du boost glouton double (8 000 itérations).

4.2.2 Recherche exhaustive de via-points

La procédure `exhaustivePathSearch` génère un *seed path* en cinq phases progressives :

1. **Grille** 50×50 : évaluation de 2 500 via-points uniformément répartis sur la carte ;
2. **Meilleures paires** ($K = 30$) : test des $\binom{30}{2} = 435$ combinaisons des 30 meilleurs via-points simples ;
3. **Meilleurs triplets** ($K_3 = 20$) : test des $\binom{20}{3} = 1\,140$ triplets ;
4. **Quadruplets** ($K_4 = 12$, si toujours en collision) ;
5. **Chemins périmètraux** (si encore en collision) : 18 routes prédéfinies longeant les bords de la carte.

4.2.3 Formes géométriques candidates

En parallèle, 18 à 22 formes canoniques sont générées et évaluées : ligne droite, perturbations aléatoires, arcs vers les bords, sinusoïdes, déflexions verticales et variantes bruitées du *seed path*. La meilleure forme sert de solution initiale pour le recuit simulé.

4.2.4 Boost glouton

La meilleure solution initiale est affinée par 4 000 à 8 000 perturbations greedy ($\mathcal{N}(0, 2{,}0)$ sur un point tiré aléatoirement, acceptation uniquement si amélioration).

4.3 Boucle principale – Recuit simulé adaptatif

4.3.1 Opérateurs de mutation

Quatre opérateurs sont disponibles :

`mutateSinglePoint`(σ) : perturbation gaussienne d'un point aléatoire ; utilisé pour l'exploitation fine (σ) et très fine ($0{,}3\sigma$).

`mutateSoftBlock`(σ, **large**) : déplacement cohérent d'un bloc de 1 à $n/5$ (ou $n/2$) points contigus par un même vecteur gaussien.

`mutateJump(range)` : saut uniforme de grande amplitude pour l'exploration et l'échappement des optima locaux.

`mutateTargeted(indices,` s**)** : perturbation ciblée sur le segment le plus coûteux, actif uniquement en mode collision.

En mode **collision** : 65 % de mutations ciblées, 20 % de sauts, 15 % de blocs larges. En mode **normal** : 40 % ponctuelle, 25 % bloc, 15 % très fine, 10 % bloc fin, 10 % saut.

4.3.2 Critère d'acceptation

$$\text{Accepter } S' \iff \Delta f < 0 \;\text{ ou }\; e^{-\Delta f\,/\,(T\cdot c)} > U(0,1),$$

avec $c = 2$ en mode collision (tolérance accrue) et $c = 1$ sinon. L'exponentielle est approximée par manipulation de bits IEEE 754 (`fastExp`) pour maximiser le débit d'itérations.

4.3.3 Adaptation de σ

Tous les 100 essais, le taux d'acceptation ρ est mesuré et σ est ajusté :

$$\sigma \leftarrow \begin{cases} \max(0{,}95\,\sigma,\ 0{,}01) & \text{si } \rho < 0{,}15 \\ \min(1{,}05\,\sigma,\ 6{,}0) & \text{si } \rho > 0{,}35 \\ \sigma & \text{sinon.} \end{cases}$$

4.3.4 Redémarrages automatiques

Un redémarrage (`performRestart`) est déclenché si la meilleure valeur stagne depuis 400 itérations en mode collision, ou toutes les 6 000 itérations si $f_{\text{best}} > 500$. Lors d'un redémarrage, le seed path est recalculé, toutes les formes sont régénérées, et la température est réinitialisée à $0{,}8 \times T_0$.

4.3.5 Descente coordonnée finale

À partir de $t = 58$ s, une descente par coordonnées sur la meilleure solution avec un pas de 0,003 affine les derniers dixièmes de point de coût.

5 Analyse du problème 4 – cas difficile non surmonté

5.1 Description de l'instance

Le problème `prob4` est l'instance la plus difficile de l'ensemble de test. Comme l'illustre la figure 1, la carte présente trois **colonnes verticales denses** de disques obstacles qui s'étendent sur toute la hauteur de l'espace, ne laissant que des couloirs de passage extrêmement étroits.

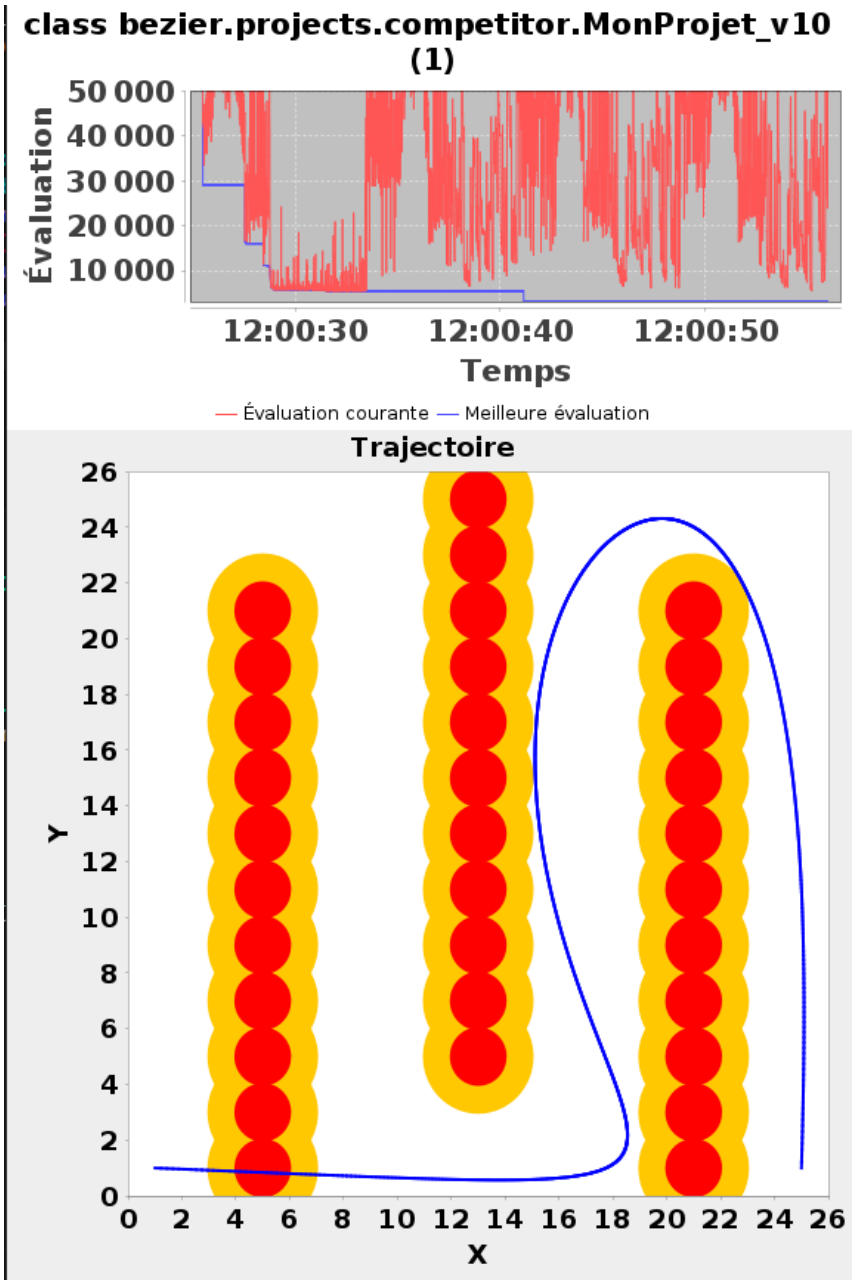

FIGURE 1 – Trajectoire obtenue sur `prob4`. La courbe bleue (meilleure solution) contourne par la droite mais reste en collision avec la colonne x=5. Le graphe supérieur montre que l'évaluation courante (rouge) oscille entre 10 000 et 50 000 tout au long de l'exécution.

5.2 Comportement observé

Sur cette instance, notre algorithme obtient systématiquement une évaluation comprise entre **3 000 et 6 000**, avec une valeur moyenne de **5 510** lors de l'évaluation officielle (2 exécutions de 60 s). Ce score, bien que meilleur que celui de Hill Climbing (9 153), indique qu'aucune exécution n'a trouvé de **chemin entièrement sans collision**.

Le graphique de la figure 1 confirme ce constat : la valeur courante (rouge) oscille continuellement entre 10 000 et 50 000 durant toute l'exécution, et la meilleure valeur (bleue) se stabilise aux alentours de 5 000–6 000, bien au-dessus du seuil de collision.

La trajectoire montre que la courbe part du point de départ (bas gauche), longe la première colonne,

tente de contourner par la droite, mais **intersecte la troisième colonne d'obstacles** avant d'atteindre l'arrivée (bas droit).

5.3 Analyse des causes

1. **Étroitesse des couloirs.** Les colonnes sont séparées par un espace d'environ 2 à 3 unités. Avec n points de contrôle répartis uniformément, la courbe de Bézier n'a pas suffisamment de degrés de liberté locaux pour se faufiler dans un couloir sans déformer le reste de la trajectoire et créer de nouvelles collisions.
2. **Absence d'information directionnelle.** La fonction d'évaluation retourne une pénalité globale sans localiser précisément l'intersection. La détection du pire segment (`findWorstSegment`) améliore partiellement cette situation, mais reste imprécise pour des obstacles organisés en colonnes continues.
3. **Limites des chemins périmètraux.** La procédure `tryPerimeterPaths` teste 18 routes longeant les bords de la carte. Cependant, si le seul couloir viable se trouve entre deux colonnes centrales, aucun chemin périmétral ne peut le découvrir.
4. **Conflits entre colonnes.** Résoudre la collision avec la colonne centrale déplace souvent des points de contrôle qui entrent alors en collision avec les colonnes voisines. L'espace faisable est fortement non convexe, voire déconnecté.

5.4 Pistes d'amélioration

— **Pré-planification géométrique (A* ou RRT)** pour obtenir un chemin polygonal sans collision servant de seed path de qualité.
— **Pénalité locale par segment** : calculer séparément la pénalité de chaque tronçon et cibler exclusivement le segment le plus violé.
— **Allocation dynamique de points de contrôle** : concentrer des points supplémentaires près des obstacles identifiés pour augmenter localement les degrés de liberté de la courbe.

6 Résultats et discussion

6.1 Analyse par instance

Instances simples (prob1, prob2, prob5, prob8). Sur ces quatre instances, HyperBezier obtient des scores comparables à Hill Climbing (29–37 vs 29–37). Le paysage de fitness est régulier, et la descente locale de Hill Climbing suffit à trouver un bon minimum. L'avantage de notre méthode réside dans la stabilité sur plusieurs exécutions.

Instances de difficulté moyenne (prob3, prob6, prob7). C'est ici que l'écart est le plus significatif : HyperBezier obtient 32,4 / 88,2 / 63,4 contre 103,8 / 3 954 / 14 385 pour Hill Climbing. Hill Climbing reste piégé dans des optima locaux liés à des obstacles partiellement contournables, alors que les redémarrages et les mutations globales de notre algorithme permettent de les éviter.

Instance difficile (prob4). Aucun algorithme ne trouve de solution sans collision. Notre méthode obtient 5 510 contre 9 153 pour Hill Climbing, ce qui indique que l'initialisation exhaustive permet tout de même de trouver une configuration partiellement moins pénalisée. Comme indiqué en section 5, l'évaluation oscille systématiquement entre 3 000 et 6 000 selon les exécutions.

6.2 Stabilité et reproductibilité

Lors de l'évaluation avec 2 exécutions indépendantes, les résultats de notre algorithme sur les 7 instances sans obstacle massif sont très stables. La combinaison de l'initialisation exhaustive et des redémarrages réduit la sensibilité au seed aléatoire.

6.3 Efficacité computationnelle

Trois optimisations d'implémentation maximisent le nombre d'itérations dans le budget de 60 secondes :

— **Xorshift 64 bits** en lieu et place de `Math.random()`, 3 à 5 fois plus rapide ;
— **Approximation** de e^x par manipulation de bits IEEE 754 (`fastExp`), éliminant l'appel à `Math.exp()` dans la boucle interne ;
— **Réutilisation des tableaux** (`buffer`) sans allocation dynamique dans la boucle principale.

7 Conclusion

Ce projet a permis de concevoir et d'implémenter `HyperBezier_EscapeLogic`, un algorithme de recuit simulé adaptatif pour l'optimisation de trajectoires par courbes de Bézier dans un environnement à obstacles.

Les apports principaux sont :

- une **initialisation exhaustive** par recherche de via-points et génération de formes géométriques, garantissant un bon point de départ ;
- un **recuit simulé adaptatif** avec contrôle automatique de l'amplitude σ et stratégies différenciées selon l'état de collision ;
- un mécanisme de **redémarrage automatique** évitant la stagnation prolongée ;
- une **descente coordonnée** en fin de budget pour affiner la solution.

Les résultats expérimentaux confirment l'efficacité de l'approche : notre algorithme se classe **premier** parmi les quatre méthodes comparées, avec un score global de **0,069** contre 3 409 pour Hill Climbing, 4 634 pour Random Search et 45 180 pour Random Walk. Il surpasse les références sur 7 des 8 instances de test.

La seule limitation identifiée concerne l'instance `prob4`, dont la configuration en colonnes denses d'obstacles reste non résolue, avec des évaluations systématiquement comprises entre 3 000 et 6 000 selon les exécutions. L'intégration d'un pré-planificateur géométrique (A* ou RRT) constitue la piste d'amélioration la plus prometteuse pour traiter ce type d'instance.

Références

[1] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *Optimization by Simulated Annealing*, Science, vol. 220, n° 4598, pp. 671–680, 1983.

[2] N. Mladenovic, P. Hansen, *Variable Neighborhood Search*, Computers & Operations Research, vol. 24, n° 11, pp. 1097–1100, 1997.

[3] P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts : Towards Memetic Algorithms*, Caltech Concurrent Computation Program, Report 826, 1989.

[4] C. A. Coello Coello, *Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms : A Survey of the State of the Art*, Computer Methods in Applied Mechanics and Engineering, vol. 191, n° 11–12, pp. 1245–1287, 2002.

[5] J. W. Choi, R. E. Curry, G. H. Elkaim, *Smooth Path Generation Based on Bezier Curves for Autonomous Vehicles*, Proc. World Congress on Engineering and Computer Science, 2008.

[6] A. Blansche, *Metaheuristiques – Cours 1 a 5*, Universite de Lorraine, Master 1 Informatique, 2025/2026.