Le réveil des robots: étude sur le pire temps de réveil possible

Léo Frémery Cyril Gavoille Clément legrand-duchesne

27 juillet 2023

Table des matières

1 Introduction

1.1 Présentation du problème

Dans le cadre de ce problème, on dispose de n robots endormis et un robot éveillé sur un plan. Le robot éveillé va chercher à les autres robots avec l'aide des robots qu'il aura pu réveiller sur sa route.

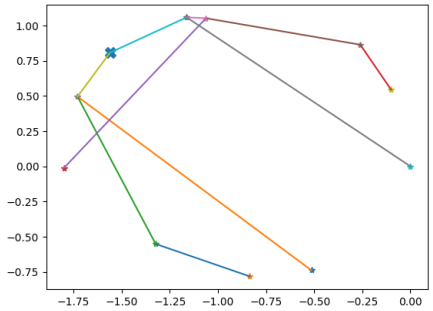


FIGURE 1 – Exemple de réveil

Sur cette figure, on commence par deux robots éveillé au niveau de la croix qui vont chacun réveiller un robot qui à leurs tours vont aller en réveiller d'autres selon les segments de couleurs de la figure.

Dans le cas général, trouver le chemin réveillant tous les robots endormis le plus vite possible est un problème NP-dur. Le meilleur algorithme existant, traçant le chemin des robots quelque soit l'emplacement des robots consiste en une programmation dynamique avec une complexité de $\frac{3^n}{\sqrt{n}}$ où n est le nombre de robots endormis.

On peut aussi s'intéresser à la pire configuration possible pour un nombre de robots endormis donnés afin de majorer le temps de réveil. On appelle la borne du temps de réveil pour n robots α_n. Cette borne est cependant très compliquée à calculer. On connaît la valeur pour $n = 1,2,3$ et 4 mais après, les valeurs sont inconnues. Clément: Il fadra qu'on revoit l'introduction, mais cela peut attendre le reste

1.2 Formalisation

1.2.1 problème du réveil des robots

Dans ce rapport, nous nous travaillerons sur le plan muni de la distance euclidienne (voir [] pour d'autres métriques). Le robot réveillé débute en $(0, 0)$ et une instance du problème consiste en n points représentant chacun un robot endormi. Quitte a renormaliser, on supposera que les robots sont disposés dans le disque de rayon 1 et de centre $(0, 0)$. On représente une solution au problème du réveil des robots par un arbre binaire couvrant tous les sommets, enraciné en $(0, 0)$. En effet, en chaque sommet un robot entre et réveille un autre robot : on dispose alors de deux robots à déplacer si on le souhaite, formant ainsi un arbre binaire dont la racine est le premier robot éveillé.

En pondérant les arêtes de notre arbre par la longueur de l'arête, on définit la profondeur de l'arbre comme étant le maximum de la somme des poids des arêtes depuis la racine jusqu'à une des feuilles de l'arbre. Le temps de réveil d'un arbre T est exactement la profondeur de celui-ci, on le notera $\mathrm{tr}(T)$. L'objectif étant de réveiller les robots le plus rapidement possible, le temps de réveil optimal est définit comme

$$\min_{\substack{T \text{ arbre binaire couvrant} \\ \text{enraciné en } (0,0)}} \mathrm{tr}(T, x_1, \ldots, x_n).$$

À nombre de robots n fixé, le "pire" cas est donc

$$\alpha_n := \max_{x_1,\dots,x_n \in \mathbb{R}^2} \min_T \operatorname{tr}(T, x_1, \dots, x_n)$$

Il est assez facile de prouver que $\alpha_1 = 1$ et $\alpha_2 = \alpha_3 = 3$. Pour $n = 4$, on peut prouver ([]) que $\alpha_4 = 1 + 2\sqrt{2}$. Gavoille, (lister les autres) ont prouvé dans [] que $\alpha_n \leqslant 3 + \frac{8\pi}{\sqrt{n}}$. Il est possible de faire mieux en pratique mais la borne ferait intervenir une constante plus compliquée que 8π.

Dans la suite, nous utiliserons régulièrement le résultat suivant dû à Gavoille (et...) :

Théorème 1.1 ([]). *Il faut au plus $r(1 + \operatorname{chord}(\theta))$ pour réveiller tous les robots dans un cône de rayon r et d'angle θ, à partir d'un robot éveillé initialement placé au centre du cône.*

2 Alpha 5

Dans cette section, nous montrerons le résultat suivant, obtenu pendant mon stage avec Cyril Gavoille.

Théorème 2.1.

$$\alpha_5 < 3.603564775$$

Dans la suite on appellera P l'ensemble des points représentant les robots endormis et on considérera γ la fonction qui à un ensemble de points associe le temps de réveil du problème associé. Pour cela, on prouvera le résultat suivant.

Lemme 2.2. *Pour tout P de taille 5, pour tout $\alpha \in \left]0, \frac{\pi}{2}\right[$, on a*

$$\gamma(P) \leqslant \max\left(3 + \operatorname{chord}(\alpha), 1 + 2\operatorname{chord}\left(\frac{\pi}{2} - \frac{\alpha}{4}\right)\right)$$

où chord *représente la longueur d'un arc de cercle de rayon 1 et d'angle θ. On a donc* $\operatorname{chord}(\theta) = 2\sin(\theta/2)$

Preuve de 2.1 en supposant Lemme 2.2. $\alpha \mapsto 3 + \operatorname{chord}(\alpha)$ étant croissante $\alpha \mapsto 1 + 2\operatorname{chord}(\frac{\pi}{2} - \frac{\alpha}{4})$ étant décroissante, le minimum des deux fonctions est atteint en leur croisement qui se produit en une abscisse d'environ 0.6131233066 et dont la valeur exacte n'est pas facilement calculable. Ainsi on a

$\forall P, \gamma(P) < 3.603564775$

Ce qui montre que $\alpha_5 < 3.603564775$

□

Lemme 2.3. *Pour tout n et x_1, x_2, ..., x_n des points sur un arc de demi-cercle dans l'ordre. Alors le chemin x_1 - x_2 - ... - x_n est le plus long lorsque les points sont uniformément répartis sur l' arc de demi-cercle.*

Démonstration. On commence par identifier les points par l'angle qui les sépare d'une des deux extrémités, obtenant ainsi une suite (θ_k) identifiant nos points. Dès lors, la longueur du chemin est :

$$2\sum_{k=1}^{n-1} \sin\left(\frac{\theta_{k+1} - \theta_k}{2}\right)$$

Or $\frac{\theta_{k+1}-\theta_k}{2} \in]0, \frac{\pi}{2}[$

Nous sommes donc sur un intervalle où sin est concave. Ainsi égaliser les $\frac{\theta_{k+1}-\theta_k}{2}$ conduit à la somme maximale et donc la longueur maximale du chemin. Cela implique alors

que la distance entre tous les points est la même et vu qu'il est toujours préférable d'avoir des points aux extremités, on a donc que le pire cas est d'avoir les points uniformément répartis sur l'arc de demi-cercle.

□

Dans la suite, on utilisera la notation $\gamma(T, P)$ pour donner le temps de réveil de P suivant l'arbre de réveil T. On a alors $\forall P, \gamma(P) = \min_T(\gamma(T, P))$ et donc $\forall P, \forall T, \gamma(P) \leqslant \gamma(T, P)$

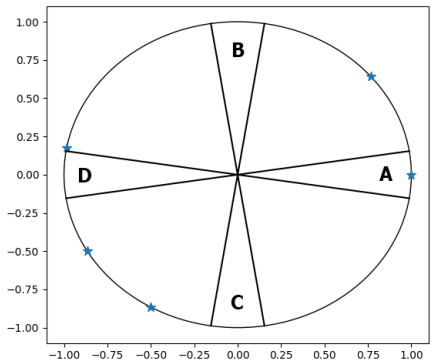
2.1 Description des Cas

La preuve fonctionne en trouvant pour chaque cas un arbre dont la longueur est au pire plus petite ou égale à la borne souhaitée.

- 2.2 Enveloppe convexe de taille 5
  - 2.2 Deux points dans un cône d'angle α en particulier, cela traite du cas où il existe 2 points dans A, B, C ou D.
  - 2.2 $|D| = 0$
  - 2.2 $|B| = |C| = 0$ (et $|D| = |A| = 1$) sans perte de généralité, on suppose que le point de D appartient au demi-cercle du haut
    - 2.2 le deuxième point du haut est entre A et B il reste à décider de la position des points du bas
      - 2.2 1 point du bas entre A et C ainsi que le dernier entre C et D.
      - 2.2 2 points entre C et D
      - 2.2 2 points entre A et C
    - 2.2 le deuxième point du haut est entre B et D de même il reste à décider de la position des points du bas
      - 2.2 2 points entre A et C
      - 2.2 2 points entre C et D
      - 2.2 1 point dans chacune des deux zones
  - 2.2 $|A| = |B| = |C| = |D| = 1$
    - 2.2 si le 5 ème point est dans un cône diagonal d'angle α
    - 2.2 sinon
  - 2.2 $|A| = |D| = 1$, $|C| = 0$ et $|D| = 1$ par symétrie, le cas $|C| = 1$, $|D| = 0$ rentre aussi dans cette catégorie, concluant la disjonction présente
    - 2.2 le point dans D appartient au demi-cercle du haut
      - 2.2 si on a en bas un point de chaque côté
      - 2.2 si les deux point du bas sont entre C et D
      - 2.2 si les deux points du bas sont entre A et C
    - 2.2 le point dans D appartient au demi-cercle du bas on a donc 4 cas selon le positionnement du point en haut et en bas
      - 2.2 Point du haut entre A et B et point du bas entre A et C
      - 2.2 Point du haut entre A et B et point du bas entre C et D
      - 2.2 Point du haut entre B et D et point du bas entre C et D
      - 2.2 Point du haut entre B et D et point du bas entre A et C
- 2.3 Enveloppe convexe de taille 3 et 4

2.2 Enveloppe convexe de taille 5

Soit $\alpha > 0$ et P un ensemble de 5 points Tout d'abord on tournera notre figure en sélectionnant un point tel que son axe coupe le cercle en deux demi-cercles contenant chacun 2 points. Un tel point existe toujours.



On sépare ensuite le cercle en différents cônes d'angle α nommés A, B, C, et D comme sur la figure 2. C'est en effet ce α qui apparaîtra dans la formule finale. On notera $|X|$ le nombre de point dans le cône X.

1er cas

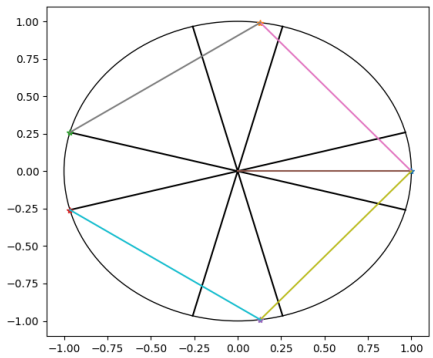
Supposons qu'il existe deux points inclus dans un cône d'angle α

Dans ce cas, en réveillant les deux robots dans le cône avec celui initial, on peut aller réveiller les trois derniers où qu'ils soient dans le cercle. Réveiller les robots dans un cône d'angle α prend au plus un temps $1 + \text{chord}(\alpha)$. On peut donc réveiller tout le monde avec $\gamma(P) \leqslant 1 + \text{chord}(\alpha) + 2 = 3 + \text{chord}(\alpha)$

2ème cas

$|D| = 0$

On prend T l'arbre commençant au noeud de A et explorant de part et d'autre de son axe en commençant par le noeud le plus proche. Soit P' l'ensemble de point sur le cercle où les deux points des demi cercles sont répartis entre A et D. On a $\gamma(P) \leqslant \gamma(T, P) \leqslant \gamma(T, P')$



En effet, d'après 2.3 le pire cas pour P est d'avoir un point le plus loin possible de A et d'avoir des arcs égaux, positionnant donc le dernier point du demi-cercle pile entre le point de A et celui collant D. On a alors

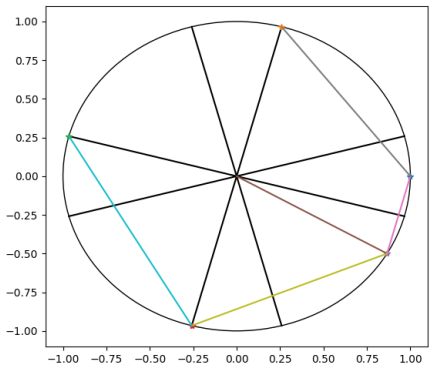
$\gamma(P) \leqslant \gamma(T, P') = 1 + 2\,\text{chord}(\frac{\pi}{2} - \frac{\alpha}{4})$

3ème cas

$|B| = |C| = 0$ ($|A| = 1$ et $|D| = 1$) Par symétrie, on peut considérer que le point dans D appartient au demi-cercle du haut et donc que l'on a un seul point en haut qui ne soit pas dans D que l'on nommera b.

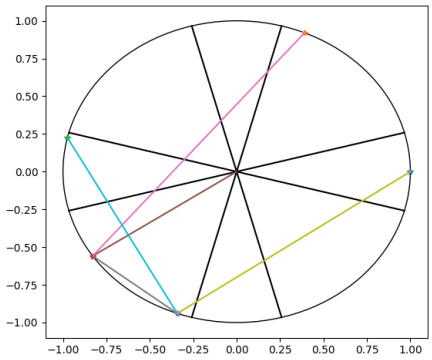
— Si b est entre A et B

— si b entre A et B et il y a en bas 1 point de chaque côté
on choisit pour T l'arbre commençant en bas à gauche et séparant le travail des robots en 2 comme avant. On prend ensuite le pire cas possible pour ce T.



En oubliant pas que l'angle entre a et le point en bas à gauche est d'au moins α on peut donc appliquer le lemme 2.3 afin de conclure $\gamma(P) \leqslant 1+2\,\mathrm{chord}(\frac{\pi}{2}-\frac{\alpha}{4})$

— si b entre A et B et deux points entre C et D l'arbre est le suivant :

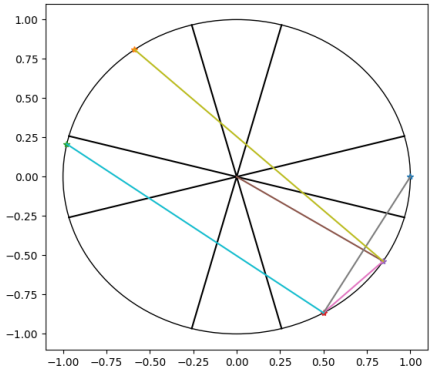


Ici on néglige la branche rose car elle a une longueur totale de 3 (ce sera aussi fait dans les cas suivants) La longueur de la branche bleue est ici $\leqslant 1+\mathrm{chord}\left(\frac{\pi}{2}-\alpha\right)+\mathrm{chord}(\frac{\pi}{2}) \leqslant 1+2\,\mathrm{chord}(\frac{\pi}{2}-\frac{\alpha}{4})$ Et pour la dernière branche, on applique 2.3 pour obtenir cette borne exacte
$\Rightarrow \gamma(P) \leqslant 1+2\,\mathrm{chord}(\frac{\pi}{2}-\frac{\alpha}{4})$

— Si b entre A et B et deux points entre A et C on applique le même arbre

— Si b est entre C et D

— Le seul cas difficile est celui où les deux points du bas sont entre A et C. suivant l'arbre suivant :



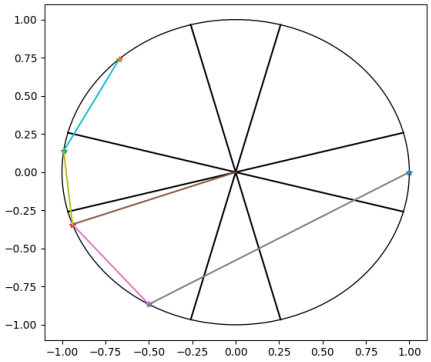
La branche bleue ici se résout encore une fois par 2.3 La branche jaune est négligée et la branche violette permet le calcul de son pire cas en maximisant

la longueur des cordes en respectant les contraintes angulaires. (en particulier que l'angle entre deux points doit être d'au moins α)

on obtient alors

$$\gamma(P) \leqslant \max\left(1+2\,\mathrm{chord}\left(\frac{\pi}{2}-\frac{\alpha}{4}\right), 1+\mathrm{chord}\left(\frac{\pi}{2}-\frac{3\alpha}{2}\right)+\mathrm{chord}\left(\frac{\pi}{2}-\frac{\alpha}{2}\right)\right)$$
$$\leqslant 1+2\,\mathrm{chord}\left(\frac{\pi}{2}-\frac{\alpha}{4}\right)$$

— Si l'on a 2 points entre C et D, alors on peut effectuer l'arbre suivant



On peut alors comme d'habitude utiliser 2.3 pour conclure

— Pour dernier cas, nous avons 1 point de part et d'autre de la vertical en bas alors par un chemin similaire au point précédent :

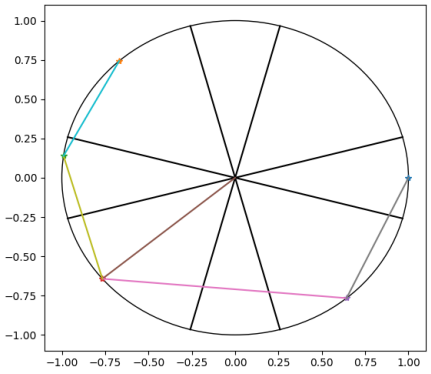


FIGURE 2 – Résolution 3e cas n5

Comme la dernière fois 2.3 nous permet de conclure

4eme cas

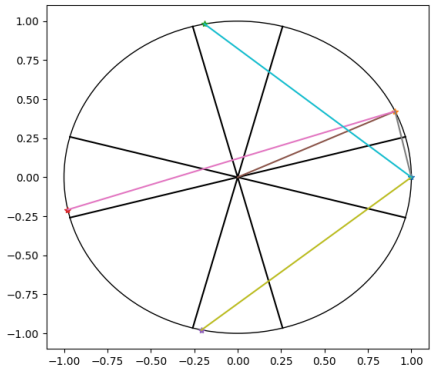
$|A| = |B| = |C| = |D| = 1$

il reste donc un 5eme point à placer qui se trouve dans un des cônes diagonaux.

— si il est dans un cône d'angle α au centre du cône diagonal.

on a alors le chemin standard commençant par le 5ème point qui nous donne par 2.3 : $\gamma(P) \leqslant 1+2\,\mathrm{chord}(\frac{\pi}{2}-\frac{\alpha}{4})$

— si ca n'est pas le cas alors on prend l'arbre commençant en ce 5 ème point mais allant vers le point le plus proche et le plus éloigné ainsi



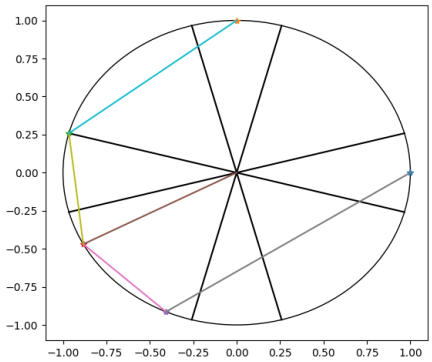
La branche rose est négligée la branche bleue est a sa première partie bornée par une corde de taille $\frac{\pi}{4}$ car le point du milieu est en dessous de la ligne diagonal d'un angle au moins $\frac{\alpha}{2}$ et la deuxième partie par $\frac{\pi}{2} + \alpha$ la branche jaune est symétrique à la branche bleue.

on a donc $\gamma(P) \leqslant 1 + \text{chord}(\frac{\pi}{4}) + \text{chord}(\frac{\pi}{2} + \alpha) \leqslant \max(3 + \text{chord}(\alpha), 1 + 2\,\text{chord}(\frac{\pi}{2} - \frac{\alpha}{4}))$

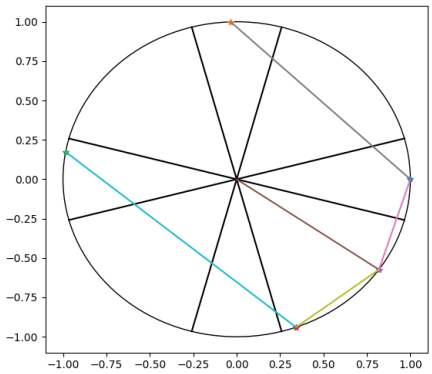
5ème cas

$|B| = |A| = |D| = 1, |C| = 0$ à noter que par symétrie, ca s'applique pour $|B| = 0$ et $|C| = 1$

— Si le point dans D appartient au demi-cercle du haut :
  — Si en bas il y a un point de chaque côté, on choisit le point en bas qui se trouve du côté de B et on termine de façon standard. On a alors $\gamma(P) \leqslant 1+2\,\text{chord}(\frac{\pi}{2}-\frac{\alpha}{4})$
  — si en bas on a les deux points du même côté, on commence par celui au milieu des deux points qui le collent pour obtenir le même résultat. Dans le cas où les deux points sont entre C et D on a donc

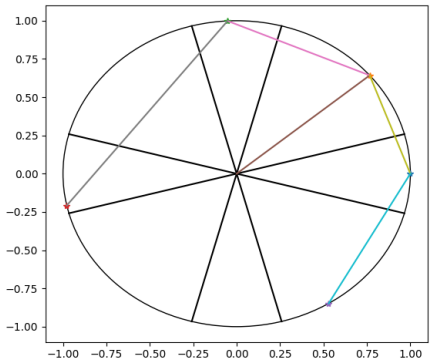


Nous sommes ici encore en capacité d'utiliser 2.3
Dans le cas où les deux sont entre A et C on a



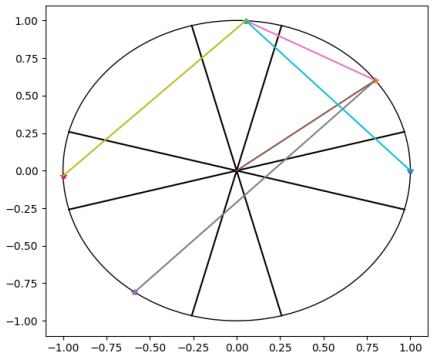
grâce à l'écart d'angle entre les points, on a bien encore une fois les conditions d'applications de 2.3

— Si le point dans D appartient au demi-cercle du bas, on a 4 cas
  — si point libre du haut entre A et B et point du bas entre A et C



en oubliant pas que le point de départ est forcément éloigné d'un angle α de a on a bien le résultat par 2.3

  — si point libre du haut entre A et B et point du bas entre C et D alors cela dépend de b. Si b est sur le côté droit, on commence effectue cet arbre :

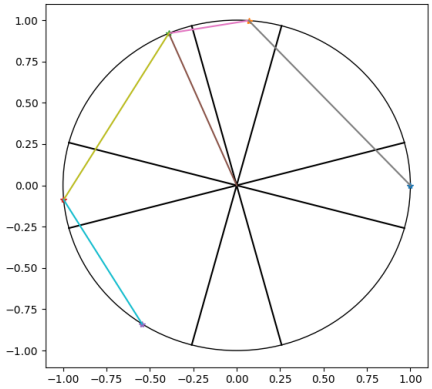


on a alors

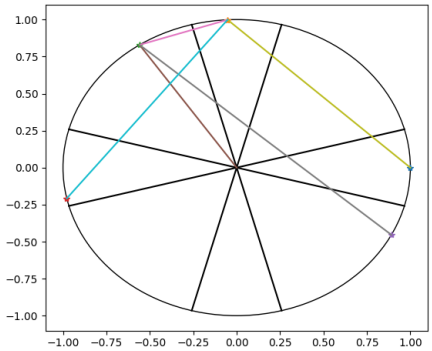
$\gamma(P) \leqslant \max\left(1 + 2\,\mathrm{chord}\left(\frac{\pi}{2} - \frac{\alpha}{4}\right), 1 + \mathrm{chord}\left(\frac{\pi}{2} - \alpha\right) + \mathrm{chord}\left(\frac{\pi}{2}\right)\right) \leqslant 1 + 2\,\mathrm{chord}\left(\frac{\pi}{2} - \frac{\alpha}{4}\right)$

si maintenant b est sur le côté gauche, on peut commencer par b. On effectuant un arbre classique, le côté gauche a la bonne borne et le côté droit est facile

— si point libre du haut entre B et D et point libre du bas entre C et D on a l'arbre suivant.



— si point libre du haut entre B et D et point libre du bas entre A et C cela dépend à nouveau de b. Si b à droite on a (en allant vers le point le plus proche) :

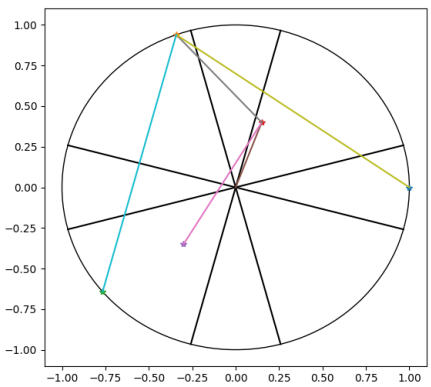


en effet, on obtient alors en allant vers le plus proche que la longueur de la branche qui effectue le croisement est plus petite que $1 + \text{chord}\left(\frac{\frac{\pi}{2}+\frac{\alpha}{2}}{2}\right) + \text{chord}\left(\frac{\pi}{4} + \frac{\alpha}{4}\right)$ or cette valeur est inférieure à $\max\left(3 + \text{chord}\left(\alpha\right), 1 + 2\,\text{chord}\left(\frac{\pi}{2} - \frac{\alpha}{4}\right)\right)$ on a alors le résultat souhaité.

cependant si b est à gauche on peut alors commencer par b pour finir le problème comme la dernière fois.

2.3 Enveloppe convexe de taille 4 ou 3

Désormais, il faut terminer avec les cas où l'enveloppe convexe est de taille 3 ou 4. Heureusement, ces cas sont plus simples. Si l'enveloppe convexe est de taille 3, on va vers le point le plus proche du centre qui soit à l'intérieur de l'enveloppe convexe, ensuite un des deux robots ira chercher le deuxième point à l'intérieur tandis que le deuxième ira réveiller le point le plus proche du triangle avant d'aller sur les deux autres.



En effet, le pire cas est si le point que l'on rejoint en premier est projeté sur l'enveloppe convexe orthogonalement ce qui nous donne $\exists\theta, \gamma(P) \leqslant \cos(\theta) + \sin(\theta) + 2 \leqslant \sqrt{2} + 2 \leqslant \max(3 + \text{chord}(\alpha), 1 + 2\,\text{chord}(\frac{\pi}{2} - \frac{\alpha}{4}))$

Note : En réalité, cela fonctionne quelque soit l'emplacement du deuxième point à l'intérieur, et donc également si il n'est pas dans l'enveloppe connexe, et il est donc possible de traiter par exactement la même méthode le cas où l'enveloppe convexe est de taille 4.

2.4 conclusion de la preuve

Cela conclut la preuve, trouvant ainsi une borne pour α_5 montrant également que $\alpha_5 < \alpha_4$. On remarquera que plus on fait de disjonctions de cas, plus il est possible d'affiner le choix de l'arbre selon les situations, permettant ainsi d'obtenir une meilleure borne.

3 Implémentation C++

Il n'est pas clair quelle est la forme de l'ensemble de point qui mènera au pire temps de réveil. Cependant, on peut conjecturer que pour le cas pair, il s'agirait d'une répartition uniforme des points sur le cercle et c'est dans ce but que l'on a étudié selon divers algorithmes le temps de réveil du cas "uniforme" à n robots que l'on appelera β_n.

En utilisant nos algorithmes afin de tracer des approximations de β_n, on a pu émettre une conjecture indiquant que β_{2n} serait décroissante. Cela serait un pas en avant dans la recherche pour montrer que α_4 est bien le maximum strict.

Pour tenter de s'assurer de la véracité de ces formules, il est d'abord intéressant de la tester pour divers valeurs de n. Mon travail a donc été ici la réimplémentation en C++ d'algorithmes python afin de les accélérer et en particulier pour le cas où les points sont répartis uniformément dans le cercle.

3.1 Premier algorithme "Segment"

principe de l'algorithme il se positionne sur un point de la forme convexe, et choisi un autre point qui coupera alors le polygone en deux parties, On envoi alors un robot pour chacune de ces parties qui s'occuperont récursivement de leurs propres polygones. Il est important de noter que l'on utilise un cache pour introduire de la programmation dynamique et réduire la complexité.

Segment est un algorithme qui a été créé pour travailler sur le cas général dans l'espoir d'obtenir un bon algorithme polynomial pour le résoudre. Le problème était NP-dur, cet algorithme ne permet en effet pas de résoudre le cas général et manque des solutions. Cyril Gavoille a tout de même prouvé qu'il existe un ordre d'entrée des points permettant à segment de donner la solution optimale. Cependant, étant donné qu'il existe $n!$ solutions, cela n'est qu'une observation. Dans le cas Convexe, on aurait pu espérer Cependant qu'en donnant les solutions dans l'ordre standard on puisse obtenir une solution optimale. En effet, Le cas où les points sont en position convexe laisse une question ouverte sur son appartenance à P.

il est important d'observer que dans le cas convexe avec l'ordre dans le sens des aiguilles d'une montre de la forme convexe, la solution obtenue présente des arêtes qui ne s'intersectent pas, on dit que cette solution est planaire.

Dans le cadre d'un cercle où les points sont répartis uniformément, une optimisation excellente consiste à introduire une certaine subjectivité, un duo de robot sur un point n'aura pour environnement de travail qu'un segment défini par sa longueur à gauche et à droite de lui. Il n'existe alors que n^2 possibilités. On obtient alors un algorithme polynomial, rapide pour résoudre les β_n.

Enfin, c'est ce qu'il aurait été possible de dire si segment permettait de donner les solutions optimales. Malheureusement, cela n'a pas été prouvé et dans le cas plus général convexe, on observe des solutions optimales non planaires et donc automatiquement, segment n'est pas optimal, en voici un exemple :

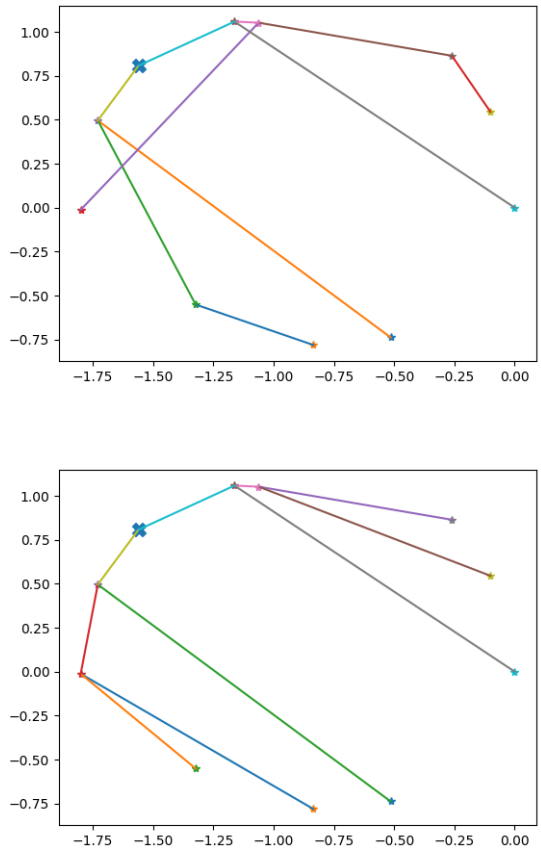


FIGURE 3 – Non optimalité de Segment

On a à gauche la solution optimale présentant un croisement, exploitant que la branche à gauche est plus courte que celle de droite tandis que sur l'image de droite, on trouve la meilleure solution planaire possible, soit celle donnée par segment. On voit donc bien que segment ne résout pas complètement le cas convexe mais l'approxime.

Ma contribution sur ce sujet aura été l'implémentation et l'optimisation en C++ qui aura ainsi permis d'aller explorer la position uniforme jusqu'à $n = 1000$ sans prendre trop de temps. Il reste encore ouvert si Segment du cas convexe permet de résoudre le cas uniforme et cela forme une nouvelle conjecture.

3.2 Brute force

Comme son nom l'indique, cet algorithme cherche à tester toutes les solutions possibles afin de trouver la meilleure possible. En utilisant un cache, il est possible de montrer que cet algorithme est dans le cas général en $\frac{3^n}{\sqrt{n}}$. Avant mon arrivée, pour calculer le temps optimal pour le cas circulaire à $n = 17$ points, il fallait à un bon ordinateur 5 minutes et 17 secondes rendant très compliqué voir impossible d'aller plus loin et donc de vérifier nos conjectures qui ne deviennent ambigües qu'à n grand.

Dans l'objectif de pouvoir aller plus loin dans l'analyse des β_n. J'ai également réimplémenter l'algorithme brute force en C++ permettant alors de calculer le cas $n = 17$ en 1 minute et 18 seconde, un progrès significatif surtout que ce temps est obtenu sur un pc bien moins bon. Ironiquement, la plus grand optimisation aura été de rajouter "-O3" à la commande de compilation pour laisser le compilateur tout optimiser à sa façon, utilisant des fonctions avancées du processeur. On descend alors à 18 secondes. Pour que cela soit possible, que le compilateur puisse optimiser autant, il faut noter qu'une grande amélioration aura été de représenter les ensembles de points présents en grande quantité dans le code par une classe spéciale basée sur un entier 64 bit où un 1 en position i signifie que i est dans l'ensemble. Un tel objet fonctionne bien car on sait que brute force ne pourra jamais atteindre $n = 64$ et donc qu'on aura jamais plus de 64 points, soit le nombre de bit dans un entier 64 bits. Par ce biais, il a été possible d'optimiser les opérations de création des ensembles, et d'en réduire le coût mais surtout de simplifier pour l'ordinateur l'étape suivante...

En effet, jusque là, les optimisations marchent dans le cas général, et n'utilisent absolument pas la structure de polygone régulier du problème. Pour représenter parfaitement le polygone régulier, on peut penser aux isométries, les opérations que j'ai nommé "miroir" de symétrie axiale et "shift" de rotation centrale. l'ensemble des rotations et d'une seule de ces symétries axiale engendre tout le groupe des isométries du polygone régulier. Si l'on cherche à traduire cela avec notre groupe de bits représentant un ensemble, cela revient à dire qu'en déplacant mes bits de droite à gauche et inversement (ainsi que le point de départ des robots) donne exactement la même solution à "shift" pres. Alors, pour représenter cela, on fait en sorte le point de départ des robots soit toujours 0 afin que toutes les so-

lutions qui soient un décalage d'une autre se reconnaissent comme étant les mêmes. Cela mène à une solution nommée "shift invariant" réduisant alors le temps d'exécution pour $n = 17$ à 9.5 secondes. Il est à noter que la rotation des bits se fait très bien avec des opérations binaires simples et donc est une opération au goût du compilateur. Il reste alors la dernière optimisation, celle du miroir. Sachant avec l'optimisation précedente que le point de départ des robots est en 0, l'opération miroir revient à effectuer un effet miroir sur les bits sauf le bit en 0 qui reste le même. Malheureusement, il n'existe aucune opération binaire simple permettant d'effectuer un effet miroir sur les bits et j'ai donc du utiliser un "bit hack" qui effectue un algorithme court pour résoudre le problème en diviser pour régner.

```
short bits = 64;
    unsigned long long mask = ~(0llu);

    while (bits >>= 1) {
        mask ^= mask << (bits);
        n = (n & ~mask) >> bits | (n & mask) << bits;
    }
```

Cet algorithme fonctionne en inversant d'abord les groupes de 32 bits, puis les groupes de 16 bits puis ... à l'aide d'un masque.

Une fois cela fait, pour faire en sorte que les opérations soient invariantes par "mirror", je trompe le cache en lui indiquant de considérer égaux des entiers non seulement si leurs ensembles sont égaux mais aussi si le miroir est égal à l'autre entier. Et cela descend le temps d'execution pour $n = 17$ à 4.5 secondes. Grâce à ces optimisations on sera descendu de 5 minutes et 17 secondes à 4.5 secondes.

3.3 Mon propre algorithme

En arrivant à mon stage, je suis arrivé avec un algorithme en tête cherchant à résoudre le problème dans le cas convexe, un algorithme se situant entre le brute force et segment. Malheureusement, il n'est pas polynomial, et il ne résout pas non plus le problème. Cette approche reste sur l'approche consistant à choisir des sous ensembles pour nos robots sauf que là où segment coupe en deux segments et brute force choisi tous les sous groupes de travail de possible, ce nouvel algorithme sélectionne des ensembles de points dont les enveloppes convexes sont disjointes. Cela revient à tracer une droite sur un cercle pour couper le cercle en deux morceaux. Segment est ainsi le cas particulier où le point de départ est forcement sur la droite de coupure et présente donc n possibilités là où mon algorithme présente n^2 explorations récursives. L'analyse de la complexité d'un tel

algorithme toujours avec cache est extrêmement compliquée et c'est seulement expérimentalement qu'on observe la nature exponentielle du nombre d'étape. Cet algorithme permet de trouver des solutions non planaires ce qui n'était pas le cas de segment. Malheureusement, 17 permet de conclure à la non optimalité de cet algorithme. Pour trouver ce contre-exemple, il a d'ailleurs fallu que j'implémente l'algorithme et que je l'optimise au même titre que l'algorithme de brute force afin de le faire tourner jusqu'à trouver un résultat où le brute force est meilleur que mon algorithme. En python, il me fallait 720 secondes pour un seul test alors qu'en c++, j'en faisais plusieurs par seconde grâce à une nouvelle implémentation. En dépit de ses défauts, un tel algorithme reste plus rapide que l'algorithme de brute force en effet, après y avoir ajouté exactement les nouvelles améliorations de brute force pour le cas uniforme, on obtient l'execution de cet algorithme pour $n = 17$ en 0.05 secondes ce qui permet enfin de viser des n plus grands. avec mon faible ordinateur, n = 32 n'était alors plus un rêve.

3.4 Conjectures

Ainsi, on possède des algorithmes plus rapides pour explorer les conjectures plus profondément. Premièrement, pourquoi cette valeur de n = 32 était importante. Il était observé par Cyril Gavoille qu'en utilisant segment on obtenait $\beta_{60} < \beta_{62}$ ce qui cassait la conjecture de décroissance de β_{2n}, on se demandait si alors en réalité, la conjecture était vraie, mais que les solutions dans le cas uniforme devenaient non planaires, chose qui était difficilement envisageable avant. C'est alors que l'on m'avait tâché d'atteindre avec des algorithmes un peu plus puissants cette valeur $n = 62$. En supposant la symétrie de part et d'autre du cercle, on pouvait essayer d'utiliser un puissant algorithme pour $n = 62$ avec 32 points et espérer d'obtenir une meilleure solution. Cependant, partir avec la moitié des points utilise moins les optimisations de cache que partir avec tous les points, faisant que le temps de calcul de $n = 32$ était au final bien plus court que celui de $n = 62$. En réalité, avec mon algorithme et en supposant la symétrie, je n'ai pas pu dépasser $n = 52$. Sans nouvelles pistes d'améliorations, je dû donc abandonner cet espoir d'atteindre $n = 62$.

Et c'est alors en voulant dessiner un graphe des β_{2n} (avec segment) pour le rapport de stage afin de justifier la conjecture que j'ai remarqué quelque chose... de surprenant.

On observe bel et bien le petit saut en $n = 62$ cependant... on observe deux pics avant cela, un pic en $n = 14$ et un faux pic en $n = 24$ qui en réalité ne casse pas la conjecture. Mais ce qui est important, c'est qu'on observe une exception bien avant $n = 62$! or pour $n = 14$ on peut aisément faire le calcul avec l'algorithme de brute force. Et l'algorithme de brute force

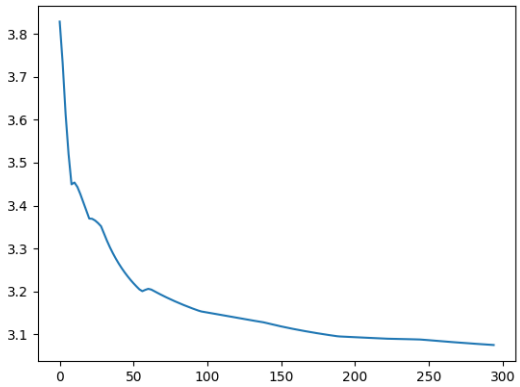


FIGURE 4 – β_{2n}

et segment coïncident pour $n \leqslant 17$ On alors bien que la conjecture était fausse. On pourrait alors imaginer de nouvelles conjectures, en particulier sur où sont les exceptions. Il a d'abord été proposé par mon encadrant des formules basées sur 4^n car 4 est un maximum, 16 est proche d'une exception et 64 est un maximum local. Cependant, n'ayant aucune autre exception pour $n \leqslant 300$ la série se casse. Peut être peut on alors se dire qu'il s'agissait d'exceptions et qu'il n'y en a plus après, et pourtant...

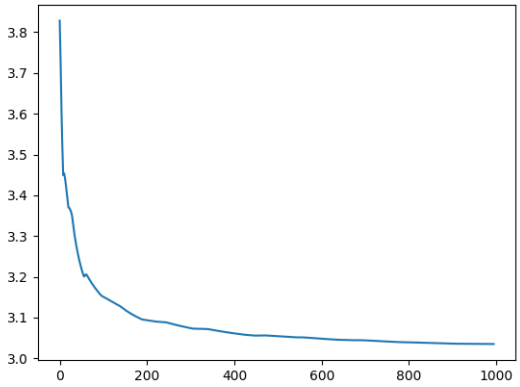


FIGURE 5 – β_{2n} jusqu'à $n = 500$

On observe rien de plus sur cette figure, elle semble lisse après les exceptions et prend désormais en compte les $n \leqslant 1000$ ce qui va bien plus loin, mais.. en cherchant avec python des maximums et des minimums, voici ce que l'on obtient :

```
local maximum:  4
local minimum:  12
local maximum:  14
local minimum:  60
local maximum:  64
local minimum:  450
```

```
local maximum:  474
local maximum:  476
local minimum:  550
local maximum:  560
local minimum:  678
local maximum:  688
local minimum:  802
local minimum:  804
local maximum:  806
```

On observe nos anciens pics mais on observe surtout de nouveaux pics qui ont l'air complètement chaotiques. Cet air chaotique est probablement donné par des solutions d'équations trigonométrique mais cela montre surtout qu'il n'y a pas d'espoir de trouver des solutions simples, que nos conjectures sont justes fausses et donc que ce que certains chercheurs s'acharnaient à vouloir montrer ne pouvaient être faits car cela n'était tout simplement pas vrai... Voici quelque part ma contribution dans le domaine des conjectures. Et tout cela grâce à des algorithmes optimisés comme possible.

4 Conclusion

Dans ce rapport, nous aurons alors pu obtenir un résultat en apparence mineur sur la série des alpha mais nous aurons également pu voir que des conjectures pourtant considérées comme étant vraies sont en réalité bien fausses rendant ainsi les conjectures précieuses et les petits résultats importants. Malgré tout, il faut continuer d'espérer que les grandes conjectures soient correctes et espérer pouvoir résoudre ces questions ouvertes telles que :

— α_4 est maximum strict ?
— α_{2^n} décroissante ?
— Pire configuration convexe et donc uniforme (cas paire) ?
— Résolution du cas convexe P ? NP-dur ?
— Résolution du cas uniforme dans P ?