Réveil des Robots

1 Alpha 5

Cette partie est une contribution personnelle dédiée à obtenir une borne de α_5 avec l'aide de Cyril Gavoille.

$\alpha_5 < 3.603565$

Afin de prouver cette borne, on effectue d'abord une disjonction de cas sur la taille de l'enveloppe convexe.

On prouvera:

$\forall P, \forall \alpha, \gamma(P) <= max(3 + chord(\alpha), 1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4}))$

où γ représente la longueur du chemin minimal pour un set de point P donné. et $chord$ représente la longueur d'un arc de cercle de rayon 1 et d'angle θ en particulier, $chord(\theta) = 2\sin(\theta/2)$

En effet, on remarque que $3 + chord(\alpha)$ est croissant et que $1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4})$ est décroissante, ainsi le minimum des deux fonctions est atteint en leur croisement qui se produit en une ordonnée $y < 3.603565$ et dont la valeur exacte n'est pas facilement calculable. Ainsi on a

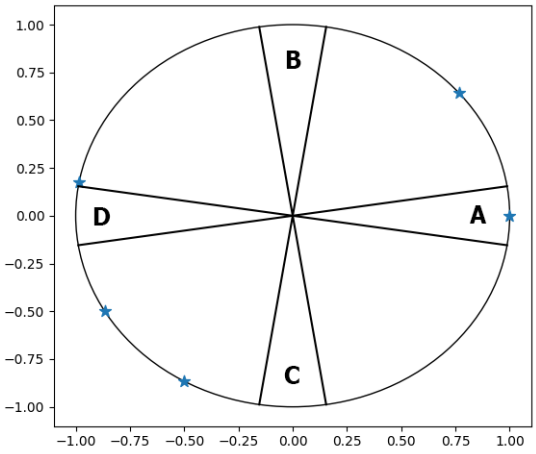
$\forall P, \gamma(P) < 3.603565$

Ce qui montre que $\alpha_5 < 3.603565$

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 $\forall P, \forall T, \gamma(P) <= \gamma(T, P)$

Enveloppe convexe de taille 5

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.



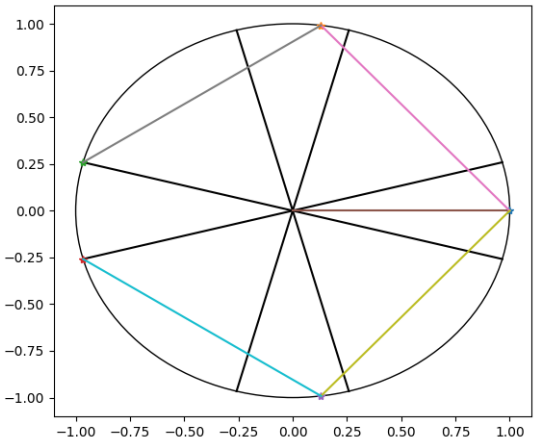
On sépare ensuite le cercle en différents cônes d'angle α nommés A, B, C, et D. C'est en effet ce α qui apparaîtra dans la formule finale. On notera $|X|$ le nombre de point dans le cône X et nous ferons des disjonctions de cas sur ces derniers.

Soit P un ensemble de point, on tourne le cercle comme prévu. Pour l'exemple, on positionnera les poitns sur le cercle mais les chemins utilisés marcheront quelque soit leur positionnement tant que l'enveloppe convexe est de la taille prévue.

1er cas: $\exists X, |X| \geq 2$ 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 $1 + chord(\alpha)$ ce qui est montré dans d'autres papiers de recherche. On peut donc réveiller tout le monde avec $\gamma(P) <= 1 + chord(\alpha) + 2 = 3 + chord(\alpha)$ Notons que cette preuve s'applique dès que deux points sont sur un cône de taille α commun. On peut donc supposer désormais que cette situation n'arrive plus.

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

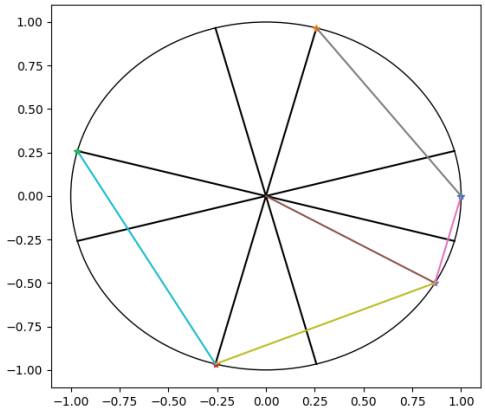
les deux points des demi cercles sont répartis entre A et D. On a $\gamma(P) \leq \gamma(T, P) \leq \gamma(T, P')$



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

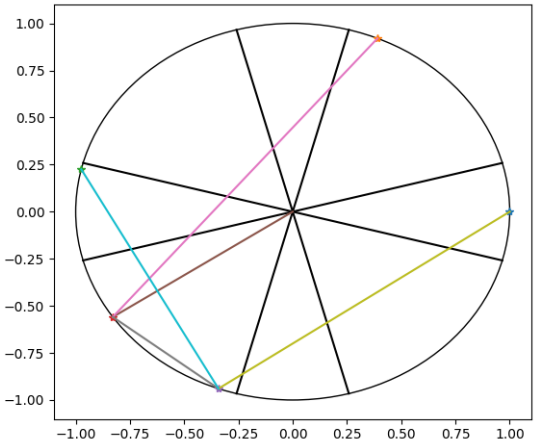
3ème cas: $|B| = |C| = 0$ ($|A| = 1$ et $|D| = 1$) Par symétrie, on peut considérer que le point dans D appartient au demi-cercle du haut et donc que l'on a un seul point en haut qui ne soit pas dans D que l'on nommera b.

-si b entre A et B et 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.

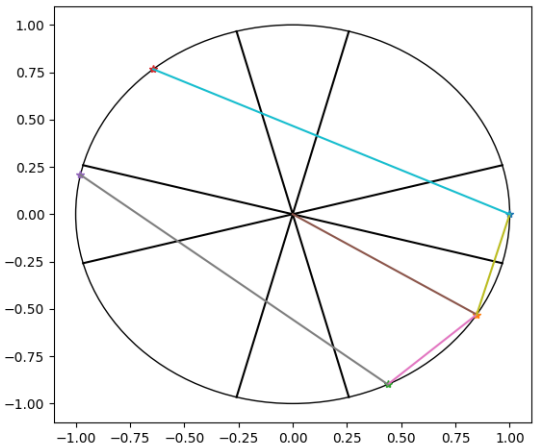


En oubliant pas que l'angle entre a et le point en bas à gauche est d'au moins α on a donc $\gamma(P) \leq 1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4})$

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



$=> \gamma(P) \leq 1+2chord(\frac{\pi}{2} - \frac{\alpha}{4})$ -par la suite, le seul sous cas difficile est celui où b entre C et D et les deux points du bas sont entre A et C. suivant l'arbre suivant:



on obtient alors $\gamma(P) \leq 1 + chord(\frac{\pi}{2} - 2\alpha) + chord(\frac{\pi}{2} + \alpha) \leq 1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4})$

4eme cas: $|A| = |B| = |C| = |D| = 1$

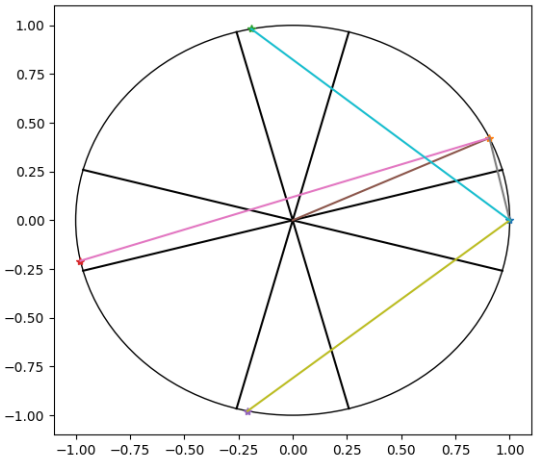
il reste donc un 5eme point à placer qui se trouve dans un des cônes diagonaux.

-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 $\gamma(P) \leq 1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4})$

-si ca n'est pas le cas alors on prend l'arbre suivant



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

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 $\gamma(P) \leq 1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4})$

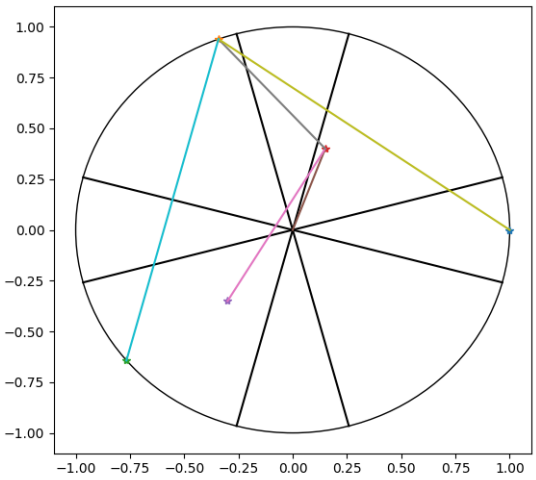
-si en bas on a les deux points du même côté, on commence par celui au milieu des deux points qui le collent pour obtenir le même résultat.

Si le point dans D appartient au demi-cercle du bas, alors commencer par b ou le point libre du haut solutionnera toujours le problème...

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 $\exists\theta, \gamma(P) \leq \cos(\theta) + \sin(\theta) + 2 \leq \sqrt{2} + 2 \leq max(3 + chord(\alpha), 1 + 2chord(\frac{\pi}{2} - \frac{\alpha}{4}))$

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.

Ainsi, on a montré le résultat voulu et par une analyse graphique, on obtient α tel que $\gamma(P)$ soit minimal. L'équation ne se résolvant pas facilement, on approxime et augmente la valeur pour obtenir $\alpha_5 < 3.603565$

2 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 qu'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" que l'on appelera β_n.

En utilisant nos algorithmes afin de tracer des approximations de β_n, on a pu émettre une conjecture disant que β_{2n} serait décroissante. Une telle conclusion serait très puissante puisque l'on peut aussi conjecturer que $\beta_{2n+1} \leq \beta_{2n}$ et donc que $\beta_4 = \alpha_4$ est bien le maximum global et de posséder une preuve de cela. Telle conclusion parait naturel et bien qu'il ne s'agisse que de conjectures, nous verrons plus tard que les conjectures dans ce problème sont très précieuses.

Pour essayer de se donner une idée de la potentielle vérité derrière ces conjectures, il est intéressant d'améliorer nos algorithmes précédemment implémentés en python afin d'obtenir des valeurs pour n plus grand.

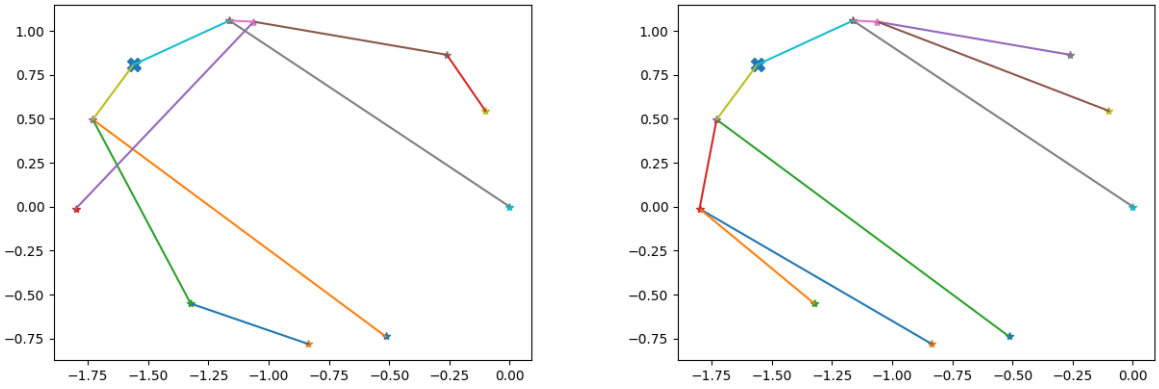
Premier algorithme "Segment"

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.

Tout d'abord, il est important d'observer que dans le cas convexe avec le bon ordre des points, la solution obtenue présente des arêtes qui ne s'intersectent pas, on dit que cette solution est planaire. Pour ce faire, cet algorithme 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.

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:



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.

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 solutions 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.

Mon propre algorithme

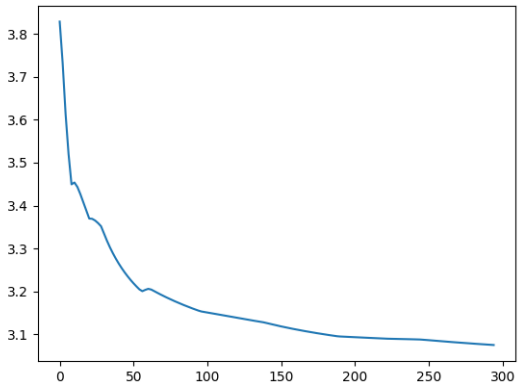
En arrivant à mon stage, je suis arrivé avec un algorithme en tête cherchant à résoudre le problème dans le cas convexe, un algorithme se situant entre le brute force et segment. Malheureusement, il est ni polynomial, et il ne résout pas non plus le problème. Cette approche reste sur l'approche consistant 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 selectionne 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 experimentalement 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, le même contre exemple que pour segment 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.

Conjectures

Ainsi, on possède des algorithmes plus rapides pour explorer les

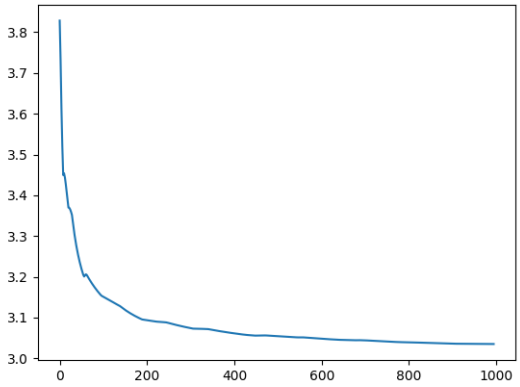
conjectures plus profondément. Premièrement, pourquoi cette valeur de n = 32 était importante. Il était observé par Cyril Gavoille qu'en utilisant segment on obtenait $\beta_{60} < \beta_{62}$ ce qui cassait la conjecture de décroissance de β_{2n}, on se demandait si alors en réalité, la conjecture était 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 et segment coincident pour $n \leq 17$ 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 $n \leq 300$ 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...



On observe rien de plus sur cette figure, elle semble lisse après les exceptions et prend désormais en compte les $n \leq 1000$ 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.