Rapport de stage assistant ingénieur

Du 07/06/2021 au 31/08/2021

GOURRET Yohann

Pavage & Contracteur par Intervalle

1 Lien Github

Voici le lien github du contracteur :

https ://github.com/pierleur/Resultat_stage_homeostatsie

2 Remerciements

J'aimerais tout d'abord remercier mon tuteur de stage, enseignant-chercheur au sein du LIRMM, M. TROMBETTONI Gilles, pour son accueil chaleureux et pour ce qu'il m'a permis d'apprendre au cours de ce stage. Je remercie également les doctorants RADWAN Verlain et DESOEUVRES Aurélien que j'ai pu assister dans leurs travaux respectifs. Enfin, de manière plus générale, j'aimerais remercier tous les membres du LIRMM et en particulier les membres de l'équipe COCONUT pour leur accueil.

Table des matières

3 Contexte.

3.1 LIRMM

J'ai réalisé mon stage au sein du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, le LIRMM. Cette unité de recherche dépend de l'Université de Montpellier et du Centre National de la Recherche Scientifique, le CNRS. J'ai travaillé au sein de l'équipe COCONUT qui fait partie du département d'Informatique.

3.2 Organisation du rapport

J'ai commencé à travailler avec RADWAN Verlain pour son pavage régulier par intervalle en programmant un projecteur et une visionneuse 3D. C'est ensuite que j'ai travaillé pour DESOEUVRES Aurélien pour un contracteur appliqué à ses problématiques. Ces deux éléments sont indépendants, nous commencerons dans ce rapport par ce qui a été mon travail le plus conséquent à savoir le contracteur avant de nous pencher sur le pavage par intervalle.

4 Contracteur pour problème d'homéostasie.

DESOEUVRES Aurélien est un doctorant en mathématiques appliqué à la biologie qui cherche à utiliser le calcul par intervalle pour des problèmes d'homéostasie. DESOEUVRES Aurélien a déjà apporté une solution à son problème. Vous pourrez trouver son algorithme dans l'article "Interval Constraint Satisfaction and Optimization for Biological Homeostasis and Multistationarity". Mon travail à été de tester une autre stratégie espéré plus efficace. Dans un premier temps, nous allons voir de quoi il s'agit et comment s'y inscrit le calcul par intervalle. Nous verrons ensuite la stratégie globale en posant certains termes, l'algorithme 3B-Cid qui est central pour notre résolution et enfin l'algorithme en tant que tel.

4.1 Problématique

Il s'agit à l'origine d'un problème pour la biologie : l'homéostasie. C'est une régulation biologico-chimique naturelle qui permet de maintenir certaines espèces, ou paramètre physique, clef pour le vivant aux concentrations, ou valeurs, adéquates nécessaires pour la santé des organismes. Par exemple, un mécanisme d'homéostasie peut être l'équilibre acido-basique, qui permet au corps humain de conserver un pH proche de 7.4. L'homéostasie est une résistance aux changements paramétriques.

L'homéostasie peut faire intervenir un grand nombre d'espèces chimiques qui interagissent entre elles. L'idée générale, dans le travail que j'ai fourni, est de déterminer en fonction de paramètres (sous forme d'un vecteur d'intervalle / boîtes) le vecteur d'intervalle / boîtes représentant les proportions des autres espèces du système chimique.

En effet, à l'équilibre chimique, les vecteurs h (vecteur des paramètres) et x (vecteur des variables) respectent un système d'équation S telles que S(h,x) = 0. Notre objectif est de trouver BoxO la boîte optimale, la boîte la plus petite telle que :

$$\forall h \in [h] \; \exists x \in \; BoxO \; tel \; que \; S(h,x) = 0$$

Pour bien en comprendre le sens, si homéostasie il y a, alors certains éléments de la BoxE devraient avoir un intervalle fin quand d'autres auront un intervalle large. Pour imager en leur prêtant conscience, certaines espèces vont se sacrifier en augmentant ou diminuant leurs concentrations pour permettre à d'autres de rester à des concentrations idéales.

4.2 Statégie globale

Trouver BoxO n'est pas envisageable, on cherche plutôt à trouver un BoxE (boîte extérieure) tel que :

$$BoxO \subset BoxE$$

Avec pour but de minimiser au maximum BoxE pour qu'il s'approche au plus près de BoxO. On définit aussi BoxI qui est une pseudo boîte intérieure telle que :

$$BoxI \subset BoxO$$

Habituellement dans la littérature, une boîte intérieure va désigner une boîte où tout point s'y trouvant est solution du système. Or ce n'est pas notre cas ici. Pour illustrer prenons le système suivant :

$$x^2 \leq 1$$

Dans ce cas BoxO = [-1, 1], BoxE pourrait être égal à [-1.5, 1.5] et BoxI égal à [-0.5, 0.5]. Dans ce cas :

$$\forall x \in BoxI,\ x^2 \leq 1$$

Très bien, sauf que dans notre cas, le système est constitué d'égalité et non d'inégalité. On aura donc plutôt :

$$x^2 = 1$$

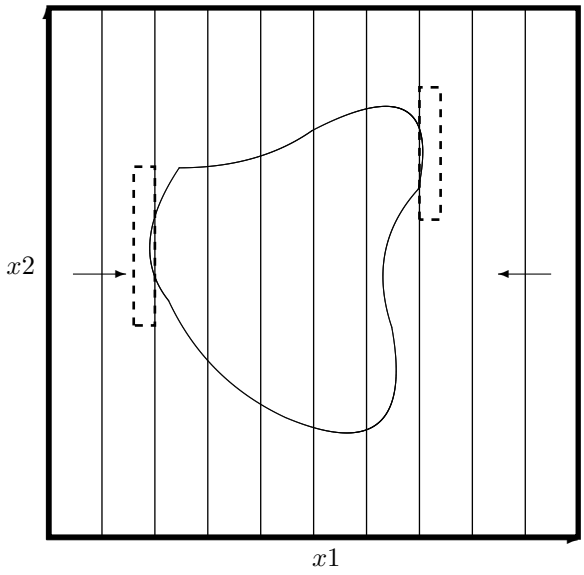
De ce fait tout x dans BoxI = [-0.5, 0.5] n'est pas forcément solution, BoxI n'est pas une boîte intérieure selon les critères habituels. BoxI est juste compris dans BoxO. Mais par abus de langage nous l'appellerons par la suite boîte intérieure.

La stratégie envisagée est de prendre une boîte extérieure initialement très grande, une boîte intérieure initialement vide, de diminuer la première et grossir la seconde. Le critère d'arrêt pourra alors être défini comme la différence entre les deux boîtes. Si la différence (ou en pratique plutôt un rapport) est en dessous d'un certain seuil, alors on s'arrête et l'on garde la boîte extérieure comme approximation de BoxO. De plus la connaissance à chaque étape de la boîte intérieure et extérieure permet de mieux savoir où chercher pour l'extension ou la contraction de l'un et l'autre.

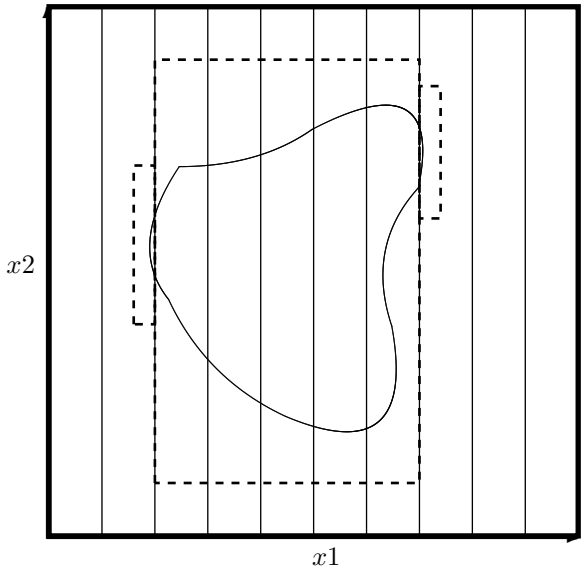
4.3 L'algorithme 3B-Cid :

3B Cid est un contracteur qui va reposer sur une découpe en tranche de la boîte à contracter. Soit une X de dimension n, on va réaliser une contraction par découpe sur chacune des n dimensions.

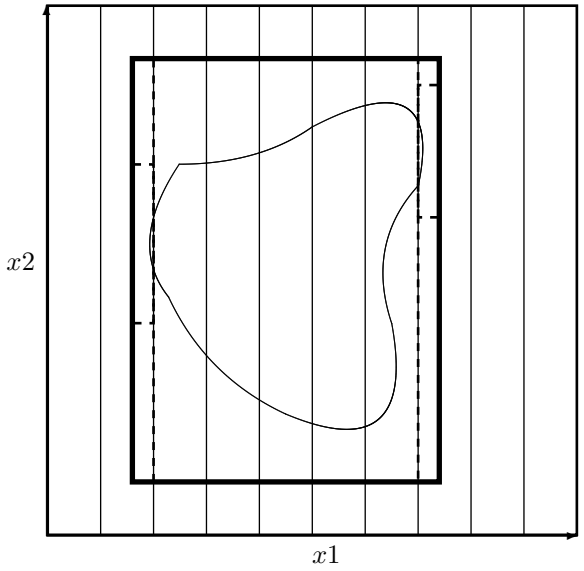
Voici comment fonctionne une découpe. Pour avoir une représentation graphique plus aisé prenons un exemple à n=2. Ci-dessous une boîte selon x1 et x2 que l'on cherche à contracter pour encadrer le patatoïde. On commence par découper la boîte selon x1 pour obtenir plusieurs sous-boîtes, ici pour l'exemple 10 sous-boîtes. Ensuite on contracte les sous-boîtes, par un contracteur annexe comme HC4, d'un côté de gauche à droite et de l'autre de droite à gauche jusqu'à trouver des résultats de contraction non vide :



L'étape suivante consiste à contracter le hull de toutes les sous-boîtes restantes :



Enfin, le hull des trois boîtes trouvées donne le résultat de la contraction 3B-Cid sur l'axe x1 :



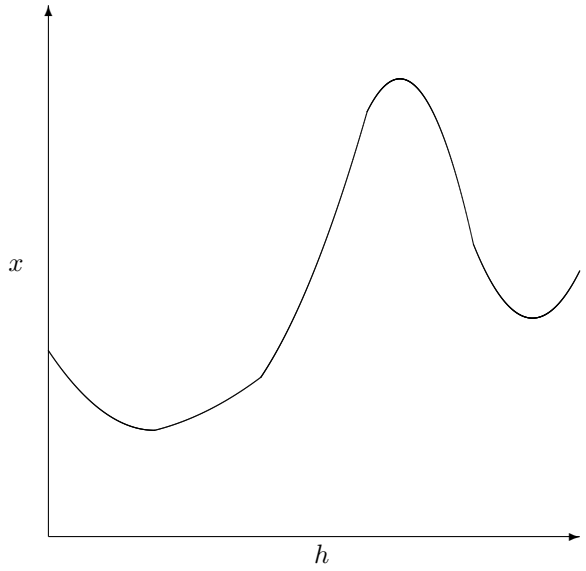
Ce travail est à faire sur toutes dimensions. Ci-dessous un pseudo code de cet algorithme :

Data: Ctc (Contracteur, ex : HC4), BoxE (Boîte d'entrée), N (nombre de sous-boîtes, ex : 10)
Result: BoxS (Boîte de sortie contractée)
BoxS = BoxE;
for *i nombre de dimension de BoxS* **do**
  list = liste de sous-boîte selon l'axe i de gauche à droite;
  BoxGauche = emptyBox;
  BoxDroite = emptyBox;
  BoxCentre = emptyBox;
  k_gauche = 0;
  k_droite = N-1;
  for *k=k_gauche ; k < k_droite & BoxGauche = emptyBox ; k++* **do**
    BoxGauche = Ctc(list[k]);
    k_gauche = k;
  end
  k_gauche - -;
  for *k=k_droite ; k > k_gauche & BoxDroite = emptyBox ; k - -* **do**
    BoxDroite = Ctc(list[k]);
    k_droite = k;
  end
  k_droite ++;
  if *k_droite - k_gauche* > 2 **then**
    BoxCentre = Hull(list[$\forall k \in (k\_gauche + 1, k\_droite - 1)$]) ;
  end
  BoxS = Hull(BoxGauche,BoxDroite,BoxCentre);
end

Algorithm 1: 3B-Cid

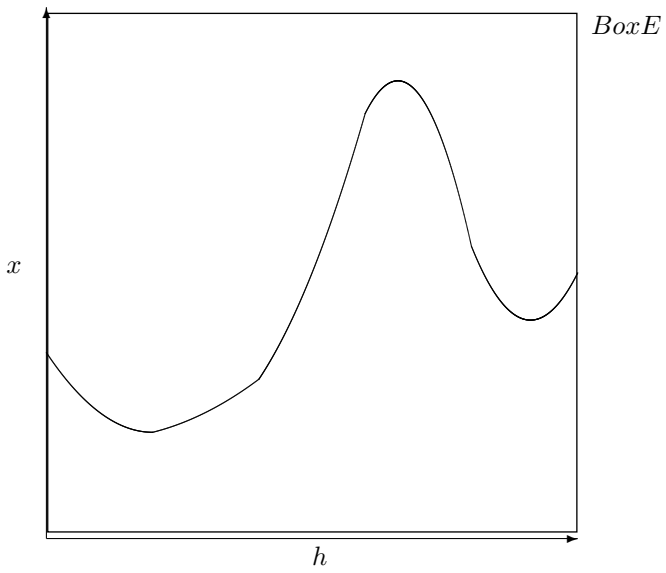
4.4 Algorithme

Pour expliquer l'algorithme du problème d'homéostasie, prenons comme exemple un problème à 2 dimensions, une dimension en paramètre et une dimension en variable.



4.4.1 Etape 0, initialisation :

On commence par une première contraction sur une boîte extérieure infinie pour les variables et borné sur l'espace de travail pour les paramètres pour avoir BoxE :



On créait alors une liste de boîtes ne contenant initialement qu'une boîte égale à BoxE. On commence alors la boucle :

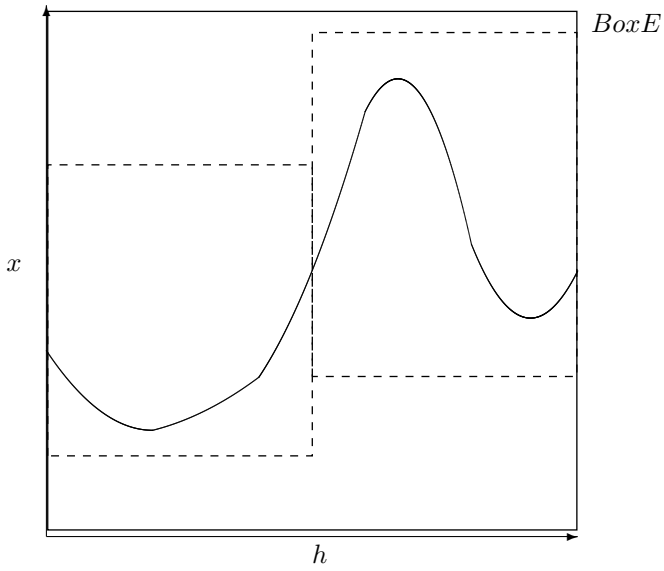
4.4.2 Etape 1, bissection, contraction :

On choisit une dimension de bissection dans le vecteur des variables ou le vecteur des paramètres. Dans l'algorithme implémenté, on prend les dimensions cycliquement dans l'ordre.

On sélectionne toutes les boîtes de la liste ayant au moins un bord des variables touchant la boîte extérieur. Chacune de ces boîtes est alors bissectée sur la dimension précédemment choisie et ajouter à la liste (en remplaçant la boîte initiale).

Chacune de ses nouvelles boîtes sont alors contractée avec une variante de 3B-Cid. En effet comme expliqué plus tôt, dans 3B-Cid, la découpe pour contracter ce fait sur tous les axes de la boîte. Ici, la variante ne va faire la découpe que selon les axes ayant au moins un bord des variables touchant la boîte extérieure.

On aurait alors pour notre problème quelque chose comme ceci avec une première bissection en h :



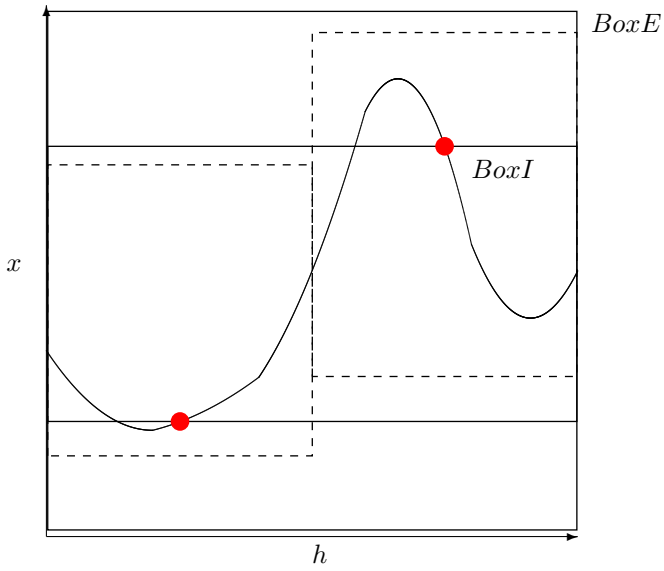
4.4.3 Etape 2, boîte intérieure :

On va maintenant travailler sur la boîte intérieure. Pour toutes les boîtes de la liste, on va y chercher un point solution.

Pour cela, on verrouille le vecteur paramètre en prenant les points au milieu des intervalles de [h]. On utilise le solveur par défaut d'Ibex puis Loup-Finder pour trouver un point solution. En effet le solveur aurait pu suffire mais le système n'étant pas carré, le solveur nous renvoie un intervalle, certes petit mais sans pouvoir nous assurer qu'il s'agisse bien d'une solution. C'est pour cela que l'on utilise Loup-Finder, pour avoir un point solution et pour nous servir de preuve de la solution, même si en réalité, c'est une confirmation à une précision près. (Le solveur et Loup Finder nécessite une fonction à maximiser ou minimiser. Dans l'implémentation, je n'ai pas réussi à contrôler cela, donc par défaut, cela ne cherche qu'à minimiser x1.)

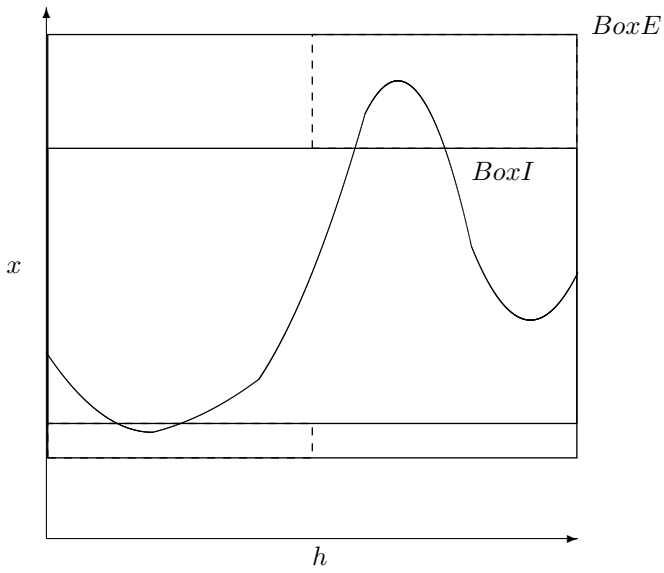
On fait alors le hull de tous les points solutions et la boîte intérieure (initialement vide) pour obtenir la nouvelle boîte intérieure. A noter que dans la boîte intérieure l'on conservera toujours sur les intervalles des paramètres les bornes de BoxE.

On a alors comme résultat :



Enfin on va soustraire, à toutes les boîtes de la liste, BoxI. Il y a trois configurations possibles. Soit la boîte est entièrement inclus dans BoxI, alors cette boîte est tout simplement supprimée. Soit la boîte se retrouve coupée que sur un seul axe, il s'agit de rogner cette boîte sur l'a dit dimension. Soit la soustraction est sur plusieurs dimensions, auquel cas le résultat est décrit par un ensemble de boîtes. Dans ce dernier cas on ne fait pas de soustraction. En effet le faire provoque rapidement une explosion inutile du nombre de boîtes à gérer dans la liste, ce n'est pas rentable.

La dernière petite étape est de recréer BoxE en faisant le hull de l'ensemble des boîtes de la liste. On obtient alors pour notre exemple le résultat suivant :



4.4.4 Etape 3 : critère d'arrêt.

Enfin, on arrive avec deux boîtes, extérieure et intérieure pour encadrer BoxO. On définit comme critère d'arrêt le ratio entre le périmètre de ces deux boîtes. On peut aussi, au choix, décider de terminer la boucle au bout d'un nombre fixé d'itération. On garde ensuite BoxE comme approximation de BoxO.

4.4.5 Remarques

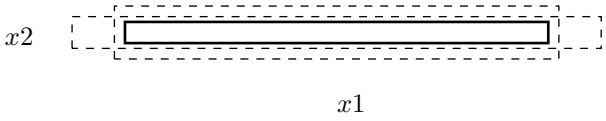
Quelques petites remarques sur certain choix de l'algorithme.

A chaque itération, c'est sur l'ensemble des boîtes de la liste touchant BoxE que l'on travaille. Cela n'a pas toujours été le cas. Au début, on travaillait sur une boîte à la fois par itération. Plusieurs critères de sélection ont été testé comme choisir la boîte touchant sur le plus grand nombre de dimensions, ou la boîte au plus grand périmètre ou volume. Mais au final cela avait tendance à finir sur beaucoup de calcul sur certain côté au détriment de l'ensemble.

L'algorithme, qui est un solveur utilise lui-même un solveur et Loup Finder. C'est une solution étrange mais cela était nécessaire comme critère de preuve pour les points intérieurs.

4.5 Conclusions et critiques

L'algorithme présenté précédemment a été comparé avec les résultats de DESOEUVRES Aurélien. La comparaison a été portée sur le temps de calcul et sur les boîtes résultantes. Pour les boîtes, on peut comparer soit leurs volumes, soit leurs périmètres. Pendant longtemps j'ai considéré ces deux approches comme équivalentes et choisis le périmètre comme critère. Au final, je suis arrivé à un périmètre inférieur, mais avec un volume supérieur.



Sur la figure ci-dessus, BoxO est la boîte continue, on peut voir que si la boîte qui l'entoure la plus longue selon x1 a un périmètre plus important que l'autre, elle a tout de même un volume plus faible. La boîte la plus longue est le résultat de DESOEUVRES Aurélien quand l'autre est le résultat de mon algorithme.

En réalité, cette différence est due au fait que mon algorithme contracte mieux les intervalles les plus larges et moins bien les intervalles les plus petits. Or, et c'est là mon erreur d'interprétation, en revenant au problème physique (ou plutôt chimique) plus qu'au problème mathématique, c'est bien les intervalles les plus fins qui nous intéressent le plus puisque nous avons affaire à un problème d'homéostasie. Dans un problème d'homéostasie les variables les plus importantes sont les variables qui doivent rester proche d'une valeur d'équilibre.

L'algorithme répond donc bien aux impératifs d'un point de vue mathématique mais moins sur le plan physique. Il pourrait donc être utilisé sur des problèmes mathématiques similaires, mais pour l'homéostasie il nécessiterait des adaptations pour creuser en priorité les intervalles les plus petits (ce qui est assez non-intuitive). C'est d'ailleurs ce que fait l'algorithme de DESOEUVRES Aurélien, il délaisse les dimensions aux intervalles trop grands pour se concentrer sur les dimensions les plus susceptibles d'être homéostatique.

Enfin, pour le temps de calcul, la comparaison est à prendre avec des pincettes car les calculs n'ont pas été fait sur la même machine et l'algorithme de DESOEUVRES Aurélien utilise le calcul en parallèle qui, si ce n'est pas encore le cas, pourrait tout à fait être implémenté pour mon programme. Pour autant l'ordinateur utilisé par DESOEUVRES Aurélien semble plus puissant que le mien. Donc pour un périmètre similaire, il me faut 2 secondes de calcul contre 3 pour l'autre algorithme ou 14 secondes sans la parallélisation.

On a donc finalement un meilleur temps de calcul, dommage que ce ne calcule pas ce qui nous intéresse.

5 Pavage régulier par intervalle.

5.1 Grotte karstique

RADWAN Verlain est un doctorant qui travaille sur une méthode SLAM par intervalles dans le but d'explorer des grottes karstiques. Une grotte karstique est comme une rivière souterraine. Pour le contexte, la ville de Montpellier est une ville en expansion qui risque dans les prochaines années d'avoir des problèmes d'approvisionnement d'eau. Pouvoir cartographier les grottes karstiques qui se trouvent aux alentours pourrait permettre à terme de les exploiter pour les besoins de la ville. Mais l'exploration de telles grottes et qui plus est sur la distance est dangereuse pour des plongeurs. C'est là qu'intervient l'utilisation de robots autonomes pour cette exploration.

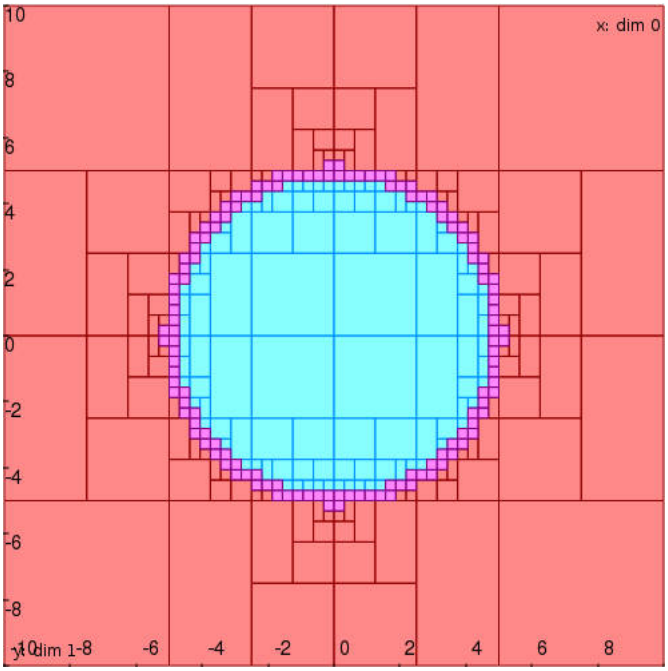
Qui dit robot autonome explorant, dit méthode SLAM. Cette méthode SLAM devant se servir comme repère de la surface de la grotte. Pour cette problématique la méthode en cours de recherche par RADWAN Verlain se base initialement de la méthode Dig SLAM de l'enseignant chercheur JAULIN Luc. Cette méthode sert de base car le problème ici considéré est en 3 dimensions et dispose d'informations plus riches.

Sans rentrer dans les détails de la méthode RADWAN Verlain car pas encore acter, J'ai pu l'aider sur deux points : la projection et la visualisation de résultat en 3 dimensions.

5.2 Architecture des données

L'idée étant de définir un espace de n dimensions en intervalles, la structure utilisée est un pavage régulier en arbre. L'origine de l'arbre est la boîte contenant l'ensemble de l'espace de travail. Dans l'exemple simplifié de deux états pour caractériser l'espace d'une grotte : eau et roche, le noeud d'origine va avoir pour état eau & roche et deux branches vers deux boîtes équi-volumique bissectant sur la dimension du plus grand côté. L'arbre se construit ainsi de manière récursive jusqu'aux feuilles, quand la boîte courante ne contient que l'état roche ou eau, ou quand la boîte atteint une petite taille définie par la résolution recherchée, la feuille prend alors l'état eau & roche.

Voici ci-après un exemple de découpage qui découle de cette structure de données. Les utilisateurs de vibes y trouveront une représentation familière. Mais ici ce n'est plus juste une découpe pour le calcul mais une découpe régulière pour le stockage et le traitement des données.



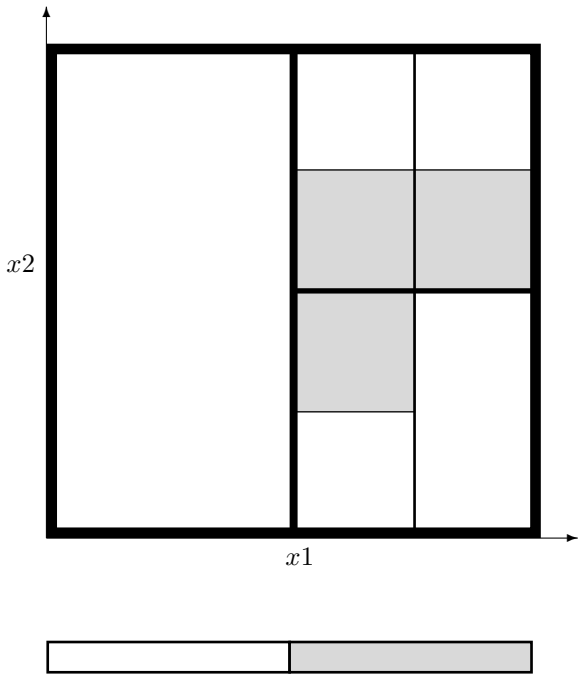
L'état eau & roche n'est qu'un exemple, en réalité il y a trois bits représentant les états primaires : in, out, maybe. La superposition de ces trois états par un OR inclusif permet d'avoir des états intermédiaires sur les branches de l'arbre et donc de faciliter les recherches lors du parcours de l'arbre.

Si cette architecture a initialement été pensée pour répondre aux problématiques de la méthode SLAM de DESOEUVRES Aurélien, elle à été travaillée de manière plus générale pour proposer un pavage régulier ainsi que des outils pouvant s'intégrer à terme à la bibliothèque CODAC.

5.3 Projection

Ma première contribution à été de réaliser une projection sur le pavage. En effet, le pavage est à n dimensions. Pour diverses raisons, il peut être intéressant de ne travailler que sur certaines dimensions du pavage. Par exemple, pour une carte en 3 dimensions, il peut être nécessaire de passer en 2 dimensions pour en faciliter l'affichage.

Cette projection nécessite des choix de priorité sur ce qui nous intéresse. Par exemple, pour une boule le résultat devra être un rond comme celui vu dans la partie précédente. Si la boule est considérée comme in, on souhaite que le rond soit in. Or une projection n'est pas une section. L'extérieur du rond est bien out mais l'intérieur devrait être in et out. C'est pour cela qu'il faut bien préciser ici le choix de considérer le in comme prioritaire et de ne garder que celui-ci dans le résultat de la projection.

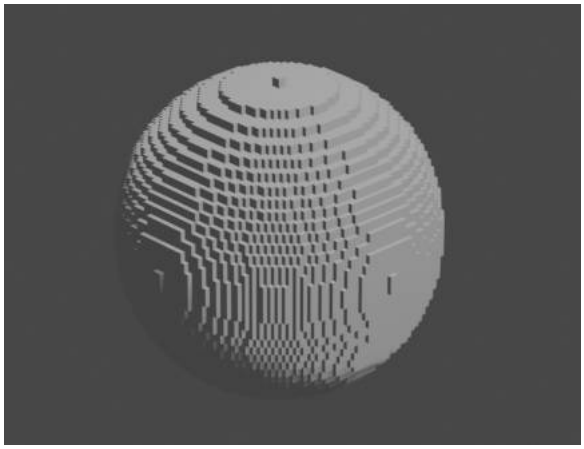


Ci-dessus, on voit l'exemple d'une projection de 2 à 1 dimension. Si les in sont visuellement collés, ce n'est pas le cas dans l'organisation de l'arbre, ils sont sur des branches différentes. En conséquence, ce n'est pas l'arbre 2D qui va donner la projection mais la projection qui va lire l'arbre 2D pour se construire. L'algorithme crée les noeuds récursivement, chaque noeud parcourt l'arbre pour tester tous les boîtes qui sont contenues dans sa boîte. S'il y a plusieurs états, le noeud se bissecte en deux branches, sinon il devient une feuille.

Le passage de 3 à 2 dimensions n'est qu'une application de cette projection. Elle peut aussi permettre d'isoler certaines dimensions pour certains calculs. En effet des dimensions ne peuvent être que des variables de calcul inutiles pour la suite. De plus la fonction que j'ai conçue peut remettre chaque dimension dans l'ordre voulu.

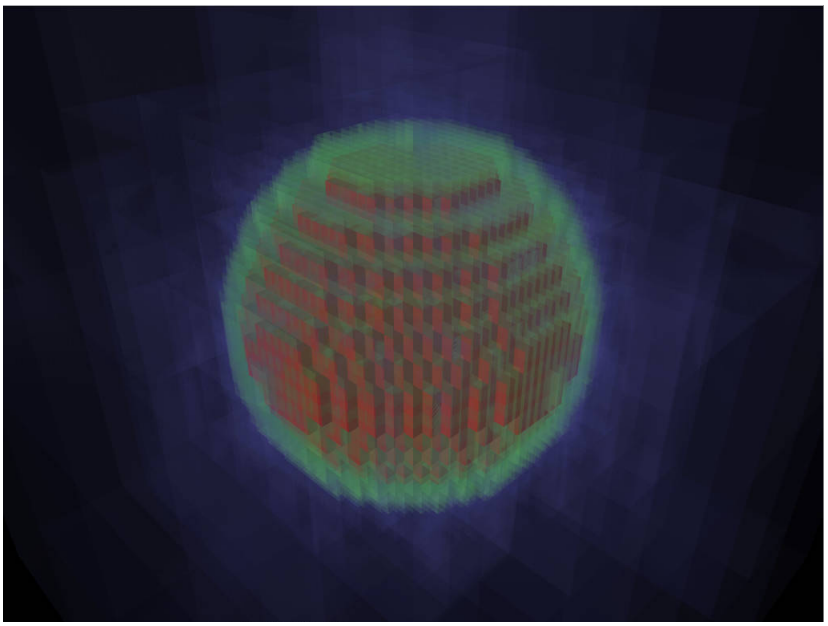
5.4 Visualisation 3D

Si la projection permet d'avoir un aperçu en deux dimensions du pavage, par la suite il sera intéressant d'avoir une visualisation directement en trois dimensions. La première étape pour cela a été de développer une fonction permettant de générer à partir d'un pavage un fichier PLY. Ce fichier peut ensuite être lu par des applications tierses comme par exemple ci-dessous avec Blender :



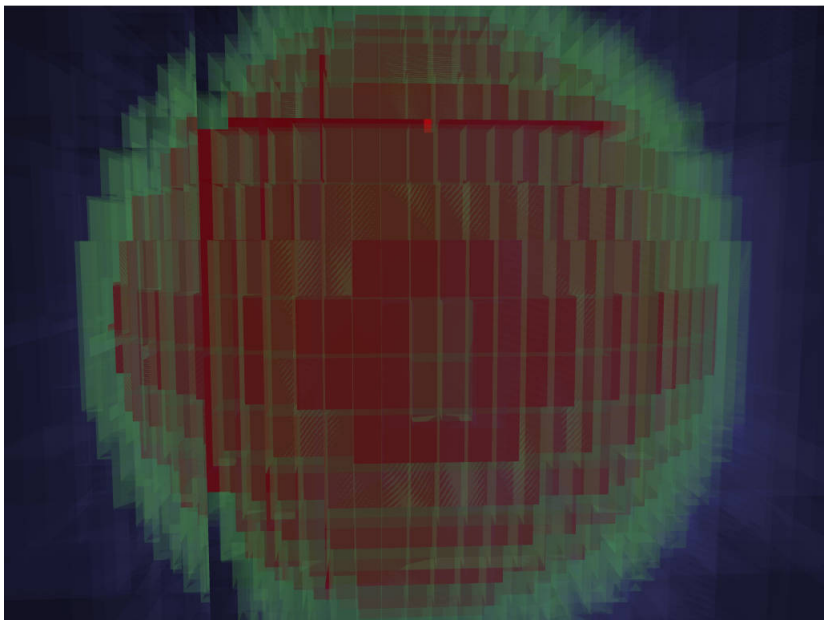
Mais j'ai échoué à utiliser la transparence avec les fichiers PLY. Cela restreint l'affichage d'un seul état de boîte. Pour voir l'ensemble de la scène, ainsi que le découpage des boîtes, il a donc été nécessaire de trouver une alternative. J'ai donc développé une application dédiée à ce problème grâce à la bibliothèque OpenGL 3 que j'ai découvert à cette occasion. Cette fois-ci, le fichier n'est plus de type PLY. En effet, s'il serait possible d'y rajouter les informations de couleur et de transparence, il faudrait relancer le programme à chaque fois pour les modifier. Avec un type de fichier dédié, on peut non plus mettre les données RGBA mais mettre l'état du pavage et ainsi modifier l'affichage plus facilement.

Ci-dessous un exemple du résultat obtenu :



Grâce à la transparence, on peut voir les différents états out (bleu), maybe (vert) et in (rouge). Les couleurs et transparences peuvent être choisies dans un fichier annexe palette.txt. Le programme permet de faire pivoter / zoomer la figure autour du centre ainsi que de changer l'orientation de la lumière.

Mais la transparence est capricieuse pour le calcul 3D en temps réel. Elle requiert d'agencer les vertices des plus éloignés aux plus proches de la caméra. Cela demande de les trier. Dans les jeux-vidéos, on cherche à limiter le nombre de texture transparente pour cela. Mais dans le cas qui nous concerne cela n'est pas possible, et trier des milliers de vertices à chaque frame non plus. De ce fait le programme précalcul l'agencement pour 8 positions, les 8 sommets d'un cube. Mais quand la direction de la caméra est presque parallèle à une face de ce cube, et puisqu'il s'agit d'une projection en perspective et non orthogonale, on a alors les problèmes d'affichage montré ci-dessous :



5.5 Problème de l'altitude

J'ai aussi pu apporter ma contribution pour révéler une mauvaise piste. En effet, le robot devant évoluer dans un environnement initialement inconnu nécessitant un SLAM, toute donnée est bonne à prendre. Par exemple celle de l'altitude. Généralement pour les robots aquatiques, la pression de l'eau est une bonne indication de l'altitude. Elle a donc été par réflexe considéré comme une donnée qui permettrait à tout instant de connaître la position du robot sur l'axe verticale.

Mais dans notre cas de grotte karstique, cela ne s'applique plus. En effet la formule :

$$P(z) = P_0 + \rho \overrightarrow{g} \overrightarrow{z}$$

n'est valide qu'en statique.

Si cela peut servir de bonne approximation pour les robots aquatiques habituels, ce n'est plus le cas pour notre situation. En effet, la grotte peut être représentée comme un long tuyau où l'eau s'écoule avec des pertes de charges a priori inconnue. Pour bien sentir le problème, ce serait comme immerger un robot dans un fleuve. La pression nous donnera la profondeur du robot par rapport à la surface du fleuve mais pas par rapport à des points fixes comme sa source ou la mer. Attention, la comparaison du fleuve est juste pour illustrer les pertes de charge, dans un tuyau rempli d'eau en mouvement la pression ne nous donnera pas la hauteur au plafond.

J'ai donc pu mettre au jour ce problème sans pour autant réussir à apporter une solution.

6 Bibliographie

1. Desoeuvres A., Trombettoni G., Radulescu O. (2020) Interval Constraint Satisfaction and Optimization for Biological Homeostasis and Multistationarity. In : Abate A., Petrov T., Wolf V. (eds) Computational Methods in Systems Biology. CMSB 2020. Lecture Notes in Computer Science, vol 12314. Springer, Cham. https ://doi.org/10.1007/978-3-030-60327-4_5

https ://www.biorxiv.org/content/biorxiv/early/2020/05/15/2020.05.14.095315.full.pdf

2. B. Neveu and G. Trombettoni, "Adaptive Constructive Interval Disjunction," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, 2013, pp. 900-906, doi : 10.1109/ICTAI.2013.138.

https ://ieeexplore.ieee.org/document/6735349

3. Luc Jaulin. Range-Only SLAM With Occupancy Maps : A Set-Membership Approach. IEEE Transactions on Robotics, IEEE, 2011, 27 (5), pp.1004-1010.

https ://www.ensta-bretagne.fr/jaulin/paper_dig_slam.pdf