Rapport TP

ALERTES AUX DÉFORMATIONS LE LONG D'UN CÂBLE

Albac Baptiste
Caillette Alice
Creuzet Antoine
Charles Esteban
INSA CENTRE VAL DE LOIRE
ALGORITHMIQUE ET PROGRAMMATION 4

Introduction

Le but de ce TP est de créer un algorithme qui prend en entrée un câble, d'une distance définie, sur lequel on applique des déformations et qui en sortie rapporte à l'utilisateur les alertes de déformations sur le câble.

Une alerte est donnée pour une position lorsque le seuil de déformation sur la position en question et ces positions voisines est dépassé.

La première partie du programme permet d'attribuer les déformation tout le long du câble selon plusieurs lois de distribution : uniforme, monomodale, bimodale, quasimonotone.

L'analyse des résultat se fera pour chaque loi de distribution. nous ne nous intéresserons pas à cette partie, on se contente de récupérer le résultat de cette partie pour l'exploiter dans nos algorithme.

Pour ce travail, l'indicateur de performance est le temps mis par l'algorithme pour affecter les tâche nécessaire qui lui sont attribués pour réaliser le travail demandé.

Note : En raison du travail à distance, nous avons pris l'initiative d'utiliser GitHub (une plateforme open source de gestion de versions et de collaboration). Nous n'utilisons pas toutes ses fonctionnalités, néanmoins ce dernier nous permet d'observer l'avancer de notre projet, les modifications apportées par nos camarades et le concept de branche nous permet de centraliser nos différents codes dans un master commun.

➢ Compte rendu de la première séance 13/03/2020

Nous avons commencé par discuter du sujet et comprendre ce que l'algorithme proposé faisait. Nous avons compris qu'en rentrant une taille d'un câble et un nombre de déformation l'algorithme allait générer un fichier qui va "distribuer" le nombre de déformation à certaines positions. Cela implique qu'à une certaine position plusieurs déformations peuvent exister. Le problème étant que si un nombre seuil (dans notre cas 100)est atteint sur la position et ses voisins une alerte s'active . Les voisins d'une position sont les 100 positions avant et après la position en question. Cependant si la position est à moins de 100 voisins de chacune des extrémités les intervalles seront plus petits (et donc potentiellement moins de risque que le nombre de déformations soit supérieur à 100 et donc que l'alerte s'active).

Version 1 :

➢ fonctionnement de l'algorithme

Voici la marche suivie dans cette première version pour résoudre le problème.

Dans un premier temps nous cherchons à créer un tableau intermédiaire (« tab_position ») qui affecte à chaque position son nombre de déformations comme ci-dessous. Effectivement le tableau renvoyé par le rapport un tableau de position pour chaque déformation (« paquet »). Ils nous semblaient plus simple de traiter le tableau inverse.

Positions	Nb de déformations
0	2
1	1
2	3
3	0

Pour obtenir ce tableau, on crée d'abord un tableau qui a autant de case que de position possible (allocation dynamique pour ne pas saturer la mémoire statique). Ensuite une boucle parcourt l'ensemble des positions possible. Pour chaque position on parcourt le tableau « paquet » à la recherche de la position. Si la position est trouvée alors il y a une déformation à cette position. On ajoute +1 dans la case qui correspond à la position dans le tableau « tab_position ».

Ensuite on parcourt ce tableau et on fait la somme des déformations de la position voulue et des 100 avant et 100 après (un total de 201 lignes). Si la somme des déformations sur les 201 positions ,position étudié plus les voisines, est supérieur au seuil d'alerte fixé alors l'alerte est donné. On enregistre dans un fichier les positions pour lesquelles il y a une alerte avec le nombre total de déformations enregistrés.

➢ Analyse des résultats

Pour cette version l'analyse des résultats est assez simple car les temps d'exécution est très long. En moyen l'algorithme met environ 150s pour effectuer toutes les tâches. Pour obtenir des résultats de performance convenable il ne faut pas dépasser une longueur de 100 km pour le câble et pas plus de 10000 déformations.

➢ Terminaison et Correction

Après avoir parcouru toutes les valeurs de déformation, **l'algorithme se termine** suivant une fonction qui, pour un algorithme itératif décroît à chaque boucle (décroissance du nombre restant de position à étudier).

L'algorithme est correct : il retourne, quand il calcule en partant des données, un objet qui est un des résultats escomptés et qui satisfait la spécification du résultat comme énoncé dans la description de l'algorithme.

➢ Analyse de complexité : le coût temporel

Nous cherchons donc à donner un ordre de grandeur pour avoir une idée de l'efficacité ou non d'un algorithme.

Nous avons donc commencé à compter le nombre d'opérations élémentaires : nous avons une dizaine d'opérations élémentaires dans la première version qui valent $O(1)$.

Nous avons aussi 1 boucle for avec un autre for imbriqué et dans chaque for nous avons des opérations élémentaires indépendantes l'une de l'autre. Cela nous donne :

$$O(n) \times O(n) = O(n^2)$$

Et enfin nous avons une dernière boucle for qui contient une autre boucle for chacune avec des opérations élémentaires donc la complexité est de $O(n^2)$.

Conclusion : $n^2 + n^2 + 1 \sim 2 \times n^2$ donc nous avons une complexité de $O(n^2)$.

Version 2:

➢ Fonctionnement de l'algorithme

Le déroulement de la seconde version est le suivant :

- Dans un premier temps, on crée une boucle qui parcourt, pour toute la longueur du câble chaque position.
- Lorsque l'on prend une position, on crée à partir de celle-ci, un encadrement [min, max].
- Puis on parcourt le paquet et pour chaque position située dans l'encadrement préalablement créé, une variable prend +1.
- Si cette variable est au-dessus du seuil d'alerte, on met dans la case d'un nouveau tableau un 1 et si on ne l'atteint pas on met 0.
- Enfin, une boucle d'affichage parcourant le nouveau tableau sera utilisée pour détecter les alertes aux indices (qui sont des positions) des cases qui comportent des 1.

Le dernier tableau, celui qu'on parcourt pour afficher les alertes, peut prendre 2 valeurs différentes pour chaque position :

- 0 : La position a été vérifiée mais il n'a pas nécessité de lancer une alerte (variable inférieur au seuil d'alerte pour la position).
- 1 : La position a été vérifiée et il est nécessaire de lancer une alerte (variable supérieur au seuil d'alerte pour la position).

➢ Prémisses de la version

L'idée de la seconde version était de faire le moins d'étapes possibles en utilisant au maximum une boucle for imbriquée.

Brouillon en pseudo langage de la seconde version :

```
Algorithme seconde_version : (tbl, SEUIL) -> (x) selon
n <- 0
x <- 0
Tant que (n = ncomp(tbl)) répéter
        min <- tbl(n) - 50
        max <- tbl(n) + 50
        i <- 0
        Tant que (i = ncomp(tbl)) répéter
                Si (min<=tbl(i)<=max) alors
                        a <- a+1
                        Si a = SEUIL alors créer_alerte(n), x <- x+1.
                .
                i <- i+1
        .
        n <- n+1
.
```

Cet algorithme est le premier algorithme sur lequel nous avons travaillé pour la seconde version.

On peut voir différents problèmes après coup sur cette version.
Premièrement, nous nous sommes rendu compte que les minimums et maximum ne sont pas toujours ceux-là. Il suffit de se placer aux extrémités de la plage d'étude pour s'en rendre compte.
Nous avons donc créé des conditions sur le maximum et le minimum pour préciser nos calculs pour des conditions particulières.
Ensuite un des problèmes était que pour les valeurs qui ne sont pas dans le tableau de déformations, le calcul ne se faisait pas.
Pour remédier à cela, la première boucle ne circulera plus sur le tableau mais sur la longueur du câble.

➢ Analyse des résultats

Lors de l'exécution de l'algorithme, le temps de l'exécution est compté sur l'ensemble des actions qu'il effectue.

Pour cette version le temps d'exécution est relativement faible. Le temps d'exécution est quasiment similaire à celui de la 3ème version et cette version est optimisée pour des cas précis comme un rapport nombre de déformations sur longueur du câble faible.

➢ Terminaison et Correction

Après avoir parcouru toutes les valeurs de déformation, **l'algorithme se termine** suivant une fonction qui, pour un algorithme itératif décroît à chaque boucle (décroissance du nombre restant de position à étudier).

L'algorithme est correct : il retourne, quand il calcule en partant des données, un objet qui est un des résultats escomptés et qui satisfait la spécification du résultat comme énoncé dans la description de l'algorithme.

➢ Analyse de complexité : le coût temporel

Nous cherchons donc à donner un ordre de grandeur pour avoir une idée de l'efficacité ou non d'un algorithme.

Nous avons encore dans cette version une dizaine d'opérations élémentaires qui valent $O(1)$.

Nous avons aussi 1 boucle for avec une autre boucle for imbriquée et dans chaque for nous avons des opérations élémentaires indépendantes l'une de l'autre. Cela nous donne :

$$O(n) \times O(n) = O(n^2)$$

Et enfin nous avons une dernière boucle for qui contient des opérations élémentaires donc la complexité est de $O(n)$.

Conclusion : $n^2 + n + 1 \sim n^2$ donc nous avons une complexité de $O(n^2)$.

Version 3 :

➢ Fonctionnement de l'algorithme

Cette version fonctionne exactement sur le même principe que la première mais avec une grosse optimisation de la création du tableau des déformation en fonction des positions.

Ont créé toujours un tableau avec autant de case que de position. Ensuite on initialise toute ces casses à 0 car avant de parcourir le paquet il y a aucune déformation sur chaque position et c'est aussi une sécurité car alloué de la mémoire ne veux pas dire qu'il n'y a rien dans l'allocation faite. Ensuite on parcourt le nombre de déformation maximal. Pour chaque déformation on lit la position attribué avec le table « paquet ». paquet[déformation] est donc une position. Cette position est directement injectée comme paramètre du table de déformation en fonction des position (« tab_position »). On ajoute donc +1 (une déformation supplémentaire) dans cette case.

Le calcul du minimum et du maximum à parcourir pour les voisins et le test si le seuil d'alerte est identique à la première version.

➢ Analyse des résultats

Lors de l'exécution de l'algorithme, le temps de l'exécution est compté sur l'ensemble des actions qu'il effectue.

Pour cette version le temps d'exécution est relativement faible, de l'ordre de la demi seconde, on fait donc une moyenne de trois exécutions consécutive.

- Uniforme : **0.386 s**

Lors de ce test 32836 alertes ont été détectés et on remarque une concentration des alertes sur le début du câble.

- Monomodale : **0.382 s**

Lors de ce test 92608 alertes ont été détectés, 3 fois plus que pour la distribution uniforme mais cela n'impacte pas le temps d'exécution. cela est logique car pour chaque position on test s'il y a une alerte quel que soit le nombre d'alerte final. On devrait quand même obtenir un temps un peu plus long car lorsqu'une alerte est détectée, elle est répertoriée dans un fichier donc plus d'alerte signifie plus d'écriture dans le fichier donc plus de temps.

Pour la répartition des déformations, on remarque qu'elles sont réparties entre la position 200000 et 400000. On peut dire que les déformation sont un peu mieux répartie dans cette version par rapport à la précédente.

- Bimodale : **0.391 s**

Lors de ce test 91942 alertes ont été détectés. les résultats sont sensiblement identiques à la distribution monomodale mais avec encore une meilleur répartition des déformation car elles sont réparties entre la position 200000 et 750000.

- Quasimonotone : **0.369 s**

Lors de ce test 32830 alertes ont été détectés. les résultats sont quasiment identiques à une répartition des déformation par une loi uniforme.

➢ Terminaison et Correction

Après avoir parcouru toutes les valeurs de déformation, **l'algorithme se termine** suivant une fonction qui, pour un algorithme itératif décroît à chaque boucle (décroissance du nombre restant de position à étudier).

L'algorithme est correct : il retourne, quand il calcule en partant des données, un objet qui est un des résultats escomptés et qui satisfait la spécification du résultat comme énoncé dans la description de l'algorithme.

➢ Analyse de complexité : le coût temporel

Nous avons encore dans cette version une dizaine d'opérations élémentaires qui valent $O(1)$.

Dans cette troisième version nous cherchions à avoir une complexité meilleure que les deux précédentes, nous avons donc 3 boucles for indépendantes qui ont chacune une complexité $O(n)$.

Conclusion : $3n + 1 \sim 3n$ donc la complexité de cet algorithme est de $O(n)$.

En comparaison avec les deux autres nous voyons une très nette amélioration de l'efficacité de l'algorithme.

Conclusion :

➢ Commentaire sur les résultats expérimentaux

Compilation de la 1ère version :

```
Simulation de 600 deformations sur 1000 positions, configuration 3... termine.
Ecriture du paquet de deformations dans simulation_deformations.dat... terminee.

quel version souhaitez vous executer?
[1] premiere version
[2] deuxieme version
[3] troisieme version
[4] exit
1
premiere version en cours d'execution...
termine
nb d'alerte : 673
Temps ecoule premiere version : 3 ms.
```

Compilation de la 2ème version :

```
Simulation de 600 deformations sur 1000 positions, configuration 3... termine.
Ecriture du paquet de deformations dans simulation_deformations.dat... terminee.

quel version souhaitez vous executer?
[1] premiere version
[2] deuxieme version
[3] troisieme version
[4] exit
2
deuxieme version en cours d'execution...
nombre d'alertes : 673
termine
Temps ecoule deuxieme version : 3 ms.
```

Compilation de la 3ème version :

```
Simulation de 600 deformations sur 1000 positions, configuration 3... termine.
Ecriture du paquet de deformations dans simulation_deformations.dat... terminee.

quel version souhaitez vous executer?
[1] premiere version
[2] deuxieme version
[3] troisieme version
[4] exit
3
troisieme version en cours d'execution...
termine
nb d'alerte : 673
Temps ecoule troisieme version : 3 ms.
```

Les 3 versions donnent le même nombre d'alertes et mettent le même temps pour les exécuter. On peut voir que le temps écoulé est très inférieur à celui demandé dans le cahier des charges mais nous sommes dans un cas ou le nombre de déformations et le nombre de positions sont faibles. Ainsi la complexité temporelle est plus faible lorsque ces deux paramètres sont plus faibles ce qui est ressorti de l'étude théorique de la complexité des codes.

➢ Conclusion sur le projet

Les deux dernières versions sont les plus optimisées et les plus complètes. Elles répondent en totalité au cahier des charges donné, notamment le fait que le code ne doit pas dépasser un temps imparti.

Cette étude de cas a été très enrichissante car le sujet était très concret, cela nous a permis notamment d'approfondir les notions vu en cours et d'en comprendre les enjeux sur un problème "réel". Le travail en groupe était stimulant et motivant malgré les contraintes auxquelles nous avons dû faire face. Nous avons réussi à nous organiser de manière claire et efficace pour faire le travail dans les délais et nous en garderons une bonne expérience.