Rami El Beblawy
Edwin Srithammavanh

Compte-Rendu de projet 2I006

Le problème de Via Minimization

Partie A

Exercice 1 – Lecture des instances et visualisation
netlist.c visunetlist.c (SVGwriter.c SVGwriter.h)
La structure utilisée est celle fournie dans l'énoncé du sujet.
La commande «make visunetlist» créé un executable VisuNetlist qui permet de tracer une instance dans un fichier html.
Usage : ./VisuNetlist [chemin de l'instance] [longueur] [largeur]
Avec longueur et largeur les dimensions x et y du graphique en pixels.

Exercice 2 - Algorithme naïf de recherche des intersections :
intersection.c

Exercice 3 - Algorithme de recherche des intersections par balayage et comparaison des performances :
balayage.c benchmarking.c

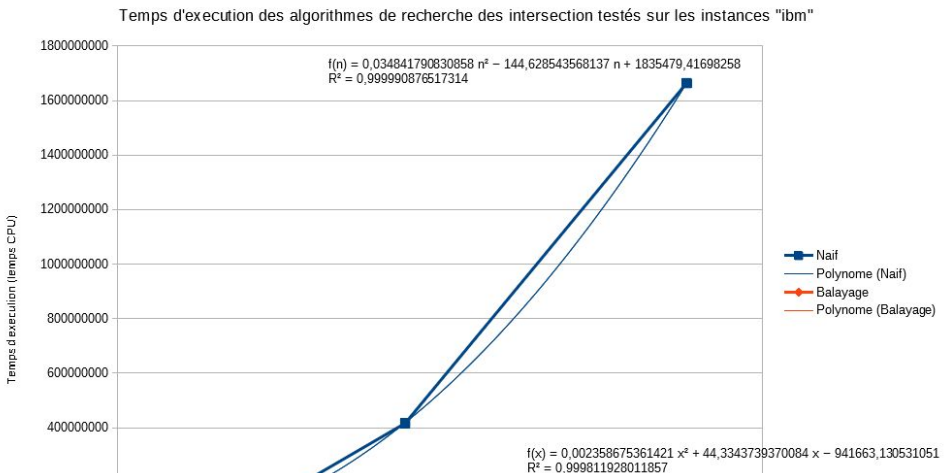
Q 3.3) Calcul de la complexité de cette méthode :

E et T sont des listes chaînées de taille O(n).
La création de E est en O(nlogn) car c'est une liste triée.
La boucle Pour parcourt O(n) éléments.

- Dans cette boucle Insérer(h, T) et Supprimer(h, T) sont en O(n) car dans le pire des cas il faut parcourir tous les éléments de T pour trouver la bonne place pour h, T étant une liste triée.
- Prem_segment_apres(y, T) est en O(n) car il faut parcourir T pour trouver le bon élément.
- La boucle Tant que peut-être itérée O(n) fois dans le cas où tous les segments horizontaux ont une ordonnée comprise entre les ordonnées des extrémités du segment vertical v considéré.

Dans ce cas l'algorithme a pour complexité **$O(n^2)$**

Avec l'hypothèse que le nombre de segments horizontaux traversés par une droite verticale du plan est borné par α la boucle Tant que est en O(α). Donc la complexité de l'algorithme est **$O(\alpha n)$**.
Donc si α<<n la complexité est en O(n).
Q 3.4) Mesure et comparaison des temps d'exécution des méthodes naïves et par balayage :



Les courbes de tendances quadratiques ont un coefficient de détermination très proche de 1 ce que semble confirmer la complexité théorique de $O(n^2)$ pour les deux méthodes.
On remarque que le temps d'exécution de la méthode par balayage est beaucoup plus court que pour la méthode naïve et de plus les instances ibm possèdent une borne α commune. Donc cela vérifie la complexité en O(αn) calculée lorsque l'hypothèse de l'existence de α est vérifiée.

Exercice 4 – Algorithme de recherche des intersections utilisant la structure d'AVL
avl.c benchmarking.c

Q 4.4) Complexités des méthodes de manipulation de l'AVL et de la méthode par balayage :

Prem_Noeud_apres(ABR ab, double y) : dans le pire des cas le nœud cherché est une feuille de l'arbre donc compléxité en **O(logn)** car la hauteur de l'arbre est en O(logn).

creerFeuille(Segment *seg, Netlist *n) : O(1)
hauteur(ABR ab) : O(1)
majhauteur(ABR ab) : O(1)
rotationDroite(ABR *ab) : O(1)
rotationGauche(ABR *ab) : O(1)
rotations(ABR *ab) : O(1)
insererEInt_avec_eq(ABR *ab, Segment *seg, Netlist *n) : dans le pire des cas on insère l'élément dans les feuilles de l'arbre puis on effectue des rotations pour l'équilibrer, la complexité est alors **O(logn)**.

coupe_max(ABR *ab) : dans le pire des cas le maximum est une feuille de l'arbre donc O(logn).

chercher_noeud(ABR *ab, Segment *seg, Netlist *n) : dans le pire des cas ce nœud est une feuille de l'arbre donc O(logn).
hauteur_abr(ABR a) : cette fonction parcourt tous les nœuds de l'arbre pour calculer leur hauteur et l'arbre possède n nœuds donc O(n).

supprimer_avc_eq(ABR *ab, Segment *seg, Netlist *n) : cette fonction fait appel à hauteur_abr(ABR a) qui est en O(n) donc elle est aussi en **O(n)**. Cependant il est possible de programmer cette fonction en O(logn).

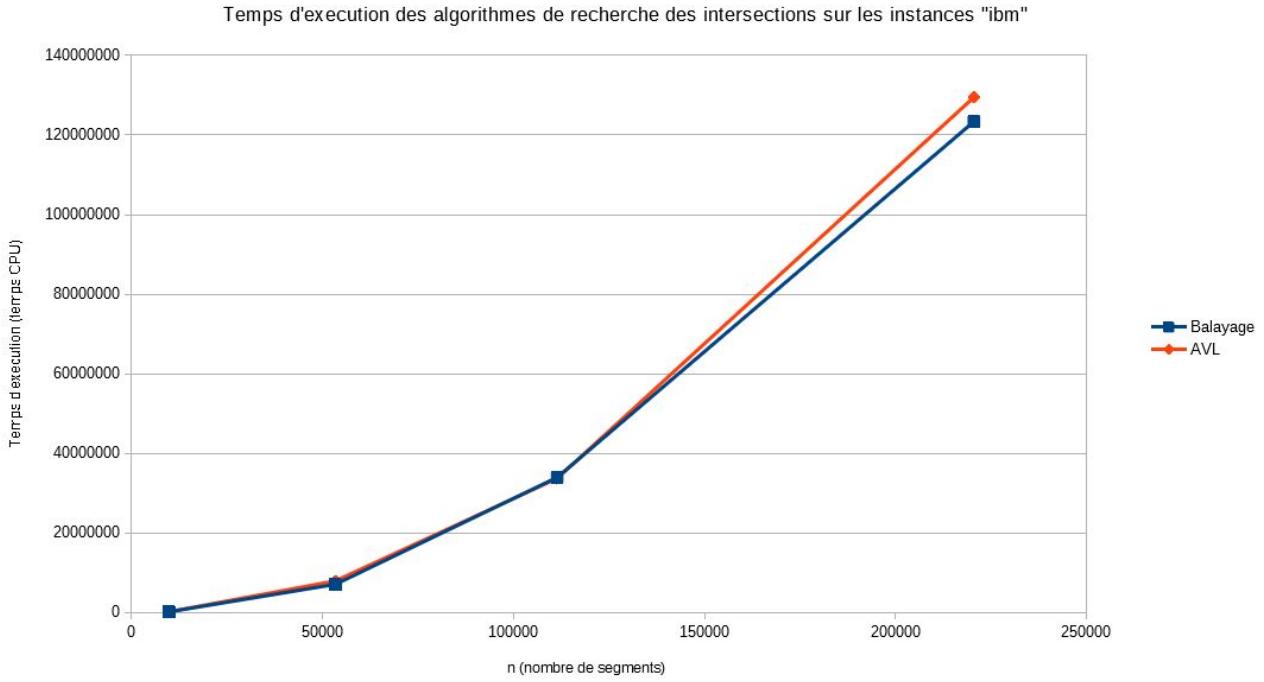
intersec_avl(Netlist *n) :

- E a une taille de l'ordre de O(n) donc la boucle for est O(n) fois.
  - L'insertion est codée en O(logn) et la suppression en O(n).
  - La boucle Tant que itère sur une liste de taille O(n) et appelle ajout_intersection() qui est en O(1) et Prem_Noeud_apres() qui est en O(logn) donc cette boucle est en O(nlogn)

Donc la méthode de recherche des intersections en utilisant les AVL est en **$O(n^2 logn)$.**

En faisant la même hypothèse d'une borne α qu'à la question 3.3) la boucle Tant que est en O(α) et l'algorithme est alors en **O(αn)**.

Q 4.5) Comparaison des temps d'exécution des méthodes par balayage et par AVL :



La méthode utilisant les AVL a un temps d'exécution très proche de la méthode par balayage pour les petites valeurs de n mais semble plus lente pour les grandes valeurs de n malgré l'existence de la borne α.

On en conclut que la méthode par balayage utilisant deux listes chaînées est meilleure que la méthode utilisant les AVL. Et cette différence devient de plus en plus marquée à mesure que le nombre de segments de l'instance augmente.

Partie B

Exercice 5
graphe.c visugraphe.c

La fonction creer_graphe (graphe.c) créée le graphe associé à un fichier int.
Elle prend un paramètre la netlist associée au fichier int et le nom du fichier int. Le graphe est créée en 3 étapes : les sommets, les arretes de conflits, puis les arretes de continuité.

Exercice 6
via_naif.c visuvianaif.c, pas de modification de l'énnoncé

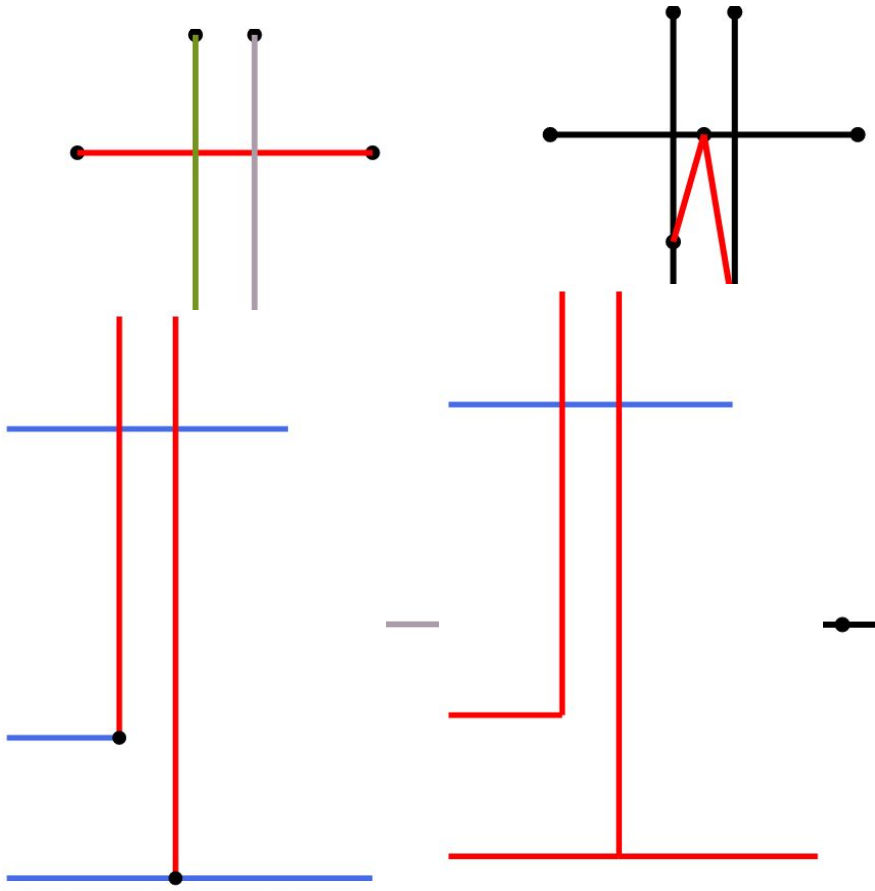
Exercice 7
via_cycle.c visuviacycle.c

La fonction detecte_cycle_impair ne renvoie -1 si aucun cycle impair n'est trouvé et l'indice du sommet où un cycle impair est détecté sinon.
Ceci a permit d'éviter d'avoir un tableau_peres erroné dans le cas où il y a un cycle pair imbriqué dans un cycle pair, on perdait alors trace d'un élément du cycle.
En faisant de cette manière, le tableau des peres n'est remplit uniquement si un cycle impair est trouvé, et ceci lors de la dérecursification.
Q7.5) Complexité de la méthode par recherche des cycles impairs :

Il y aura au moins un appel sur chaque sommet (2 si ce sont des cycles), on parcourt donc le graphe en O(n). Pour chaque cycle trouvé, on réalise une opération de «chaînage» de ces sommets, en O(α) où α est le nombre de sommets du cycle trouvé.
Le coloriage des faces est ensuite réalisé en O(n).

La complexité globale est donc O(αn) soit **$O(n^2)$.**

Visualisations



Visualisation de l'instance test.net et graphe associée à l'instance avec les arrêtes de conflit en rouge

Solution trouvée par l'algorithme naïf et l'algorithme par détection des cycles impairs.