**TP 2 : Job Shop**

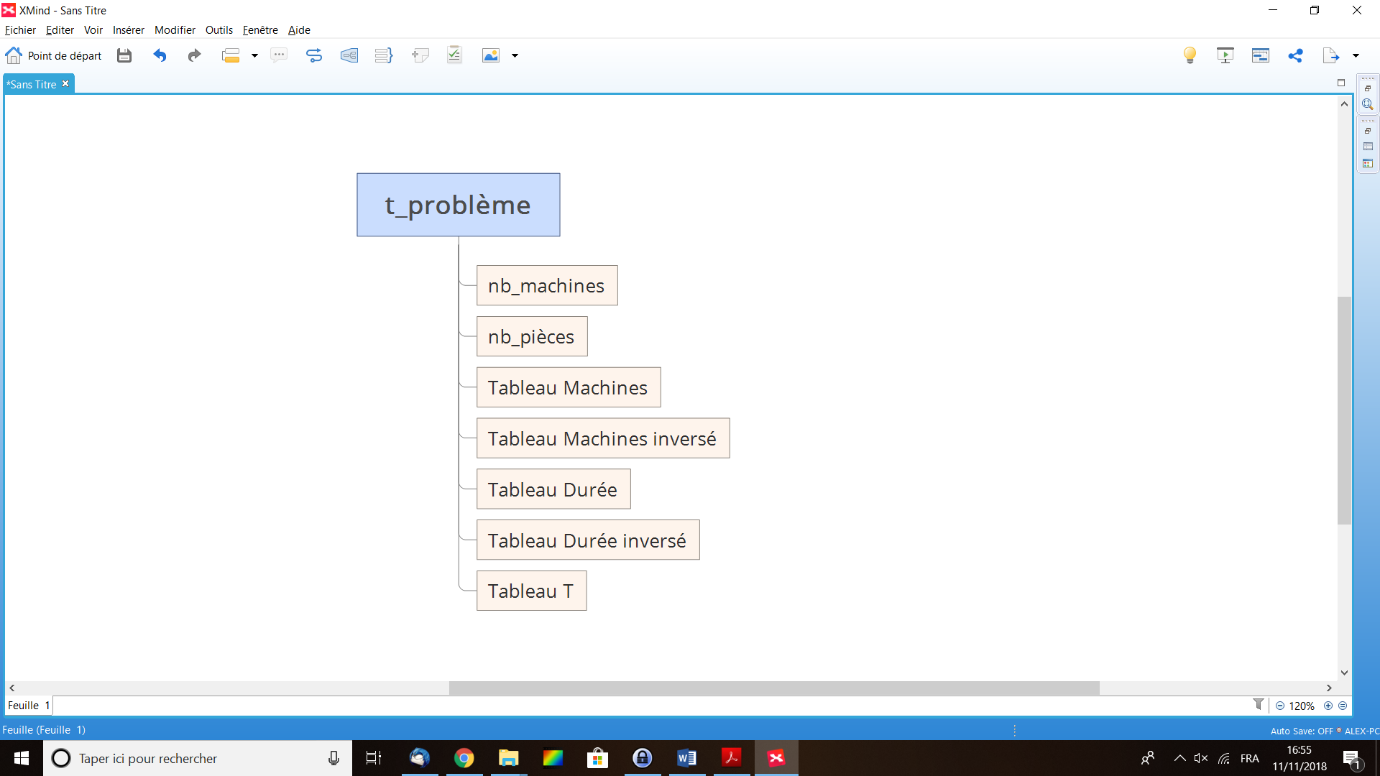
**Description de l’objet du TP**

L’objectif de ce TP est de récupérer des instances classiques d’un problème Job-Shop (lire dans un fichier texte), de les placer dans une structure de données adaptée pour optimiser le traitement. Ensuite, on évalue le graphe trouvé pour calculer le makespan. Ce makespan est amélioré grâce à une recherche locale. Enfin, l’implémentation d’un algorithme génétique optimise la recherche globale.

**Schémas des structures de données et des fichiers de données utilisées :**

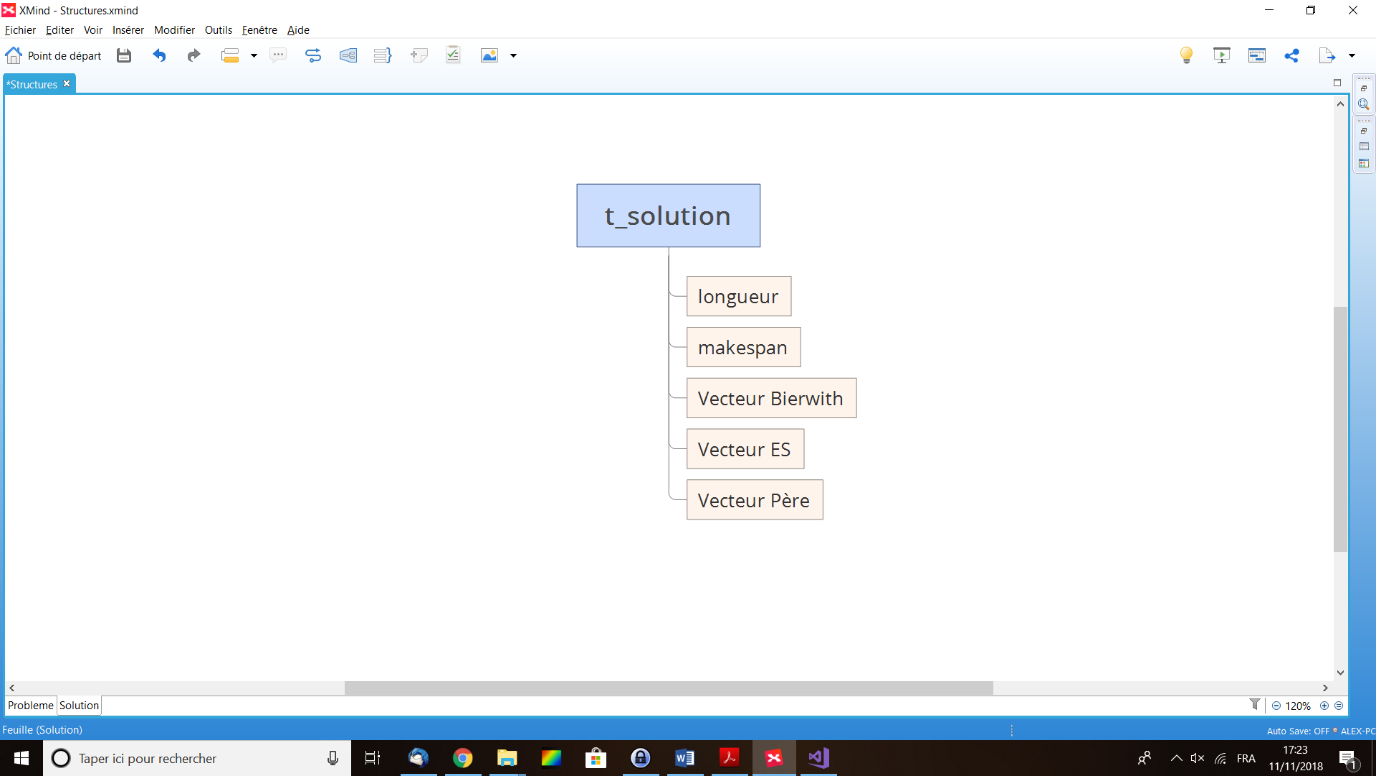
**Note**: Tous les tableaux utilisés sont surdimensionnés afin de se défaire de tous problèmes de taille.

**Structure du problème (graphe) :**

Cette structure servira à stocker :

1. : nombre des machines du problème
2. : nombre de Jobs (pièces)
3. : matrice où chaque ligne représente un numéro de job, et contient la suite des numéros de machines pour chaque Job.
4. : matrice qui renseigne l’ordre de passage de chaque machine pour chaque job
   * ;
5. : matrice des *processing times* (durée des opérations)
6. : matrice de durée
7. : matrice de position dans le tableau des Earliest Starting Time
   * ; position dans ES.

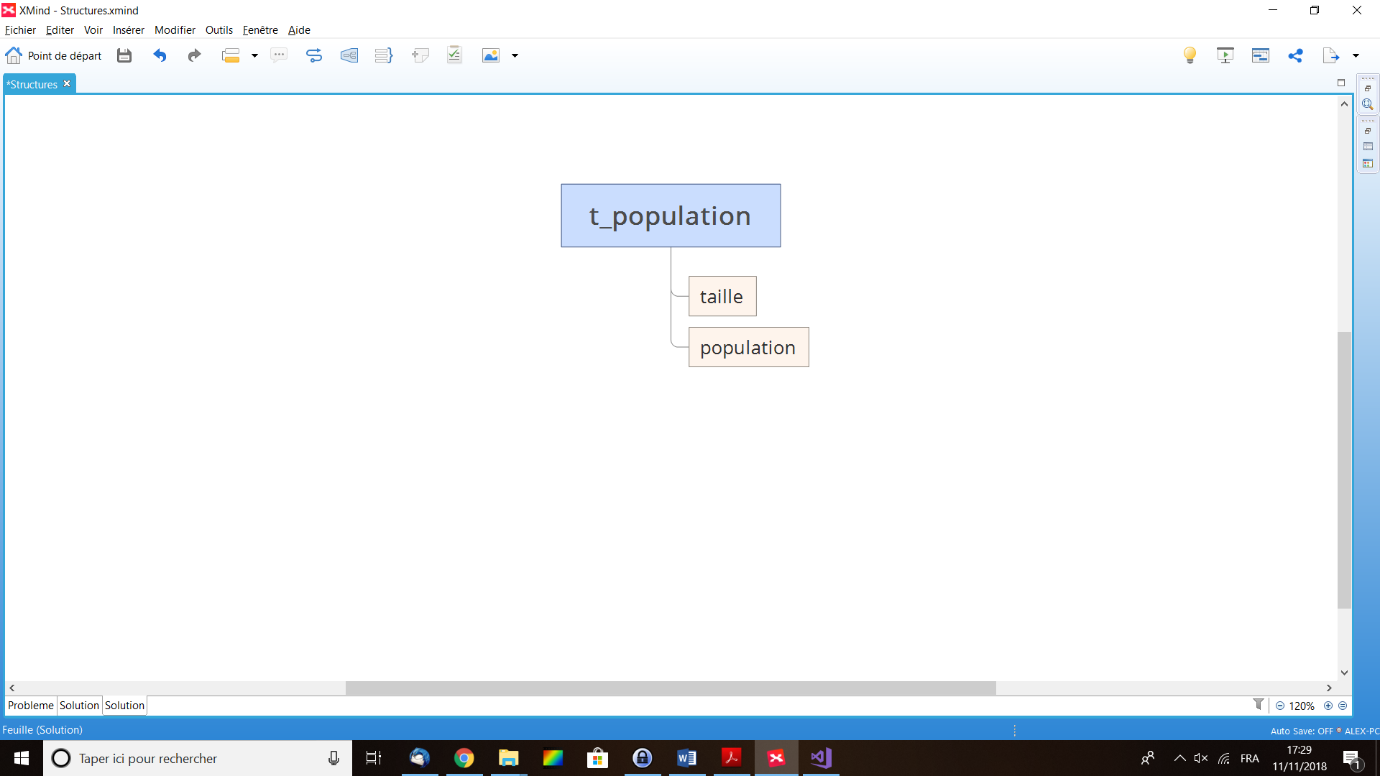
**Structure de la solution :**



Cette structure servira à stocker :

1. : nombre d’opérations total
2. : durée totale problème
3. : ordre de passage sur une machine
4. : tableau des temps de début au plus tôt (*earliest starting time*) pour chaque opération
5. : tableau donnant l'antécédent de chaque opération pour le chemin critique

**Structure de la population :**



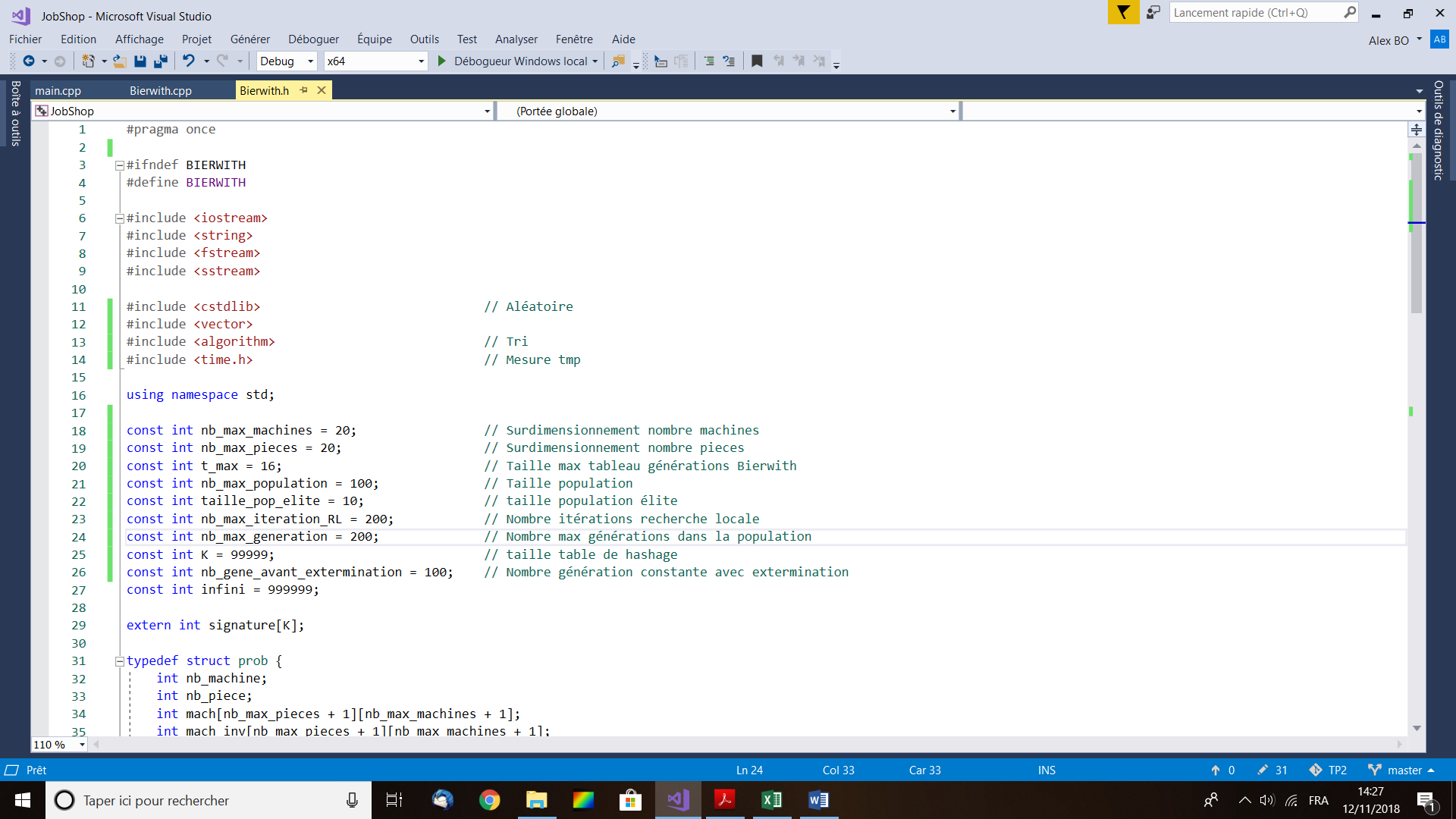
Cette structure servira à stocker :

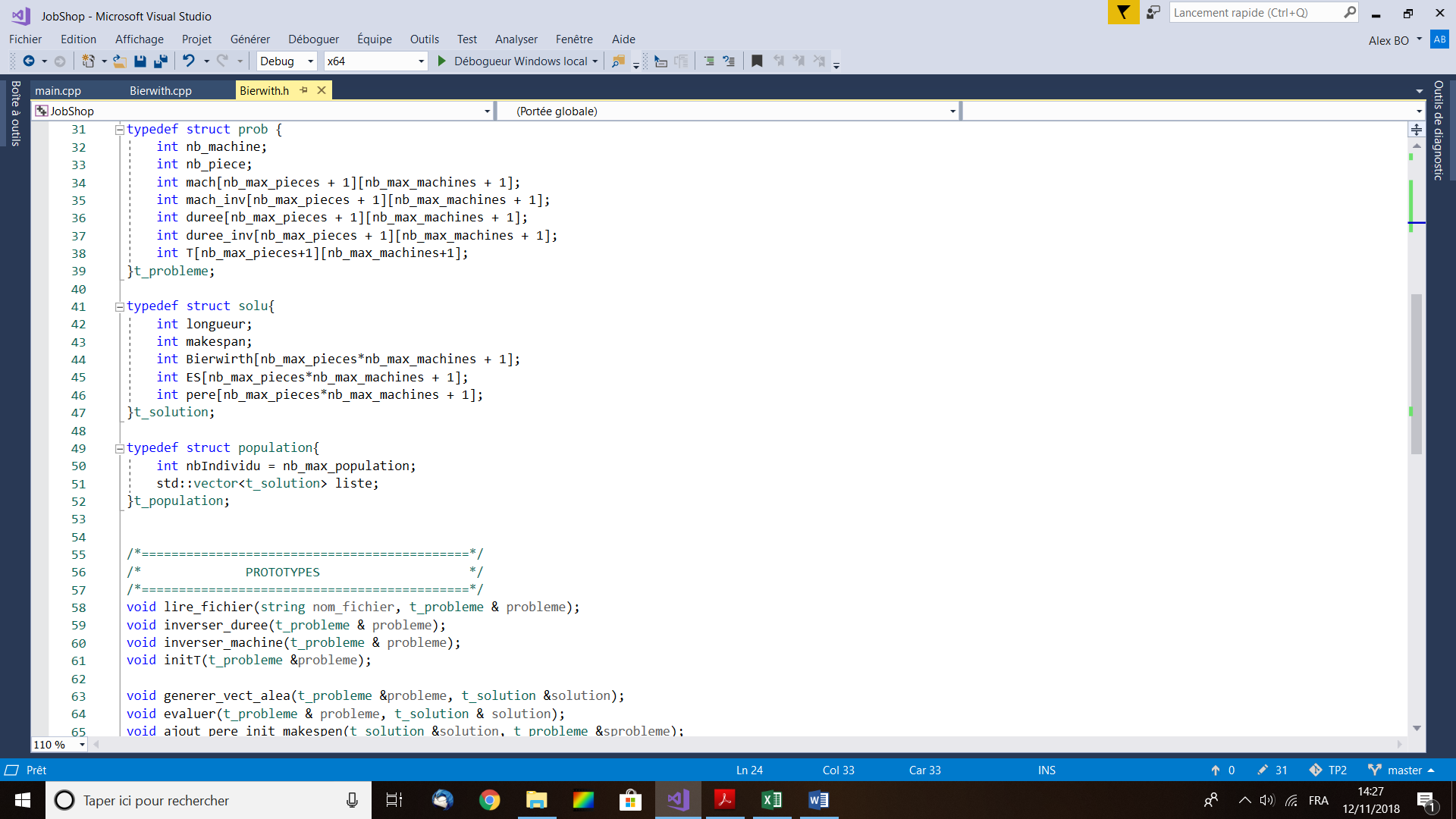
1. : taille de la population
2. : ensemble de solutions du problème

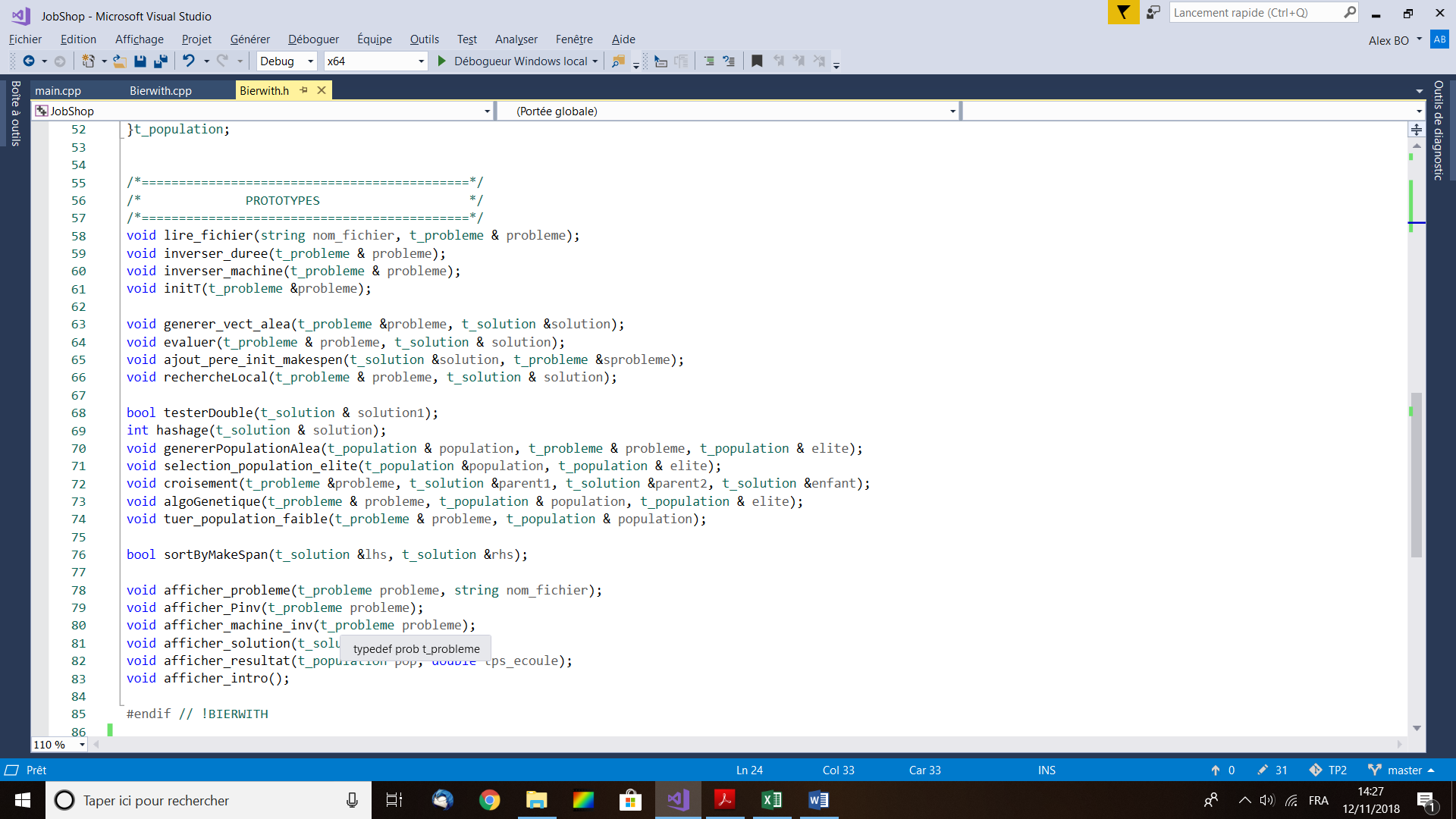
En outre, on utilisera un tableau de taille *K = 99999*, nommé *signature* afin de pouvoir mémoriser si une solution a déjà été considérée.

**Organisation et code source :**

Détail fichier Bierwirth.h contenant les structures et les prototypes :







**Description fonctions principales :**

* **ÉVALUER** :

Objectif: Evaluer une solution (problème + vecteur de Bierwirth associé).

Argument :

* Entrée :
  + Structure du problème (graphe)
  + Un vecteur de Bierwirth, un vecteur par répétition, généré aléatoirement, de taille nombreDeMachine \* nombreDeJobs, où V[i] = numéroDeJob.
* Sortie :
  + *Makespan*
  + Tableaux *ES* (date au plus tôt)
  + Tableau *père* : chemin critique.

Principe :

Pour chaque opération :

Si l’opération est précédée par une autre opération (même job) :

MAJ date de commencement : max (date de commencement + durée opération)

Si l’opération est précédée par une autre opération (même machine) :

MAJ date de commencement (date de commencement + durée opération)

Difficulté : pour une machine i donnée, remonter à la dernière opération effectuée sur cette machine. Utilisation de *Machine\_inv* et du tableau *T* afin de retrouver la dernière opération sur cette machine et sa place dans le vecteur de Bierwirth.

* **RECHERCHE LOCALE** :

Objectif : Etant donné une solution, optimiser cette solution par modification du vecteur de Bierwirth.

Argument :

* Entrée :
  + Un problème (graphe)
  + Une solution du type *t\_solution*, contenant un vecteur de Bierwirth.
* Sortie :
  + Une solution meilleure (ou au moins égale) à la solution initiale.

Principe :

* Tant que le nombre d'itération maximale n’est pas atteint :
  + Sélection de 2 sommets appartenant aux chemin critique en partant de la fin
  + Si on se trouve sur un arc disjonctif
    - Permutation de ces 2 sommets
    - Evaluation nouveau vecteur de Bierwirth
    - Si le nouveau makespan est meilleur :
      * Conservation nouvelle solution
      * Redémarre de l’extrémité du chemin critique
    - Sinon :
      * On continue à remonter le chemin critique
* **TESTER DOUBLE :**

Objectif : Eviter de re-travailler sur des solutions déjà explorées.

Argument :

* Entrée : Une solution
* Sortie : un booléen

Principe : este si cette solution a déjà été considérée. Pour cela, une fonction de hachage vérifie que la signature de la solution :

n’est pas présente dans le tableau des signature.

Si la case à l’indice h du tableau signature vaut 1, cela veut dire que cette solution a déjà été traitée. La fonction renvoie un booléen.

* **ALGORITHME CROISEMENT :**

Objectif : générer un nouvel élément "*fils*" héritant des propriétés des deux parents : le premier appartient aux bons éléments de la population, le second aux moins bons.

Arguments :

* Entrée :
  + Le *problème*
  + Un *parent* dit "bon"
  + Un *parent* dit "mauvais"
  + Un *enfant*
* Sortie :
  + Un enfant héritant des propriétés des deux parents.

Principe :

Tirage au hasard d’un point de coupure dans le vecteur parent

Copie dans le vecteur de *Bierwith* Fils des éléments du bon parents qu’au point de coupure

Remplissage de la seconde moitié du vecteur fils à partir du vecteur *parent mauvais* :

Si l’indice courant est ajoutable (respecte les propriétés du vecteur de Bierwith) :

Ajout dans *fils*

Passage à l’indice suivant

Remarque : cette méthode assure la validité du vecteur fils ainsi créé (solution conforme).

* **ALGORITHME GÉNÉTIQUE :**

L’objectif est de générer des populations afin d'améliorer le makespan à travers les différentes générations créées par croisements entre de “bonnes solutions” et des “mauvaises solutions”. Il permet, en complément avec une recherche locale, d’effectuer une recherche plus générale voir globale.

Arguments :

* Entrée :
  + Le *problème*
  + La population *initiale*
  + La population *elite.*
* Sortie :
  + Une population dont le premier individu est la meilleure solution trouvée au problème.

Principe :

Génération d’une population aléatoire initiale.

Tant que le nombre de générations maximal n’est pas atteint :

Tri population par makespan

Sélection de la population *elite.*

Pour chaque individu de la population mauvaise (90%) :

Tirage au hasard d’un bon père (10er %)

Génération par croisement du père bon et du mauvais d’un *enfant*

Recherche locale sur la solution enfant.

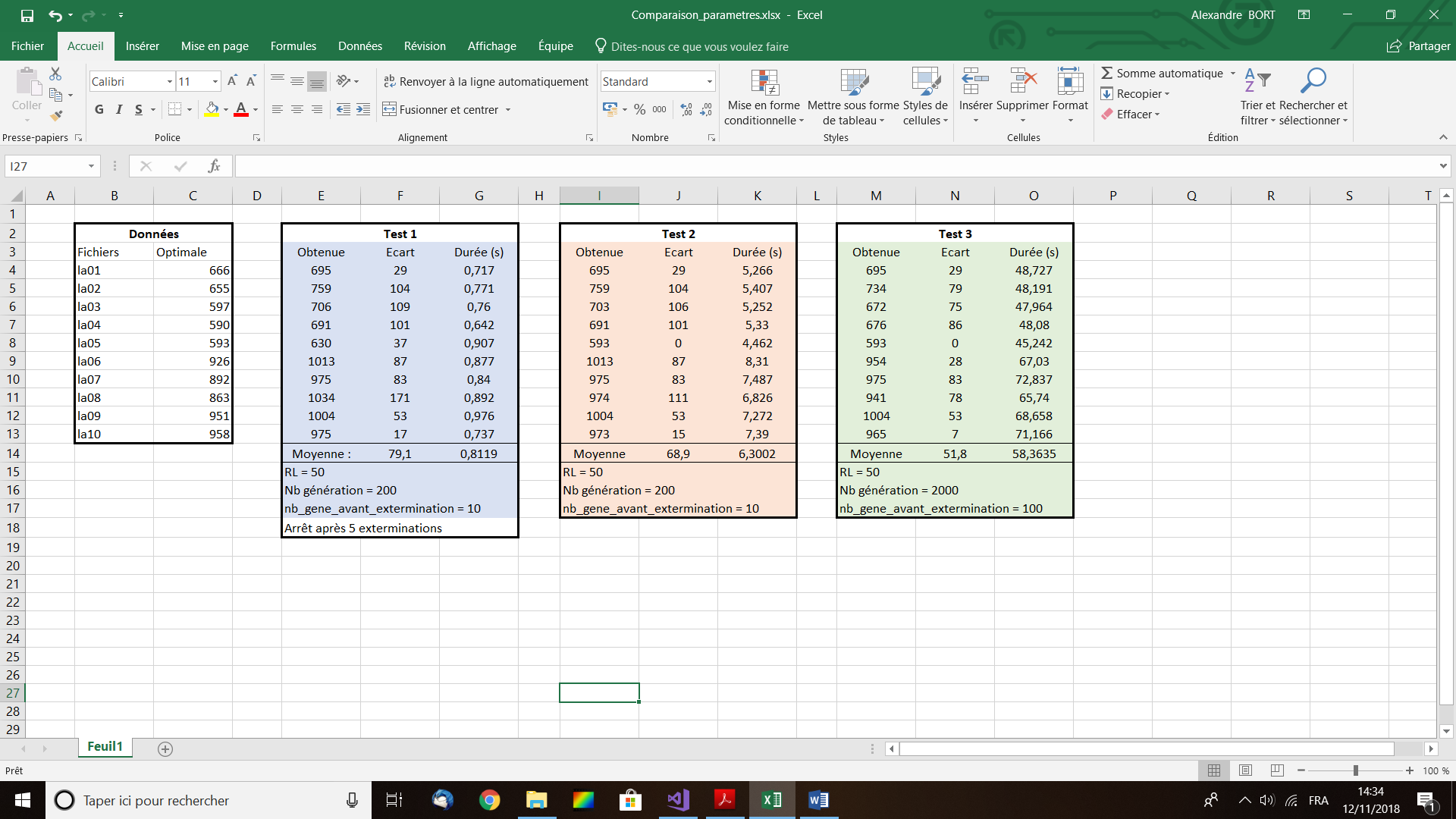
Tirage au sort d’un des parents et remplacement par l’enfant.

Si après un certain nombre d’itérations, le makespan ne s'améliore pas :

Extermination des 90% mauvais de la population

Régénération aléatoire de ces 90%.

**Tests :**



**Piste d’amélioration :**

* **Optimiser la fonction évaluer :**
  + Définir un point de départ de la fonction évaluer afin d’éviter de recalculer l’ensemble de la solution. En effet, une modification du vecteur de Bierwith ne modifie pas la partie à gauche de la première modification.
  + Sauvegarder les différentes instances dans une pile.
  + Plus de point de départ est grand, plus cette solution est intéressante.
* Tester d’autres méthodes de croisement
* Vérifier la diversité de la population :
  + Solution exogène :
    - Intégrer dans chaque population quelques nouveaux individus tirés aléatoirement (immigration).
  + Solution endogène :
    - Définir une distance entre les vecteurs de Bierwith et instaurer un critère fonction du makespan et de la distance.
    - Ex : avec qui décroit au fur et à mesure des itérations. Cela permet de conserver une certaine diversité au fur et à mesure des générations.