

Projet Picture

Isabelle Angeletti – Xavier Maupeu

11 Décembre 2013

Première partie

Présentation du Projet

1 Définition et cadre du projet

Nous avons réalisé le projet Picture dans le cadre du cours d'Algorithmie en première année d'Informatique à l'Enseirb-Matmeca. Ce projet avait pour objectif de mettre en pratique les différentes connaissances obtenues au cours du premier semestre.

Le projet Picture consistait à implémenter la réalisation et la résolution de grilles de nonograms. Il fallait aussi attribuer un niveau de difficulté à chaque grille et indiquer si la grille possédait une ou plusieurs solutions.

2 Analyse du projet

La principale difficulté du projet Picture est la mise en place d'un algorithme de résolution des grilles. Il faut en effet pouvoir résoudre des grilles de toutes tailles en un temps limité. Pour réaliser ce projet, il faut aussi pouvoir charger et lire des grilles de nonograms dans notre programme. Nous devons ensuite stocker ces grilles, qui peuvent être de tailles différentes, dans une structure adaptée. Un autre problème est donc le choix de la structure à utiliser pour le stockage de la grille et de ses données.

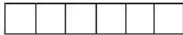
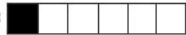


3 Conception de la solution algorithmique

3.1 Résolution grossière des grilles

Pour concevoir l'algorithme de résolution des grilles, nous avons commencé par analyser la résolution d'une grille à la main pour essayer de trouver quels algorithmes sont utilisés lors de la recherche de la solution.

Nous avons décidé de travailler sur la résolution particulière d'une seule ligne et d'une seule colonne puis d'appliquer nos différents algorithmes à l'ensemble des lignes et colonnes de la grille.

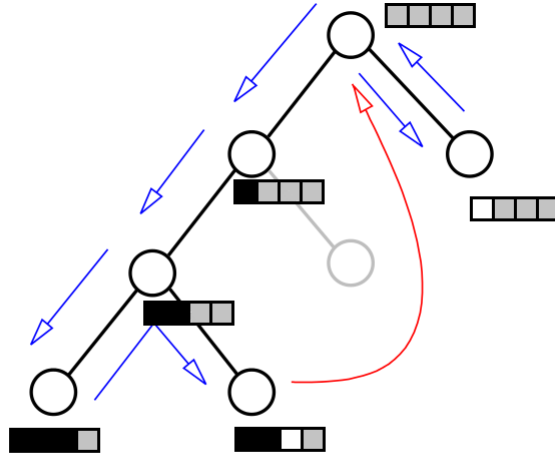
Nous avons commencé par traiter les cas simples de résolution pour une ligne :

- ligne vide ou entièrement noircie, ⁰ 
- un des bords est déjà noirci, ³ 
- un des nombres de données est supérieur à la moitié de la largeur de la grille, ⁴ 
- la somme des données, en comptant les espaces entre chaque donnée, est égale à la largeur de la grille. ^{1 1 2} 

Nous avons ensuite implémenté ces algorithmes pour les lignes, et les avons ensuite transposés pour les colonnes.

3.2 Backtracking

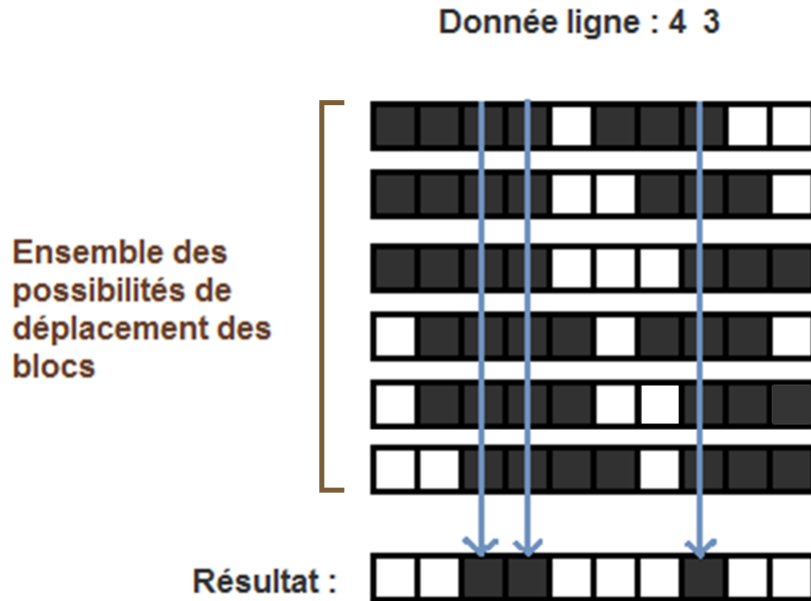
Ces simples algorithmes ne suffisait pas pour résoudre entièrement une grille, en effet il restait encore un nombre important de cases indéterminées, nous avons donc décidé d'appliquer un algorithme basé sur le backtracking aux cases qui restaient indéterminées. Le backtracking consiste à essayer un bloc et à continuer uniquement si la solution respecte les données de la grille. Si le bloc blanc et le bloc noir ne fonctionnent pas on revient en arrière.



Le principal problème de cet algorithme est que sa complexité est exponentielle dans le pire des cas. Donc plus la grille à résoudre est grande, plus il va rester de cases indéterminées et donc plus le backtracking va mettre de temps pour trouver la solution.

3.3 Détermination des blocs noirs par intersection

Afin d'améliorer le temps pris par notre algorithme pour résoudre les grilles, nous avons réfléchi aux autres algorithmes que nous pouvions implémenter. Un des algorithmes que nous avons trouvé calcule toutes les possibilités pour une ligne et ne retient que l'intersection de ces possibilités.



Cette fonction nous permet d'augmenter le nombre de cases noires trouvées sans utiliser le backtracking. Cependant, même combinée à nos autres algorithmes, la résolution des grilles de taille élevée reste compliquée.

3.4 Chargement des grilles

Un autre des problèmes de ce projet était le chargement des grilles de Picture dans notre programme. Une fois le format de fichier contenant la description de la grille défini pour l'ensemble de la promotion, nous avons mis en place un algorithme capable de récupérer les données de ce fichier.

Comme vous pouvez le voir sur l'image ci-dessous, le fichier est constitué de trois lignes :

- une pour la largeur et la hauteur de la grille
- la deuxième contient les nombres inscrits à droite de la grille
- la dernière contient les nombres inscrits en haut de la grille

Notre algorithme va récupérer ces lignes une par une. Pour la première ligne aucun traitement n'est nécessaire. Nous appliquons le même algorithme sur les deux autres lignes. Cet algorithme va parcourir la ligne et va en extraire les données de la grille grâce à l'analyse des séparateurs. Si un espace est lu, le nombre

coeur.txt

```
7 7
2 2;7;2 4;7;5;3;1
3;5;2 3;6;6;5;3
```

correspond toujours à la même ligne ; si un point-virgule est lu, nous changeons de ligne.

Nous avons ensuite voulu implémenter le chargement des grilles à partir d'images afin de pouvoir créer de nouvelles grilles. Pour ce faire, nous avons réalisé un autre algorithme qui parcourt l'image, récupère la couleur des pixels et stocke l'ensemble de ces valeurs dans un tableau. Notre algorithme analyse ensuite ce tableau pour en récupérer les données de la grille.

Notre algorithme est efficace, mais il permet seulement de charger des images au format bmp, et de taille inférieure à 20x20 pixels.

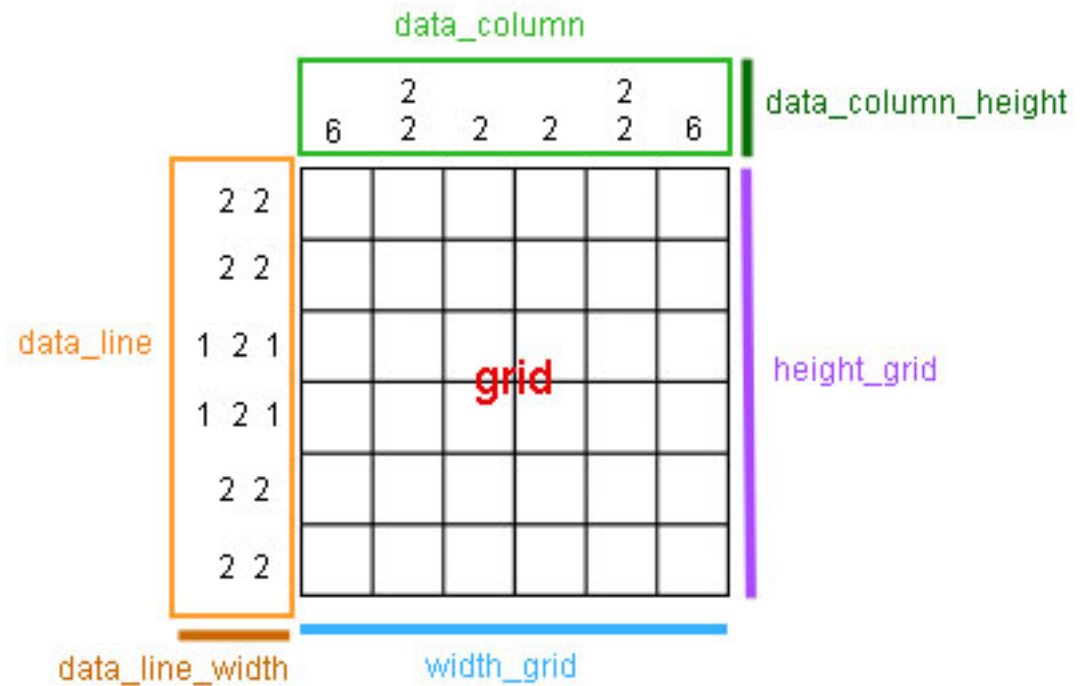
Deuxième partie

Réalisation du projet

4 Structures de données

4.1 Stockage de la grille

Pour stocker notre grille après son chargement dans notre programme, nous avons décidé d'utiliser une structure de données. Cette structure est composée de 7 éléments qui correspondent aux différents éléments indiqués sur le schéma.



Pour les trois tableaux nous avons décidé d'utiliser des tableaux 2D alloués dynamiquement car leurs tailles doivent pouvoir varier en fonction de la taille spécifiée par l'utilisateur :


```

1  // Allocation grid
2  f_picture->grid = (block**) malloc(f_picture->width_grid * sizeof(block
   *));
3
4  if (f_picture->grid == NULL)
5  {
6      exit(EXIT_FAILURE);
7  }
8
9  for (int i = 0; i < f_picture->width_grid; i++)
10 {
11     f_picture->grid[i] = (block*) malloc(f_picture->height_grid *
        sizeof(block));
12
13     if (f_picture->grid[i] == NULL)
14     {
15         exit(EXIT_FAILURE);
16     }
17
18     // init grid
19     for (int j = 0; j < f_picture->height_grid; j++)
20     {
21         f_picture->grid[i][j] = UNDEFINED;
22     }
23 }

```

4.2 Cases de la grille

Afin de faciliter la gestion des cases de notre grille, nous avons créé une énumération de type pour chaque couleur que peut prendre une case.

```

1  typedef enum
2  {
3      BLACK, WHITE, UNDEFINED
4  } block

```

Au chargement de la grille, toutes les cases sont initialisées à UNDEFINED. Ensuite, notre algorithme de résolution va modifier la valeur des cases en la mettant soit à WHITE soit à BLACK. Suivant la valeur des cases elles seront affichées blanches, noires ou grises, respectivement pour WHITE, BLACK et UNDEFINED.

5 Organisation et répartition du travail

Pour réaliser le projet Picture, nous nous sommes répartis les tâches en découpant le projet en différentes fonctions :

- Le chargement des grilles :
 - Depuis un fichier
 - Depuis une image
- La résolution des grilles :
 - Résolution des lignes et des colonnes
 - Backtracking
- L’affichage de la grille
- Le calcul de la difficulté de la grille

Nous avons ensuite travaillé chacun sur des fonctions différentes tout en mettant notre travail en commun régulièrement et en nous aidant mutuellement.

Comme nous sommes très vite tombés d’accord sur la répartition des tâches, nous n’avons eu aucun soucis au niveau de l’organisation du projet.

6 Implémentation des différentes fonctions

6.1 Résolution des lignes et des colonnes

Nous avons commencé par implémenter la résolution d’une ligne. Nous avons d’abord écrit les différentes fonctions correspondant aux cas simples de résolution. Les premiers cas que nous avons traité sont ceux correspondant à une ligne vide ou à une ligne pleine. Nous avons ensuite créé une fonction qui calcule la somme des données d’une ligne, une autre qui va retourner la dernière donnée réelle de la ligne. En effet, nos données sont stockées dans un tableau de taille fixe et la dernière donnée ne correspond pas forcément à la dernière colonne. Nous avons également codé un algorithme qui va compléter le bloc de donnée si la première ou la dernière case de la ligne est remplie. Cet algorithme va aussi ajouter un espace après le bloc. Pour ce faire, nous vérifions d’abord si la première ou la dernière case est remplie. Si c’est la cas, nous complétons le bloc de cases noires avec la donnée correspondante, dans le cas où la dernière case est noircie il s’agit de la dernière donnée réelle de la ligne.

Nous avons ensuite travaillé sur l’algorithme pour remplir les cases lorsqu’il existe une donnée supérieure à la moitié de la largeur de la grille. Nous avons pour cela calculé l’intervalle des cases à noircir. Par exemple : pour une grille de taille 10, si la donnée est égale à 7 il faut remplir les cases allant de 4 à 7. Cet algorithme a ensuite été amélioré pour calculer toutes les intersections possibles pour les différents blocs de cases d’une même ligne.

Pour cela on a besoin de créer un tableau contenant tous les déplacements possibles ici appelé `_tab_shift`, par exemple :


```

1  while(_tab_shift[_nb_range -1] <= _shift_max)
2  {
3      int _sum = 0;
4
5      for (int i = 0; i < _nb_range - 1; i++)
6      {
7          if (_tab_shift[i] > _shift_max)
8          {
9              _tab_shift[i] = 0;
10             _tab_shift[i+1] ++;
11         }
12
13         _sum += _tab_shift[i];
14     }
15
16     // we can keep _tab_shift only if the sum of shift is less than shift_max
17     if (_sum + _tab_shift[_nb_range -1] > _shift_max)
18     {
19         // increment shift
20         _tab_shift[0] ++;
21
22         continue;
23     }
24
25     // intersect
26     ...
27
28     // increment shift
29     _tab_shift[0] ++;
30 }

```

Afin de savoir si une ligne est résolue ou non, nous avons défini deux fonctions. La première fonction parcourt la ligne et vérifie si les blocs de cases noires présents dans la ligne correspondent aux données inscrites à gauche de la grille. Ainsi si le nombre de cases noires remplies dépasse le nombre de cases indiqué dans les données, la ligne n'est pas valide. Au contraire, si la ligne est valide il suffit de changer en blanc la couleur des cases indéterminées.

L'autre fonction compte le nombre de cases noires ou indéfinies et le compare à la somme des données. Si ces deux nombres sont égaux, cela signifie que la ligne peut être complétée. Cette fonction permet de savoir si le nombre de cases blanches sur la ligne est suffisant, si c'est la cas il faut juste noircir les cases indéterminées.

Nous avons ensuite transposé ces différents algorithmes pour résoudre une colonne. En général, il suffisait de remplacer les parcours dans les boucles, parfois cela demandait un peu plus de réflexion.

6.2 Chargement des grilles

En ce qui concerne le chargement des grilles, nous avons d'abord implémenté le chargement depuis un fichier. Il nous a donc fallu chercher les fonctions en langage C permettant de lire dans un fichier. Après avoir récupéré les données dans le fichier, notre algorithme va les traiter pour les insérer au bon endroit dans notre structure de grille. Ainsi, les deux premiers nombres du fichier iront dans les variables `width_grid` et `height_grid`. Les deux autres lignes subissent d'abord un traitement avant de remplir les tableaux de données de la grille.

Plus tard, nous avons décidé de créer des grilles à partir d'images. Nous avons donc écrit plusieurs fonctions permettant de convertir l'image en grille. Pour commencer, une des fonctions va récupérer les différentes valeurs des pixels et les stocker dans un tableau de pixels. Une autre fonction permet de déterminer si le pixel est plutôt de couleur blanche ou de couleur noire : nous additionons les valeurs RGB du pixel et les comparons à une valeur type. Nous avons considéré que les couleurs dont la somme des composantes RGB est supérieure à 450 correspondent à des couleurs plutôt proches du blanc, les autres seront donc traitées comme du noir.

Nous parcourons ensuite notre tableau de pixels et comptons les différents blocs de couleur noire pour obtenir les données des lignes et celles des colonnes. Nous stockons ensuite ces données dans nos tableaux `data_line` et `data_column` et nous obtenons notre grille.

6.3 Difficulté de la grille

Dans le projet Picture, il nous était aussi demandé de donner une difficulté à chaque grille. Pour déterminer cette difficulté, nous avons réalisé un algorithme qui va comparer la taille de la grille au nombre de cases à noircir. Pour ce faire, nous parcourons un des tableaux de données et additionons chaque donnée. Une fois le parcours terminé, nous comparons ce nombre à la taille de la grille, c'est-à-dire la largeur multipliée par la hauteur. Suivant la valeur du ratio obtenu, nous considérons que notre grille est de difficulté facile, moyenne ou difficile.

7 Correction et complexité des algorithmes

Les différents algorithmes mis en place pour la résolution d'une ligne ou d'une colonne sont en général de complexité linéaire. En effet, pour la plupart de ces algorithmes il faut parcourir soit la grille soit les tableaux de données. Comme pour résoudre notre grille, nous appelons ces différents algorithmes sur toutes les lignes et toutes les colonnes, au final la complexité au niveau de la résolution est polynomiale.

L'algorithme de backtracking va tester tous les emplacements possibles pour chaque bloc de données. Sa complexité est donc exponentielle par rapport aux nombres de cases indéterminées.

Afin de réduire la complexité de résolution des grilles, nous aurions peut-être pu mettre au point d'autres fonctions de résolution. Nous avons pensé à implémenter une fonction qui permettrait d'assembler ou de séparer deux blocs de données. Cependant, par manque de temps et de solution algorithmique efficace nous n'avons pas implémenté cette fonction.

Au final, notre algorithme complet de résolution d'une grille ne permet pas de résoudre dans un temps limité les grilles de taille trop élevée. En effet, sa complexité étant exponentielle, il met un temps trop important pour résoudre la grille.

8 Problèmes de mise en œuvre et tests de validité

8.1 Problèmes rencontrés

Nous n'avons pas eu beaucoup de problèmes au niveau de la réalisation de notre projet.

Un des problèmes rencontré venait d'une inversion lors de la lecture de la grille. Nous chargions la hauteur à la place de la largeur et donc lors de l'allocation de nos tableaux de données une erreur se produisait. Ce problème a vite été résolu.

8.2 Tests de validité

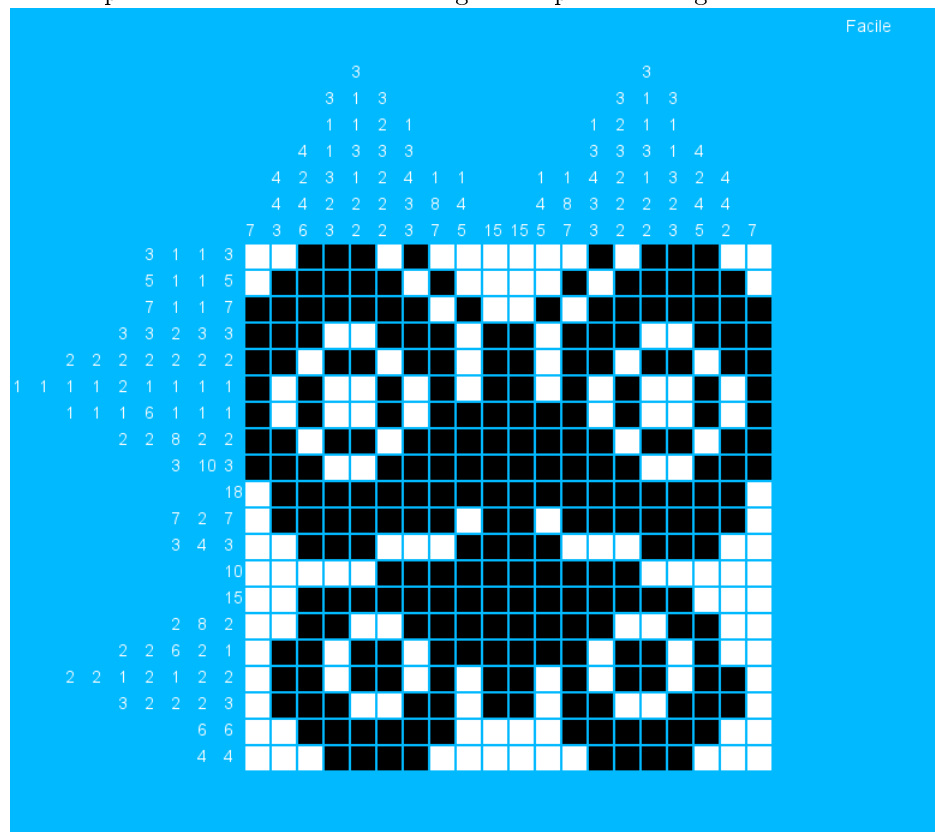
Dans le but de vérifier le bon fonctionnement de notre programme, nous avons testé de nombreuses grilles. Nous avons testé des grilles de tailles variées afin de déterminer la limite de fonctionnement de notre programme. Nous nous sommes rendus compte qu'au delà des grilles de dimension 25x25, notre programme n'arrivait plus à résoudre les grilles en temps limité.

Pour la création de nouvelles grilles, nous avons testé plusieurs images. Les grilles obtenues après résolution restent fidèles à l'image d'origine. On peut cependant observer quelques différences causées par le redimensionnement ou la pixelisation de l'image.

9 Résultats obtenus

Au début, nous utilisions le terminal pour visualiser nos grilles, nous avons ensuite décidé d'utiliser la librairie SDL car nous voulions obtenir un programme plus visuel et que nous avions déjà tout les deux utilisé cette bibliothèque. Grâce à l'utilisation de cette librairie, notre programme est très visuel

et nous permet de créer des nouvelles grilles à partir d'images.



Au niveau des différentes fonctionnalités, nous avons fourni un programme répondant aux attentes du sujet du projet. Notre programme affiche les grilles et leur difficulté, et résout les grilles. La seule fonction que nous n'avons pas eu le temps d'implémenter est celle permettant de déterminer si une grille possède une ou plusieurs solutions.

Nous avons aussi un chargement de grille à partir de fichier qui fonctionne ainsi qu'une création de grilles à partir d'images que nous avons rajouté.

10 Conclusion

Au niveau du travail réalisé nous sommes satisfait du programme que nous avons produit. Nous aurions cependant bien aimé aller plus loin dans la réalisation en améliorant notre algorithme de résolution et en implémentant des grilles en couleur.

Ce projet nous a permis de mettre en pratique les différents enseignements du premier semestre. C'était un projet très intéressant aussi bien au niveau algorithmique qu'au niveau programmation.