Complétion de grilles de mots croisés

Sujet proposé par Didier Rémy

Didier.Remy@inria.fr

(Page de suivi)

1 Détail du sujet

Le projet consiste à réaliser un programme qui énumère les solutions d'une grille de mots croisés partiellement remplie, sans tenir compte des définitions.

Une grille de mots-croisés est décrite par une matrice $p \times q$ dont les éléments sont des cases blanches ou noires et par la donnée d'une définition pour chaque sous-colonne ou sous-ligne de cases blanches maximale. Les définitions sont habituellement formulées en langue naturelle et ajoutent des contraintes sémantiques aux contraintes géométriques (être un mot du dictionnaire de la bonne taille). Ici, nous ignorerons totalement les contraintes sémantiques et ne retiendrons que les contraintes géométriques.

Une grille $p \times q$ sera donnée dans un fichier ASCII contenant exactement p lignes de q caractères (sans compter les retour-à-la-ligne). Les cases blanches sont représentées par le caractère 1 et les cases noires par le caractère 0. Nous considérerons également des grilles partiellement remplies où certaines cases sont des lettres.

Un dictionnaire est donnée par une suite de mots séparés par un retour à la ligne (le caractère `\n`). On se limitera à un alphabet ne comportant que les lettres de `a` à `z`.

Un problème est donc entièrement déterminé par la donnée d'un dictionnaire et d'une grille partiellement remplie.

Une solution est une grille identique à l'énoncé sauf ses cases blanches qui auront été toutes remplacées par des lettres telles que toutes les définitions soient satisfaites, *i.e.* chaque sous-mot vertical ou horizontal de longueur maximale soit dans le dictionnaire.

Une réponse à une grille est l'ensemble des solutions de la grille, qui seront présentées comme dans les grilles données en énoncé, mais affichées les unes à la suite des autres et séparées par une ligne blanche.

Exemple

Par exemple, voici ci-dessous a) un dictionnaire à cinq mots, b) une grille à remplir et c) une solution pour la grille (b) avec le dictionnaire (a) :

(a)	(b)	(c)
`mars`	`1111`	`mois`
`mois`	`1001`	`a00o`
`plus`	`1001`	`r00i`
`tard`	`1111`	`soir`
`soir`		

On pourra utiliser le dictionnaire plus étoffé francais.dict.gz fourni, qui comporte presque tous les mots de la langue française avec leurs déclinaisons.

Évaluation

Votre programme sera évalué en considérant d'abord sa correction : est-ce qu'il trouve bien toutes les solutions ? Puis sa robustesse : est-ce qu'il peut traiter des problèmes compliqués sans «casser» (par exemple par manque de mémoire) ? Enfin, on prendra en compte son efficacité : met-il un temps raisonnable pour résoudre différents types de problèmes ? La rapidité n'est pas significative à un facteur 2 près, mais un ou deux ordres de grandeur feront la différence ! Bien sûr, on considérera également l'originalité et l'élégance de la solution (simplicité et généralité au regard des performances).

2 Quelques indications

Il existe plusieurs approches possibles pour attaquer ce problème et il n'est pas évident de deviner laquelle sera la plus performante. On se contentera d'une solution «raisonnable» qui, nous ne verrons, n'est certainement pas la meilleure. D'autres solutions sont décrites dans la littérature.

Abstraction du problème

La grille qui détermine le problème possède des contraintes géométriques dont on voudrait s'abstraire. Numérotons les cases de la grille $(i, j) \mapsto k(i, j)$. Pour l'instant nous supposons la numérotation quelconque, mais nous imposerons des contraintes par la suite. Réexaminons le problème sous sa forme linéarisée en repérant une case par son numéro k.

Chaque case appartient à la fois à un dictionnaire horizontal $h(k)$ et à un dictionnaire vertical $v(k)$ (on pourra traiter les cas pathologiques séparément ou bien leur associer des dictionnaires triviaux). De plus, chaque case est à une position $p_h(k)$ dans le dictionnaire $h(k)$ et à une position $p_v(k)$ dans le dictionnaire $v(k)$.

On pourrait chercher les solutions en énumérant toutes les combinaisons de mots des dictionnaires, puis ne retenir que les combinaisons telles que pour chaque case k la lettre $p_h(k)$ du mot présenté par le dictionnaire $h(k)$ soit égale à la lettre $p_v(k)$ du mot présenté par le dictionnaire $v(k)$. Cette solution est cependant beaucoup trop coûteuse, car elle ne prend pas en compte les contraintes suffisamment tôt et explore inutilement tout l'espace de recherche.

Trouver une approche raisonnable revient donc à trouver une stratégie pour entrelacer énumération et vérification. L'approche décrite ci-dessous fixe un ordre statique entre les cases qui permet de rechercher les solutions dans un ordre lexicographique en s'arrêtant le plus tôt possible dès qu'une contrainte de la grille n'est pas satisfaite.

Une approche possible

Un dictionnaire d détermine un sous-ensemble de ℓ cases (où ℓ est la longueur des mots du dictionnaire d) et impose un ordre strict sur ces cases $k_1 < \ldots < k_\ell$ en se déplaçant de la gauche vers la droite si le dictionnaire est horizontal ou du haut vers le bas si le dictionnaire est vertical.

L'ensemble des dictionnaires réunis définit donc un ordre partiel sur les cases (on vérifiera facilement qu'il n'y a pas de contradiction possible) qui peut toujours être complété en un ordre total. Nous supposons dorénavant que la numérotation $(i, j) \mapsto k(i, j)$ est un tel ordre.

Un préfixe $a_1 \ldots a_k$ est une solution potentielle de la grille si c'est une solution pour le problème obtenu en noircissant les cases $(k+1) \ldots n$ de la grille et en complétant le dictionnaire par l'ensemble de ses préfixes.

Nous pouvons énumérer les solutions dans un ordre préfixe par l'algorithme suivant : Soit $w = a_1 \ldots a_k$ une solution potentielle de rang (longueur) k. Cherchons une solution potentielle de rang $k+1$:

1. Considérons la case $k+1$ et ses paramètres statiques h, p_h, v, p_v définis ci-dessus. À w correspond un sous-mot w_h obtenu en ne retenant que les cases de w liées au dictionnaire h et qui est un préfixe dans le dictionnaire h, et un sous-mot w_v défini de façon analogue.
2. Soit A^h l'ensemble des lettres qui étendent w_h en un préfixe de h et A^v défini de façon analogue. Soit $A_{k+1} = A^h \cap A^v$.
3. Si A_{k+1} est non vide, on choisit son plus petit élément que l'on prend pour valeur de a_{k+1}. Le mot wa_{k+1} est alors, par construction, une solution potentielle de rang $k+1$.
4. Si A_{k+1} est vide, alors w ne peut pas être étendu en une solution de la grille. Dans ce cas, on invalide le choix de a_k, c'est à dire on retire a_k de A_k et on cherche une autre valeur possible pour a_k dans A_k en retournant à l'étape 3 au rang k.
5. Lorsqu'on a trouvé une solution pour la case n, on a une solution pour la grille (que l'on affiche), puis, pour trouver les solutions suivantes on retire la lettre a_n de A_n et on reprend à l'étape 4 au rang n.

L'efficacité de l'algorithme repose sur l'hypothèse qu'on puisse facilement calculer A_{k+1} et revenir en arrière.

Les calculs de A^h et A^v sont aisés si les dictionnaires sont représentés par une structure d'arbre avec partage des préfixes (voir ci-dessous). Si w est un préfixe du dictionnaire d, alors on peut retrouver le sous-dictionnaire des suffixes qui complètent w en un mot de d par un parcours de d le long de w (coût proportionnel à la longueur de $|w|$). Mieux, on obtient immédiatement le sous-arbre des suffixes $d_{/wa}$ de wa à partir du sous-arbre des suffixes $d_{/w}$ (précédemment calculé et qu'il suffit de mémoriser) en projetant $d_{/w}$ sur la lettre a.

L'algorithme fonctionne par retour en arrière (*backtracking*). L'état de la recherche au rang k peut être mémorisé dans une pile de longueur k donnant, pour chaque case $m \in 1..k$, les deux sous-dictionnaires $h(m)_{/w_h(m)}$ et $v(m)_{/w_v(m)}$. L'élément minimal de A_m est la plus petite première lettre qui commence un mot (en général différent) dans chacun de ces deux sous-dictionnaires.

2.1 Représentation du dictionnaire

La représentation arborescente du dictionnaire est fréquemment utilisée pour la recherche dans de gros dictionnaires. Elle consiste à partager les préfixes des mots du dictionnaire. Par exemple, la liste des mots ci-dessous (figure 1 à gauche) peut être écrite de façon plus concise (figure du milieu) en faisant apparaître une structure d'arbre binaire (figure de droite). Le nœud au sommet

aab	aab
abc	bc
abd	d
bc	bc

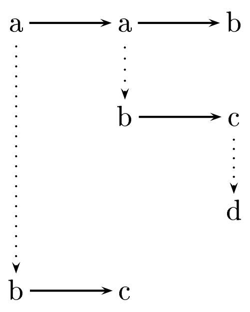

FIG. 1 – Représentation du dictionnaire

est la plus petite lettre ℓ qui commence un mot du dictionnaire. Le fils droit (en trait continu) est le dictionnaire obtenu à partir des mots qui commencent par ℓ par retrait du préfixe ℓ. Le fils gauche (en pointillés) est le dictionnaire des mots qui ne commencent pas par ℓ. Les nœuds du dictionnaire sont ainsi mis en bijection avec l'ensemble des préfixes des mots du dictionnaire. Dans le cas général, il faut aussi coder le fait qu'un nœud puisse terminer un mot tout en étant préfixe d'autres mots. Cependant ceci ne pourra pas se produire si tous les mots du dictionnaire ont la même longueur.

On pourra, si besoin, améliorer cette structure de base de différentes façons :

- On peut partager les arbres suffixes par *hash-consing*. Cela peut réduire significativement la place nécessaire pour stocker le dictionnaire, mais n'aura guère d'influence sur la vitesse de recherche.
- On peut représenter l'arbre par des nœuds n-aires, en codant par un tableau les chaînes de fils droits ou bien les chaînes de fils gauches. Cela peut permettre d'économiser de la place (typiquement un facteur deux, donc pas très significatif) ou bien d'accéder directement au sous-dictionnaire

des fils commençant par une lettre arbitraire sans suivre l'ordre lexicographique (chaînes de fils gauches).
– On peut aussi facilement représenter tout l'arbre dans un tableau, par exemple pour permettre une représentation concise sur disque.

2.2 Prise en compte de la géométrie

L'introduction d'une case noire remplace le dictionnaire horizontal de taille ℓ par deux dictionnaires de tailles p et q avec $p+q+1=\ell$ utilisés à sa gauche et à sa droite et de même verticalement. Il y a bien sûr les cas pathologiques où soit p soit q est vide.

Le remplacement d'une cache k blanche par une lettre ℓ impose la contrainte supplémentaire que A_k doit être réduit à $\{\ell\}$. Plutôt que d'attendre d'arriver à la case k pour vérifier cette contrainte, on préférera bien sûr projeter les dictionnaires $w(k)$ et $h(k)$ en ne retenant que les mots vérifiant déjà la contrainte.

En fait, les cases blanches ou remplies ne sont que des cas particuliers du cas général où chaque case est annotée par un sous-ensemble non vide de l'alphabet définissant les lettres autorisées sur cette case.

2.3 Autres approches

L'approche proposée donne priorité à un parcours de gauche à droite et de haut en bas, alors que le problème est bien entendu symétrique en lisant les mots à l'envers. Selon la géométrie des contraintes, il pourrait s'avérer plus intéressant de le résoudre en lisant les mots à l'envers. Ce qui montre que la solution proposée n'est pas optimale.

Les autres approches s'appuient également sur un mécanisme de *backtracking*, mais certaines tentent de mieux s'adapter à la géométrie en choisissant dynamiquement la définition qui va probablement réduire le plus l'espace de recherche. En contrepartie, elles perdent la structure lexicographique et devront, en général, revenir en arrière «plus profondément». Pour en savoir plus, vous pouvez lire les références ci-dessous, en trouver d'autres sur le réseau, ou en essayer de nouvelles (le problème ne semble pas encore avoir été totalement exploré).

Références

[1] Andrew W. Appel and Guy J. Jacobson. The world's fastest Scrabble program. *Communication of the ACM*, 31 (1988), 572–585.

[2] Steven A. Gordon. A faster Scrabble move generation algorithm. *Sofware-Practice and experience*, Vol 24(2) (February 1994), 219–232.

[3] Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search lessons learned from Crossword puzzles. *AAAI* (1990), pages 210–215.