



HAL
open science

Programmation par ensembles réponses pour simuler l'assolement d'un paysage

Thomas Guyet, Yves Moinard

► **To cite this version:**

Thomas Guyet, Yves Moinard. Programmation par ensembles réponses pour simuler l'assolement d'un paysage. Reconnaissance de Formes et Intelligence Artificielle (RFIA) 2014, Jun 2014, France. hal-00989199

HAL Id: hal-00989199

<https://hal.science/hal-00989199v1>

Submitted on 9 May 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Programmation par ensembles réponses pour simuler l'assolement d'un paysage

T. Guyet^{1,3}

Y. Moinard^{2,3}

¹ AGROCAMPUS-OUEST/IRISA UMR 6079, Rennes

² Inria

³ IRISA, Campus de Beaulieu, 35042 Rennes-Cedex

thomas.guyet@agrocampus-ouest.fr, yves.moinard@inria.fr

Résumé

Le développement d'outils pour la simulation de paysages virtuels est un enjeu important pour l'étude des relations croisées entre les paysages, les processus biophysiques et leurs acteurs. Pour simuler de manière réaliste les colocalisations de cultures, nous formalisons le processus de simulation comme un problème générique de décomposition d'un graphe par des sous-graphes ("structures"). Ce problème est hautement combinatoire et nous proposons d'utiliser la programmation par ensembles réponses (ASP) pour le résoudre. L'article compare plusieurs variantes du programme ASP.

Mots Clef

Graphe, ASP, programmation logique, simulation, paysage

Abstract

The development of tools for the simulation of virtual landscapes is an important issue for the study of the relationships between landscape, biophysical processes and their actors. We are interested in simulating realistic colocalizations of cultures. We formalize the simulation process as a generic problem of decomposition of a graph by subgraphs ("structures"). This problem is highly combinatorial and we propose to solve it using answer set programming (ASP). The article compares several versions of the ASP program.

Keywords

Graph, ASP, logic programming, simulation, landscape

1 Introduction

De plus en plus de recherches en agronomie s'intéressent aux processus agro-écologiques en interaction avec un paysage. Mais les agronomes ne disposent généralement pas d'une variété importante de paysages réels pour identifier les interrelations entre les caractéristiques de paysage et les processus agro-écologique. La simulation de paysages virtuels, réalistes et paramétrables doit pallier ce manque de données réelles. Ces paysages virtuels serviront de support

à la simulation du processus étudié pour en comprendre les interactions.

Nos travaux s'intéressent à la génération d'un assolement pour un paysage agricole, c'est-à-dire à l'attribution des types de culture (blé, prairie, maïs, etc.) à chaque parcelle agricole d'un territoire. Le simulateur de paysage doit servir à l'étude des populations de carabes, insectes vivants dans les espaces interstitiels des parcelles agricoles (haies et talus). Pour les besoins de cette application, nous cherchons à reproduire les caractéristiques organisationnelles des paysages agricoles, *i.e.* d'avoir des colocalisations de parcelles réalistes.

Dans nos travaux précédents, nous avons mis en place une procédure de caractérisation des colocalisations de parcelles à partir d'un paysage numérique représenté sous la forme d'un graphe d'adjacence des parcelles [5]. Notre méthode utilisait un algorithme de fouille de données pour extraire toutes les colocalisations de parcelles sous la forme de graphes, appelés par la suite *structures*.

Dans cet article, nous formalisons le problème de la simulation d'un assolement comme un problème combinatoire de reconfiguration des structures pour reconstituer un graphe de parcelles avec les assolements. Les points abordés dans cet article sont :

1. l'introduction d'un problème de décomposition d'un graphe par un ensemble de structures,
2. la proposition d'une première approche de sa résolution à l'aide d'*Answer Set Programming* (ASP) [7].

2 Simulation de paysages réalistes

Nous définissons un paysage agricole comme l'organisation spatiale d'éléments paysagers naturels (*e.g.* parcelles agricoles, haies, plans d'eau) ou artificiels (*e.g.* bâtiments, routes) avec lequel interagissent des acteurs aussi divers que des agriculteurs, des insectes, la faune en général, la végétation naturelle et cultivée, etc. Un paysage virtuel est une représentation numérique d'un paysage. L'approche de simulation de paysage réaliste que nous mettons en œuvre est inspirée de Le Ber et *al.* [6]. Cette méthode de simulation neutre de paysage consiste à extraire des caractéris-

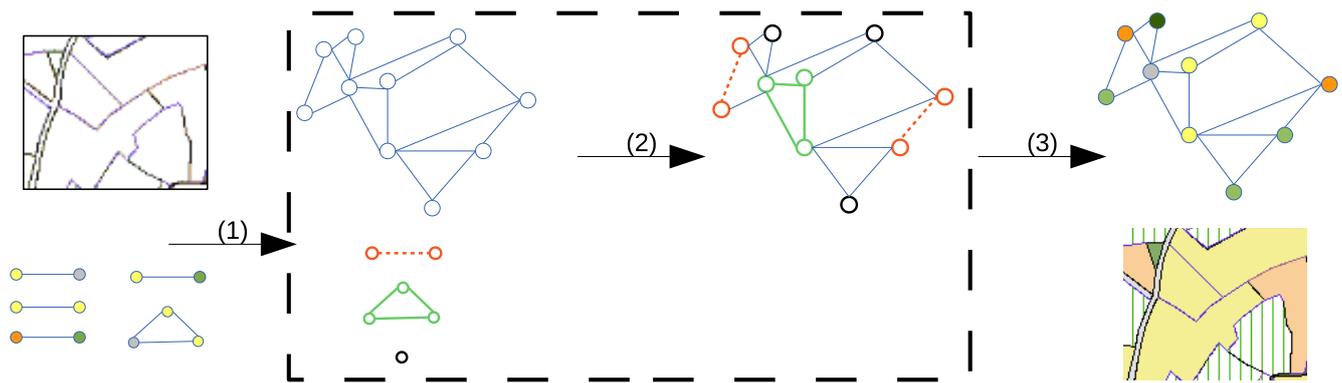


FIGURE 1 – Processus de simulation d’une occupation du sol en trois étapes. L’article se focalise sur la partie encadrée.

tiques des paysages et de les reproduire dans les paysages simulés. La méthode est dite “neutre” par opposition aux modèles décisionnels qui s’appuient sur la modélisation des acteurs et de leurs pratiques pour générer les paysages (par ex. Akplogan *et al.* [1] ou Bergez *et al.* [3]).

Dans notre approche, un paysage est représenté par un graphe dont les nœuds sont des parcelles agricoles et dont les arcs sont des relations spatiales entre les parcelles agricoles [5]. Cette représentation numérique met en exergue des structures dans le paysage (*i.e.* des colocalisations de parcelles) et leurs interfaces de cultures. L’utilisation de méthodes de fouille de sous-graphes fréquents permet d’extraire des sous-graphes récurrents dans un paysage. Ces sous-graphes caractérisent le paysage.

Le processus de simulation prend en entrée un parcellaire, *i.e.* un ensemble de parcelles, et un ensemble de structures qu’il faut reproduire. La simulation se déroule en trois étapes (*cf.* figure1) :

1. générer un graphe de parcelles “nues” (sans attribution d’occupation du sol) à partir du parcellaire. En parallèle, on génère également l’ensemble des structures “nues” (graphes sans attributs sur les nœuds). Dans l’exemple de la figure, on a deux types de structures (paire de nœuds en orange et un triplet de nœuds en vert) auxquels on ajoute le graphe singleton (*i.e.* un graphe ne comportant qu’un seul nœud, sans arc).
2. décomposer le graphe en sous-graphes correspondant aux structures nues existantes (dans la suite, on se focalisera sur cette étape)
3. attribuer les occupations du sol aux nœuds en tirant aléatoirement les structures pour chaque structure nue correspondante. Dans l’exemple de la figure, il y a 4 possibilités pour colorer une structure à 2 nœuds. Le choix se fait aléatoirement, ce qui permet de générer un grand nombre d’assolements avec le même découpage du graphe.

On introduit maintenant le problème de décomposition optimale qui formalise l’étape 2 de ce processus de simulation.

3 Décomposition optimale d’un graphe par des structures

3.1 Présentation du problème

Soit $G = \langle V, E \rangle$ un graphe où V est l’ensemble des nœuds du graphe et $E \subset V \times V$ l’ensemble des arcs du graphe. Les graphes sont non-attribués (*cf.* point 3 dans la section ??) et non orientés. Soit \mathcal{S} un ensemble de graphes connexes¹ appelés ici “structures”. On suppose que les graphes sont tous distincts (*i.e.* il n’y a aucune paire de structures qui soient isomorphes).

Une **instance** IS d’une structure $S = \langle V_S, E_S \rangle \in \mathcal{S}$ dans un graphe G est un sous-graphe de G isomorphe à la structure S : tout sommet de IS est un sommet de G et tout arc de IS est un arc E de G .

Une **décomposition du graphe G par \mathcal{S}** est un ensemble d’instances de S tel que chaque nœud de G est associé à une seule et unique instance.

La décomposition est dite **optimale** lorsque c’est une décomposition dont le nombre d’instances est le plus petit de toutes les décompositions possibles.

La décomposition n’est pas nécessairement possible dans le cas général. En revanche, quelque soit le graphe G , si \mathcal{S} comporte comme structure un graphe-singleton (constitué d’un seul nœud), alors la décomposition triviale en autant d’instance-singleton que de nœuds est possible, et donc une décomposition optimale existe. La décomposition optimale n’est bien sûr pas nécessairement unique.

La figure 2 illustre ces définitions.

3.2 Représentation du problème en ASP

La programmation par ensembles réponses [2] (abrégié ici par l’acronyme anglais ASP, “*answer set programming*”) est proche de l’idéal de la programmation déclarative : *le problème est le programme*. La solution est un ensemble de modèles particuliers appelés ensembles réponses. Cette méthode de programmation est bien adaptée à des problèmes de parcours dans des graphes et permet d’écrire des

1. Graphe connexe : il existe au moins un chemin entre chaque paire de nœuds.

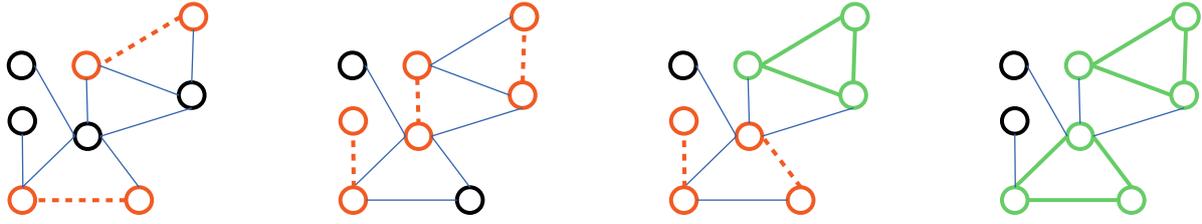


FIGURE 2 – Quatre exemples de décomposition de problème de la figure 3 : à gauche, une décomposition avec 6 instances (2 paires et 4 singletons), en deuxième, une décomposition avec 5 instances (3 paires et 2 singletons), à droite, deux décompositions optimales avec 4 instances (2 paires, 1 singleton et 1 triangle, ou 2 triangles et 2 singletons).

programmes élégants, y compris pour les règles avec exception. Les moteurs de résolution des programmes ASP s'appuient sur des solveurs de contraintes efficaces. En pratique, nous utilisons l'outil `clingo4`² [4] dont les performances sont intéressantes et qui respecte la syntaxe standard d'ASP³.

Le graphe et les structures sont représentées à l'aide de cinq prédicats `node`, `edge`, `structure`, `snode`, `sedge` de la façon suivante :

- `node(X)` : `X` est un nœud du graphe
- `edge(X,Y)` : le graphe comporte un arc entre les nœuds `X` et `Y`
- `structure(S)` : `S` est une structure
- `snode(S,P)` : `P` est un nœud de la structure `S`
- `sedge(S,P,Q)` : la structure `S` a un arc entre les nœuds `P` et `Q`

Notons que pour faciliter l'écriture des programmes, on choisit ici d'utiliser des ensembles d'entiers consécutifs partant de 0 comme ensembles des valeurs pour ces paramètres `X`, `Y`, `S` et `P`. La représentation d'une solution au problème de décomposition d'un graphe par un ensemble de structures utilise les prédicats `instance` et `sgraph` :

- `instance(I,S)` : `I` désigne une instance de la structure `S` dans le graphe (on choisira ici des noms de nœuds de `G` pour désigner ces instances).
- `sgraph(I, X, P)` : le nœud `X` du graphe est un élément du sous-graphe (instance) désigné par `I` correspondant au nœud `P` de la structure qui a fourni l'instance `I`. L'ensemble de ces atomes en `sgraph` décrit ainsi la décomposition du graphe en sous-graphes.

La figure 3 illustre un AS, *i.e.* un modèle qui représente une décomposition optimale.

4 Résolution avec ASP

En complément des données qui sont représentées par les prédicats ci-dessus, le programme ASP est décomposé en trois parties : (1) La partie "génération" consiste à générer des modèles solution (AS) sans que ceux-ci répondent nécessairement à toutes les contraintes du problème. (2) La partie "contraintes" ajoute les contraintes qui sont vérifiées

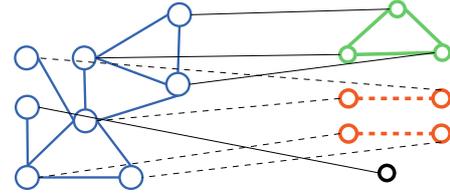


FIGURE 3 – Illustration d'un AS. À gauche : le graphe, à droite : les instances. Les traits fins entre les nœuds des instances et les nœuds du graphes illustrent les atomes `sgraph`.

sur les solutions générées, ce qui élimine les solutions générées qui ne satisfont pas toutes les conditions voulues. (3) La partie "optimisation" définit les contraintes d'optimisation, ce qui va identifier les solutions "les meilleures" par rapport aux critères choisis.

4.1 Première approche

Le programme ASP résout le problème de décomposition en se basant sur la génération des instances. L'idée principale de cette méthode est de générer des modèles qui mettent en correspondance des structures avec des nœuds du graphe et de ne conserver comme AS (ensembles-réponse du programme complet) que les modèles dont les instances font aussi correspondre les arcs de la manière voulue.

- ```

1 0 { instance(I,S): structure(S) } 1 :- node(I).
2 snumber(S, L) :- L = #count{ N : snode(S,N) }, structure(S).
3 L { sgraph(I, X, P): node(X), snode(S,P) } L :- snumber(S, L),
 instance(I,S).

```

Listing 1 – Partie "génération"

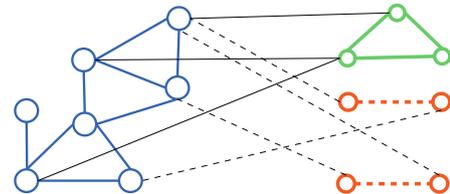


FIGURE 4 – Exemple de modèle généré par le listing 1. Ce modèle n'est pas un AS de notre problème.

La première ligne de la génération (listing 1) indique que chaque nœud du graphe engendre aléatoirement au plus un

2. Potassco : <http://potassco.sourceforge.net/>  
3. Standard ASP : <https://www.mat.unical.it/aspcomp2013/ASPStandardization>

atome **instance(I,S)** pour chaque nœud **I**. La figure 4 illustre la génération de trois instances (1 triangle et 2 paires). Le numéro de nœud **I** sert d'identifiant à l'instance, ce qui garantit qu'il n'y aura pas plus d'instances que de nœuds. Notons que l'on génère ainsi, entre autres, la solution qui fait correspondre l'unique nœud de l'instance singleton à chaque nœud du graphe  $G$ . Cela assure l'existence d'au moins un AS. Bien sûr cette solution est (sauf exception) très éloignée de la solution optimale, puisqu'il s'agit de la solution qui utilise le plus grand nombre possible d'instances.

Pour chaque instance **I**, on engendre autant d'atomes **sgraph** qu'il y a de nœuds dans la structure de l'instance. Ces atomes font correspondre les nœuds d'une instance à des nœuds du graphe. Ces correspondances sont illustrées par les traits fins dans les illustrations des figures 3, 4 et 5. À ce stade, on assure simplement qu'il y aura autant d'atomes **sgraph(I,\_,\_)** qu'il y a de nœuds dans l'instance **I**. En particulier, il peut y avoir plusieurs correspondances pour un nœud d'une structure et plusieurs nœuds de structure peuvent correspondre à un même nœud du graphe. De telles solutions ne sont pas des solutions à notre problème et doivent être éliminées à l'aide de contraintes complémentaires.

Les contraintes *d'unicité* (lignes 2 à 4 du listing 2) assurent l'existence d'une bijection entre l'ensemble des nœuds de structures générées, et l'ensemble des nœuds du graphe  $G$  (cf. figure 5) : À la ligne 2, on indique qu'un même nœud **P** d'une instance **I** ne peut pas être attribué à deux nœuds différents du graphe. Les lignes 3-4 indiquent la réciproque : un nœud du graphe est couvert par un unique nœud d'une unique instance. La ligne 6 du listing 2 indique que chacun des nœuds du graphe  $G$  doit correspondre à un nœud d'une structure générée. En effet, cette contrainte (règle sans tête, donc "avec tête fausse", élimine les solutions partielles où un nœud du graphe  $G$  ne figure dans aucun des atomes en **sgraph** générés :  $0\{\dots\}0$  signifie que le cardinal de l'ensemble  $\{\dots\}$  est nul, et donc qu'il n'existe aucun **sgraph(I,N,P)** généré. La figure 5 montre le cas d'un AS qui ne respecte pas cette contrainte (un nœud du graphe ne correspond à aucun nœud des structures concernées).

```

1 % unicités
2 :- sgraph(I,X,P), sgraph(I,Y,P), X<Y.
3 :- sgraph(I,X,P), sgraph(J,X,Q), Q<P.
4 :- sgraph(I,X,P), sgraph(J,X,Q), I<J.
5
6 :- 0{ sgraph(I,N,P) }0, node(N).

```

Listing 2 – Partie "contraintes" (1/2)

Ces 4 contraintes vont imposer un nombre total de nœuds des structures égal au nombre de nœuds du graphe.

```

1 % contraintes de correspondance des arcs
2 :- instance(I, S), sgraph(I, Y, Q), sgraph(I, X, P), sedge(S, P, Q), not edge(X,Y).

```

Listing 3 – Partie "contraintes" (2/2)

La seconde partie des contraintes, listing 3, vérifie l'existence d'arcs entre les nœuds du graphe  $G$  (resp. nommés **X**

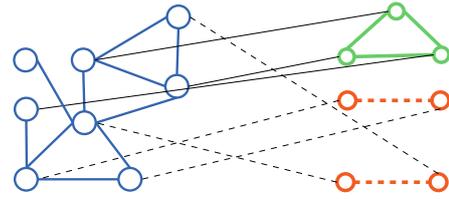


FIGURE 5 – Exemple de modèle généré avec les contraintes d'unicité, mais qui n'est pas un AS.

et **Y**) qui sont en correspondance avec des nœuds d'une instance **I** (resp. nommés **P** et **Q**). Les AS qui sont conservés (tel que l'AS illustré par la figure 3) constituent les solutions du problème de la décomposition du graphe.

```

1 nbinstances(L) :- L={instance(I,S)}.
2 #minimize { L@1 : nbinstances(L) }.

```

Listing 4 – Partie "optimisation"

Finalement, la partie optimisation précise qu'on cherche un modèle dont le nombre d'instances est minimal. La recherche de solutions avec un nombre d'instances optimal favorise l'utilisation des grosses structures.

Le programme proposé est complet et correct. Avant l'introduction de la règle avec **minimize** l'ensemble des AS est l'ensemble des solutions du problème de décomposition.

Le programme décrit est assez facile à écrire, puisqu'on a traduit en un langage logique naturel l'ensemble de données du problème. Un inconvénient de ce programme est l'inefficacité de sa phase de génération. Un même ensemble d'instances peut être généré en utilisant des numéros de nœuds du graphe différents. On démultiplie ainsi inutilement le nombre d'AS à traiter. Voici une première amélioration dans la génération des solutions possibles.

## 4.2 Amélioration de la génération

La première amélioration réduit la combinatoire dans la génération des instances en remplaçant la génération précédente par le listing ci-dessous. Dans la solution précédente (voir listing 1), un nœud donne un identifiant pour une instance. Lorsque la structure a plusieurs nœuds, chaque nœud engendre une solution différente, mais équivalente pour le problème posé (par exemple, il y aura  $3 \times 2 \times 2 \times 1$  pour 1 instance de la structure verte, 2 instances de la structure orange et 1 instance du singleton). Dans la solution ci-dessous, une combinaison donnée de structure générée ne sera évaluée qu'une seule fois.

Le principe de cette approche est de fixer un nombre d'instances **K**, et d'engendrer les atomes **instance** pour associer chacune de ces instances à une structure. Toute la combinatoire de structures est ainsi explorée (e.g. pour 2 structures :  $K$  instances de  $S_1$  et 0 de  $S_2$ ,  $K - 1$  instances de  $S_1$  et 1 de  $S_2$ , etc.). Ceci est effectué pour toutes les valeurs de **K** possibles, i.e. de 1 au nombre de nœuds du graphe.

```

1 { nbinstances(K): node(K) } 1.

```

```

2 instid(1..K) :- ninstances(K).
3 1{ instance(I,S) : structure(S) } 1 :- instid(I).

```

### 4.3 Ajout de la contrainte de nombre de nœuds

La seconde amélioration rend explicite la contrainte sur la somme des nombres de nœuds des instances. Celle-ci doit correspondre au nombre de nœuds du graphe. Cette contrainte est redondante avec les contraintes d'unicité, mais elle peut être utilisée en amont de la génération des atomes en **sgraph** et ainsi élimine-t-elle très tôt des AS potentiels non valides.

La contrainte est exprimée à l'aide de l'agrégat **#sum**. **#sum** { **L** : **suminstance**(**I**, **L**) } somme les valeurs de **L**, le nombre de nœuds dans une instance.

```

1 suminstance(I, L) :- snumber(S,L), instance(I,S).
2 :- M { node(N) }, M=Tot+1, Tot=#sum{ L : suminstance(I, L) }.
3 :- { node(N) } M, M=Tot-1, Tot=#sum{ L : suminstance(I, L) }.

```

### 4.4 Modification de la contrainte sur les arcs

On propose finalement la modification ci-dessous de la contrainte sur les arcs (listing 3).

```

1 % L est le degré du nœud P dans la structure S
2 sdegree(S, P, L) :- L= #count{ N : sedge(S, P, N) }, structure(
 S), snode(S, P).
3
4 % contraintes de correspondance des arcs
5 :- { sgraph(I, Y, Q): edge(X,Y), sedge(S, P, Q) } M, instance(
 I, S), sgraph(I, X, P), sdegree(S, P, L), M=L-1.
6 :- M { sgraph(I, Y, Q): edge(X,Y), sedge(S, P, Q) }, instance(
 I, S), sgraph(I, X, P), sdegree(S, P, L), M=L+1.

```

La contrainte est exprimée à l'aide du nombre d'arcs d'un nœud **X** : { **sgraph**(**I**, **Y**, **Q**) : **edge**(**X**,**Y**), **sedge**(**S**, **P**, **Q**) } désigne l'ensemble des nœuds du graphe **Y** pour lequel 1) il existe un arc entre **X** et **Y**, et 2) il existe un arc entre **P** et **Q**. On vérifie alors que la taille de cet ensemble est exactement le degré du nœud correspondant dans la structure. Comme en fin de §4.3, l'égalité s'exprime ici comme l'impossibilité des deux inégalités strictes (ligne 5 – élimine les cas avec un cardinal supérieur ou égal à **M** – et ligne 6 – élimine les cas avec un cardinal inférieur ou égal à **M**). Notons que par construction on a la correspondance dans un sens, et donc vérifier cette égalité assure la correspondance bijective.

## 5 Résultats et discussions

Nous disposons donc de quatre programmes : la version de base P0 (§4.1), et les versions P1 à P3 (§4.2 à 4.4), qui incluent incrémentalement les trois améliorations proposées. Ces programmes résolvent le problème de minimisation en trouvant les décompositions optimales des graphes.

### 5.1 Évaluation des temps de résolution

Nous comparons les programmes en nous intéressant au temps de résolution. Des graphes aléatoires sont générés. On note  $s > 0$  le nombre de nœuds d'un graphe et  $d \in$

$[0, 1]$  la densité des arcs. Les tests sont effectués en utilisant 10 structures différentes comportant au plus 4 arcs.

Le tableau 1 donne les résultats obtenus pour les trois versions avec plusieurs valeurs de  $d$  et  $s$ . Les résultats sont obtenus pour 5 réplifications de la résolution sur des graphes aléatoires (les mêmes graphes servent aux trois méthodes). Pour des raisons de temps d'expérimentation, on borne les temps d'exécution à 120s. Les temps moyens sont calculés en utilisant uniquement les résolutions pour lesquelles un optimum a été atteint dans ce laps de temps. La proportion des succès est indiquée dans le tableau 1.

On constate tout d'abord, en particulier pour  $s = 8$ , que chacune des améliorations proposées diminue significativement les temps de calcul. Les mêmes résultats se retrouvent indirectement en regardant les nombres de résolution qui n'ont pas abouti aux optimums dans le temps imparti. Sans les améliorations, ces situations sont plus fréquentes. Les deux améliorations majeures sont, d'une part, l'amélioration de la génération et, d'autre part, la modification sur les contraintes d'arcs.

L'amélioration de la génération était un gain en temps espéré, dans la mesure où cela réduit drastiquement le nombre d'AS à explorer. La modification de la contrainte d'arcs offre un gain de performance assez inattendu. Bien qu'elle soit définie comme une contrainte, il semble qu'elle agisse sur la génération des atomes **sgraph**, mais ceci reste à confirmer par une analyse plus fine de la traduction en contraintes par **clingo**.

### 5.2 Limite liée à la combinatoire des solutions

L'approche de résolution proposée contient néanmoins toujours des redondances liées à la combinatoire du problème. En premier lieu, la symétrie des structures engendre beaucoup d'AS redondants, *i.e.* correspondant à la même solution. Dans l'exemple du triangle de la figure 3, chaque permutation circulaire des **sgraph** pour le triangle est un nouvel AS (combinaison différente de **X** avec des **P** dans nos notations). En pratique, cette seule symétrie triple le nombre d'AS par rapport au nombre de "solutions réelles".

Une solution partielle à ce problème, qui peut diminuer sérieusement le nombre d'AS affichés, utilise l'option `-project` de **clingo**. Il conviendrait d'ajouter une règle de "projection" comme la règle suivante :

```

1 % nœud couvert
2 ncouvert(I,X) :- sgraph(I, X, P).
3 #show ncouvert/2.
4 #show instance/2.

```

Il faut aussi ajouter au programme des "instructions" **#show** demandant de n'afficher que certains prédicats. Ici, on a besoin au moins des prédicats **ncouvert** et **instance**, et surtout pas **sgraph**.

Les solutions affichées sont alors considérablement moins nombreuses, puisque des variantes qui ne diffèrent que par les attributions arbitraires de numéros sont confondues. Par

| <b>d=0.2</b> | <i>s</i> =8       | <i>s</i> =12   | <i>s</i> =16     | <i>s</i> =20    |
|--------------|-------------------|----------------|------------------|-----------------|
| <i>P0</i>    | 58 ± 15 (1)       | – (0)          | – (0)            | – (0.6)         |
| <i>P1</i>    | 29 ± 24 (1)       | – (0)          | – (0)            | – (0.2)         |
| <i>P2</i>    | 6.9 ± 2.6 (1)     | – (0)          | – (0.6)          | – (0.6)         |
| <i>P3</i>    | 0.55 ± 0.15 (1)   | 11 ± 17 (1)    | 5.9 ± 0.98 (0.4) | 74 ± 50 (0.6)   |
| <b>d=0.3</b> | <i>s</i> =8       | <i>s</i> =12   | <i>s</i> =16     | <i>s</i> =20    |
| <i>P0</i>    | 61 ± 36 (1)       | – (0)          | – (0)            | – (0)           |
| <i>P1</i>    | 8 ± 10 (1)        | 45 ± 17 (0.4)  | – (0)            | – (0)           |
| <i>P2</i>    | 5.1 ± 4.1 (1)     | 46 ± 27 (0.4)  | 38 ± 32 (0.4)    | 22 ± 22 (0.4)   |
| <i>P3</i>    | 0.63 ± 0.064 (1)  | 3 ± 2.8 (1)    | 16 ± 21 (1)      | 8.1 ± 1.7 (1)   |
| <b>d=0.4</b> | <i>s</i> =8       | <i>s</i> =12   | <i>s</i> =16     | <i>s</i> =20    |
| <i>P0</i>    | 37 ± 50 (0.8)     | – (0)          | – (0)            | – (0)           |
| <i>P1</i>    | 1.7 ± 1.9 (1)     | 20 ± 18 (1)    | – (0)            | – (0)           |
| <i>P2</i>    | 0.97 ± 0.56 (1)   | 7.6 ± 13 (1)   | 4 ± 1 (1)        | 7.6 ± 1.1 (0.8) |
| <i>P3</i>    | 0.7 ± 0.1 (1)     | 2.1 ± 0.18 (1) | 5.3 ± 0.43 (1)   | 8.2 ± 0.52 (1)  |
| <b>d=0.5</b> | <i>s</i> =8       | <i>s</i> =12   | <i>s</i> =16     | <i>s</i> =20    |
| <i>P0</i>    | 1.3 ± 0.085 (0.8) | – (0)          | – (0)            | – (0)           |
| <i>P1</i>    | 0.6 ± 0.41 (1)    | 36 ± 33 (1)    | – (0)            | – (0)           |
| <i>P2</i>    | 0.67 ± 0.45 (1)   | 1.4 ± 0.29 (1) | 2.9 ± 0.29 (1)   | 5.7 ± 0.34 (1)  |
| <i>P3</i>    | 0.77 ± 0.079 (1)  | 2.4 ± 0.27 (1) | 5.8 ± 0.45 (1)   | 9.5 ± 0.7 (1)   |

TABLE 1 – Temps de résolution (en secondes) en fonction de *s* et *d*. Les durées sont exprimées en secondes. Dans chaque cellule, on donne la valeur moyenne, l'écart type et, entre parenthèses, on indique la proportion (entre 0 et 1) de graphe pour lesquels un optimum a été trouvé (dans la limite de 120s). Le “–” indique qu’aucune résolution n’a trouvé d’optimum, les parenthèses donnent alors la proportion de problèmes pour lesquels aucune solution n’a été trouvée en 120s.

contre, si on veut reconstituer les correspondances entre arcs, qui sont utiles elles aussi, il faut ajouter un petit post-traitement. Ce problème d'élimination de solutions semblables est un problème fréquemment rencontré avec ASP.

## 6 Conclusion et perspectives

Dans ce travail, nous avons présenté le cadre général de la simulation de paysages agricoles qui reproduisent des organisations structurelles souhaitées. Nous avons proposé d'utiliser un langage de programmation logique, la programmation par ensemble réponse (ASP), pour résoudre le problème combinatoire de détermination d'un découpage optimal du parcellaire agricole selon des structures, et nous avons formalisé ce problème. Plusieurs programmes ont été proposés et comparés. Le premier programme simple montre que face à un problème très combinatoire, il est nécessaire d'améliorer les programmes ASP traduisant le plus directement les données du problème. L'amélioration de la résolution passe alors par deux méthodes :

1. Améliorer la phase de génération des ensembles réponses (en limitant a priori le nombre de modèles qui seront évalués, on limite le temps de résolution).
2. Ajouter des contraintes supplémentaires, y compris redondantes, pour aider la résolution par le solveur de contraintes.

Dans ce travail, nous avons simplifié la combinatoire du problème initial, ne nous intéressant qu'à des graphes non-attribués. En ne prenant pas en compte les attributs, on diminue fortement le nombre de structures. Une perspective à

court terme est d'intégrer l'attribut des nœuds directement dans la génération. En utilisant les possibilités de contrôle du processus de résolution de `clingo4`, il serait possible de traiter ce problème dans une seconde phase.

Un des intérêts de l'ASP est de faciliter l'expression de contraintes. La résolution du problème de l'allocation des structures spatiales peut se coupler facilement à des contraintes du domaine expert comme, par exemple, “on ne doit pas positionner des parcelles agricoles à côté d'un cours d'eau”. La perspective de ce travail est donc de proposer un outil dans lequel l'utilisateur pourra définir ses propres contraintes sous forme logique et obtenir des paysages simulés respectant ces contraintes.

## Références

- [1] M. Akplogan, S. de Givry, J-Ph. Métivier, G. Quesnel, A. Joannon, and F. Garcia. Solving the crop allocation problem using hard and soft constraints. *RAIRO - Operations Research*, 47(2) :151–172, 2013.
- [2] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.
- [3] J.E. Bergez, N. Colbach, O. Crespo, F. Garcia, M.H. Jeuffroy, E. Justes, C. Loyce, N. Munier-Jolain, and W. Sadok. Designing crop management systems by simulation. *European Journal of Agronomy*, 32(1) :3–9, 2010.
- [4] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. *Potassco : The Pots-*

dam answer set solving collection. *AI Communications*, 24(2) :107–124, 2011.

- [5] Th. Guyet. Fouille de données spatiales pour la caractérisation spatiale de paysages en lien avec des fonctionnalités agro-écologiques. In *Spatial Analysis and GEOmatics (SAGEO'10)*, page 3, 2010.
- [6] F. Le Ber, C. Lavigne, K. Adamczyk, F. Angevin, N. Colbach, J-F. Mari, and H. Monod. Neutral modelling of agricultural landscapes by tessellation methods - application for gene flow simulation. *Ecological Modelling*, 220 :3536–3545, 2009.
- [7] Y. Moinard. Utiliser la programmation par ensembles réponses pour de petits problèmes. In *Actes de la conférence RFIA 2012*, 2012.