CENTRALE NANTES

OPTION INFORMATIQUE

Projet de Groupe

Guide utilisateur d'Abita4Rhino

Guide d'utilisation du module Abita4Rhino pour Rhino et Grasshopper

Encadrants :
Myriam SERVIERES, Vincent TOURRE

Client :
Laurent LESCOP (laboratoire AAU)

Étudiants :
Alexandre BOULANGER, Alexis DELAGE, Anne-Sophie JOURLIN, Lénaëlle LE ROY et Louis PAUTRAT

29 mars 2022 - Version 1

Table des matières

1 Préambule

1.1 Rappel du problème

On pose le problème comme suit : à partir d'une **surface** découpée en **éléments**, on souhaite regrouper les éléments en **lots** afin de former une **solution**. Chaque lot correspond à un appartement (de type T1, T2, etc...) ou à un espace commun aux différents appartements (cet espace commun permet alors d'accéder aux différents appartements).

Un élément peut être soumis à une des contraintes suivantes :

- Commun Possible : cet élément peut être attribué à un lot correspondant à un espace commun (par défaut il ne l'est pas) ;
- Commun Imposé : cet élément doit être attribué à un lot correspondant à un espace commun ;
- Entrée-Sortie : cet élément correspond à une entrée/sortie du logement.

On souhaite alors calculer une **solution** possible, étant donné les contraintes précédentes, afin que :

- la valeur de la solution (calculée en fonction du nombre de lots habitables et de leur taille) soit maximale ;
- tous les lots soient connectés à un élément Entrée/Sortie, soit directement, soit via un espace commun.

L'algorithme doit alors proposer plusieurs solutions possibles, classées par valeur décroissante.

1.2 Intégration du projet Abita4Rhino au sein du projet Abita

Le diagramme en figure 1 présente les différentes manières de résoudre le problème, et les liens possibles entre les différents projets. On distingue principalement 3 projets :

- Abita+ : ce projet contient un programme `Solveur.exe` permettant de résoudre le problème : celui-ci prend en entrée un fichier au format `.abi`, et donne en sortie les solutions sous un nouveau fichier au même format. Ce projet contient aussi un second programme `Visualiseur.exe` permettant de visualiser les fichiers `.abi`.
- AbitaPy : ce projet consiste essentiellement en la réécriture du solveur d'Abita+ en langage Python, afin de permettre une intégration de la méthode de calcul à d'autres logiciels.
- Abita4Rhino : ce projet consiste en le développement de blocs fonctionnels pour Grashopper, exécutés sous Rhino. Ces blocs s'appuient sur AbitaPy.

1.3 Environnement logiciel

Le projet Abita4Rhino a été développé à partir de AbitaPy2, qui est une version rétroportée en Python 2.7 du programme AbitaPy (développé pour Python 3.7).

Ce projet propose à l'utilisateur d'ajouter des "blocs" fonctionnels à l'interface de Grasshopper permettant de résoudre le problème précédemment posé. Ces blocs s'appuient sur l'exécution de code Python, à travers le plug-in IronPython de Rhino.

À noter que les tests ont uniquement été réalisés sous Rhino7. Le projet peut être compatible avec d'autres versions de Rhino, mais sans certitude. De même, le projet pourra aussi être utilisé en théorie sous d'autres logiciels que Rhino, à condition que ceux-ci soient compatibles à la fois avec IronPython et Grasshopper.

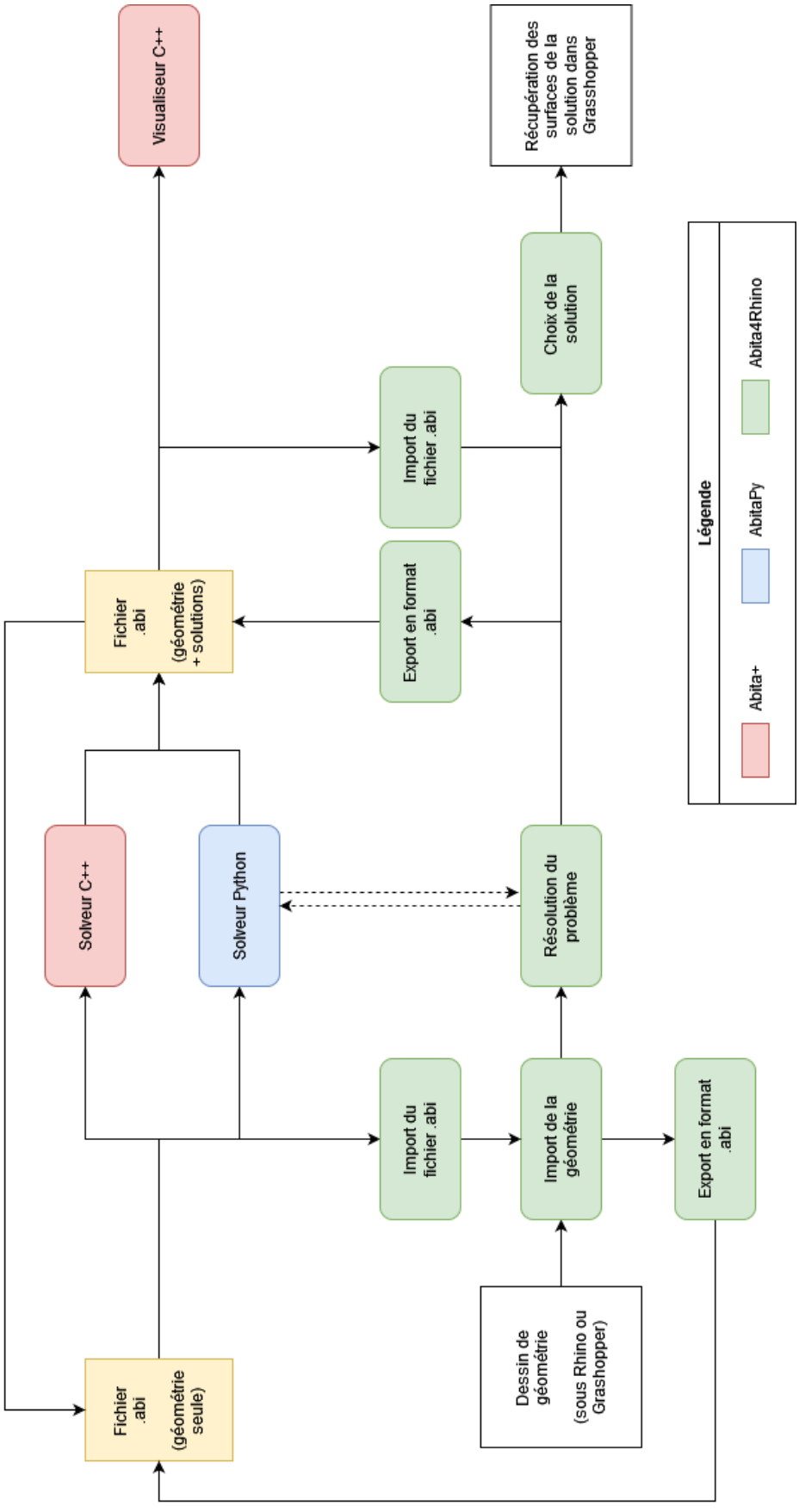

FIGURE 1 – Les différents moyens de résoudre le problème avec les outils Abita développés

2 Installation d'Abita4Rhino

Dans ce guide, nous allons détailler comment procéder à l'installation des composants nécessaires à l'utilisation de Abita4Rhino sous Rhino et Grasshopper.

L'installation du projet néccessite d'avoir Rhino 7 ainsi que Grasshopper installés. Vous aurez aussi besoin de télécharger l'entiereté du dossier `Abita4Rhino` du repository Gtihub situé à cette adresse : https://github.com/hydrielax/Abita (ce repository étant privé, vous pouvez contacter les auteurs du projet pour demander un accès).

2.1 Installation des modules Python

Avant toute chose, il est nécessaire d'ajouter les modules python `ply‘ etabitaPy2` au plug-in IronPython de Rhino (version 7 de préférence).

Pour ce faire, copiez et collez les dossiers `ply` et `abitaPy2` du dossier `Abita4Rhino/Libraries` dans le dossier des libraries Python de Rhino : `<root>/Rhino/Plug-ins/IronPython/Lib` où `<root>` est le dossier dans lequel Rhino est intallé.

Remarque : les codes sources de ces deux modules fournis dans le dossier ne sont pas forcément à jour, leur code source a simplement été recopié dans ce dossier par commodité pour l'utilisateur. Les vraies sources maintenues à jour de ces modules peuvent être retrouvées ici :

- pour `abitaPy2`, dans le dossier `AbitaPy/abitaPy2` du même repository ;
- pour `ply`, en téléchargeant l'archive du code source depuis cette page https://pypi.org/project/ply/#files.

2.2 Installation des modules Grasshopper

Une fois les modules python installés, on peut lancer Rhino puis Grasshoper. Nous allons alors charger les fichiers des modules dans Grasshopper. Pour ce faire :

- Dans une fenêtre Grasshopper, ouvrez l'onglet `File`
- Sélectionnez `Special Folders` puis `User Object Folder`
- Le dossier contenant les modules personnalisés de Grasshopper s'ouvre. Copiez-collez alors tous les fichiers .ghuser du dossier `Abita4Rhino/UserObjects` dans ce dossier.
- C'est terminé ! Si l'installation est réussie, vous devriez voir un nouvel onglet `AbitaPy2` de plus dans la barre d'outils de Grasshopper, qui contient les composants propres à `Abita4Rhino`.

3 Utilisation d'Abita4Rhino

Cette partie présente les différents blocs d'Abita4Rhino utilisables sous Grasshopper et leur méthode d'utilisation. Ces blocs consistent en un ensemble de neufs "User Objects", aussi appelés blocs fonctionnels ou composants. L'onglet obtenu avec ces neufs blocs Grasshopper est présenté en figure 2.

FIGURE 2 – Onglet des composants créés pour Grasshopper

Ces blocs sont réunis par catégories, en fonction de ce qu'ils permettent de faire. Les parties qui suivent explicitent le fonctionnement de ces différents composants.

3.1 Catégorie 1 : lecture et écriture de fichiers .abi

La première catégorie concerne la lecture et l'écriture de fichiers .abi : il y a un composant pour chacune de ces deux actions ici. Leur aspect dans Grasshopper est visible en figure 3.

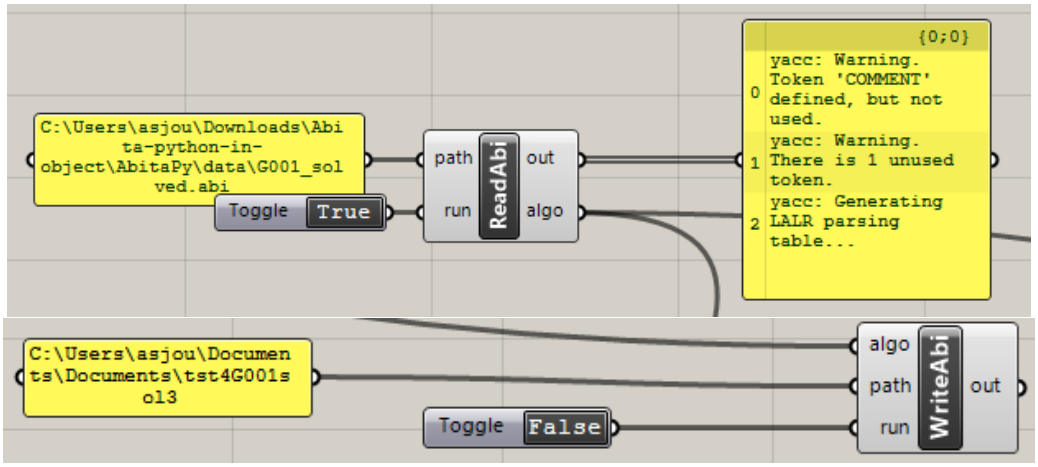

FIGURE 3 – Lecture et écriture de fichiers .abi

Les entrées des deux boîtes sont suffisamment semblables pour être traitées ensemble : on dispose d'une entrée qui correspond au path absolu depuis la racine de l'ordinateur vers un fichier .abi que l'on désire soit lire soit écrire. Dans le cas de l'écriture, il n'est pas nécessaire que le fichier existe pour que l'action ait lieu. Dans chacun des deux cas, si on omet la mention ".abi" en fin de fichier elle est rajoutée automatiquement par le script. Il y a de plus une entrée pour un booléen : il suffit de passer l'entrée à `True` pour lancer soit la phase de lecture soit la phase d'écriture. Nous avons fait ce choix pour permettre à l'utilisateur de rompre une partie de la continuité des actions sur Grasshopper. Nous trouvions ce fonctionnement opportun ici puisque l'utilisateur interagit avec des fichiers extérieurs à Grasshopper.

Dans le cas du composant pour la lecture, il y a une sortie : une entité de la classe `Algo`, qui correspond donc au problème inscrit dans le fichier .abi lu. Notons ici que s'il s'agit d'un fichier contenant une solution, il y a aura la solution dans l'entité générée, mais que sinon nous ne faisons pas tourner l'algorithme à cette étape, qui n'est véritablement qu'une simple lecture des informations contenues dans le fichiers, sans aucun ajout. Il est donc possible d'effectuer différentes actions sur l'objet en fonction de s'il y a une solution contenue ou non.

Les sorties générées dans la sortie "out" sont soit des erreurs car les fichiers cherchés ne sont pas trouvés dans une tentative de lecture, soit des indications générées par le parseur, donc un des scripts construits pour le coeur du solveur. Enfin il y a une indication dans la cas où l'action est effectuée avec succès.

3.2 Catégorie 2 : Construction des éléments du problème

Nous construisons ici les instances de classe nécessaires pour pouvoir lancer le problème. Nous les avons réfléchies pour essayer d'optimiser leur praticité, mais il y a bien entendu des failles.

L'ensemble de ces composants servent à définir les différents éléments de la structure et des paramètres avant de les fournir en entrée du composant qui assemble le tout. Ce composant, présent en figure 4, assemble le tout et génère un problème mais ne le résout pas : il faut utiliser un autre composant pour cela. Nous avons choisi ce fonctionnement afin de ne pas dupliquer du code inutilement : on voulait en effet permettre la lecture et la résolution de fichier .abi, et c'est pour cela que ce composant fournit le même type d'objet que celui de lecture de fichier .abi. Ses entrées correspondent pour une part à des paramètres, et pour une part aux informations de géométrie. Toutes les informations de géométrie et une information de paramètres nécessitent de faire appel aux autres composants de la catégorie.

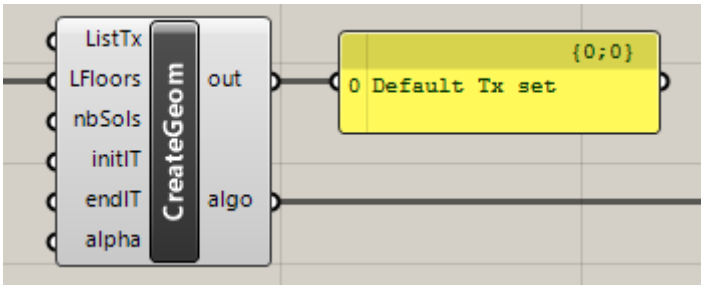

FIGURE 4 – Création du problème

Le premier composant est un composant de paramètre, visible en figure 5, et sans doute un des moins pratiques, est celui que l'on utilise pour définir les Tx de la solution, c'est-à-dire les tailles des T1, T2, T3, T4 et T5, ainsi que des limites de nombre pour chacun et une valeur pour indiquer à l'algorithme s'il doit privilégier un de ces types. Par défaut, toutes les valeurs sont initialisées à 0 à l'exception du nombre de solution à générer, qui est initialisé à 100.

Nous indiquons ici ce composant comme le moins pratique à utiliser, car le composant d'assemblage final prend une liste de tels éléments, et qu'il est important de les renseigner dans l'ordre.

Notons que le paramétrage par défaut est possible, il suffit pour l'utiliser de n'avoir aucune entrée.

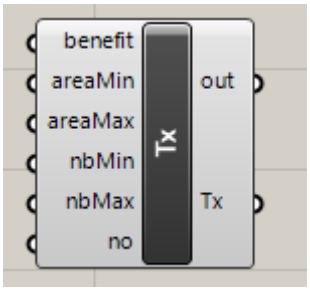

FIGURE 5 – Création de Tx

Ensuite viennent les composants de géométrie, avec tout d'abord celui pour créer des instances de la classe `Element` : ce composant est très simple d'utilisation, il suffit d'entrer la liste des points qui vont composer la surface, et de choisir parmi les quatre types de surfaces possibles avec un simple booléen `True` qu'on relit au type choisi. De plus, le composant indique en `out` le nombre de point de l'élément créé ainsi que son type,

et des indications détaillées si l'utilisateur lui a fournit des entrées qui ne lui conviennent pas. Le composant est visible en figure 6.

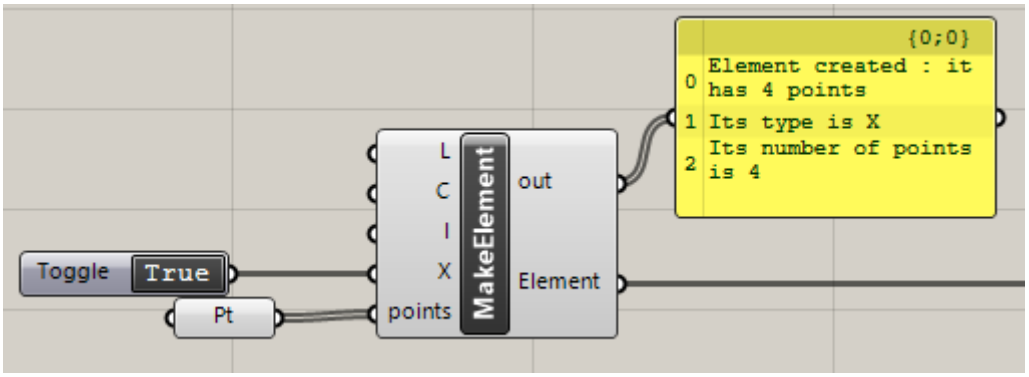

FIGURE 6 – Création d'instance de la classe `Element`

Il reste à agglomérer les éléments en étages, et on utilise pour cela le composant présenté en figure 7. Le numéro est un identifiant unique, qu'il faut définir mais dont l'importance dans l'algorithme est relative. De plus, il n'est pas obligatoire d'ordonner la liste d'étages que l'on fournit au composant de création du problème, car ce facteur n'intervient pas dans les calculs qui vont suivre.

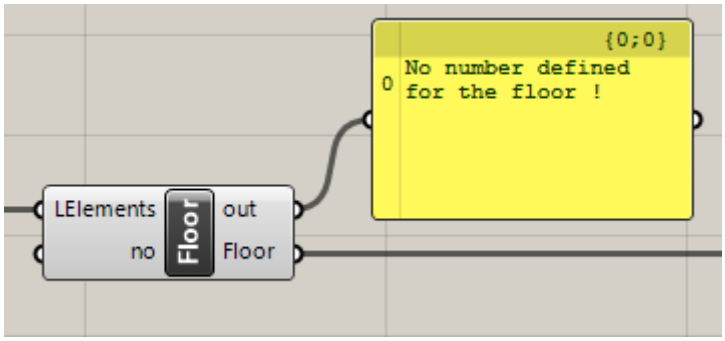

FIGURE 7 – Création d'instance de la classe `Floor` à partir d'une liste d'éléments

Il s'agit ici de la partie que nous avons pu le moins tester du fait de notre manque de connaissance sur le monde de l'architecture et d'un manque de temps. En effet, nous avons pu faire des tests simplistes, mais peu avec des exploitations de solutions derrière, tout simplement par manque de temps. C'est donc la partie la plus susceptible de nécessiter une révision.

3.3 Catégorie 3 : Visualisation et récupération de surfaces

Lorsqu'on a des instances de la classe algo, donc soit avec juste la géométrie sans solution calculée, soit avec une solution, il est nécessaire de pouvoir visualiser ce que contient la géométrie ou de regarder les solutions, puis éventuellement de récupérer les surfaces pour d'autres traitements dans Grasshopper. C'est cela que permettent les deux composants de cette catégorie.

Le premier composant sert à la visualisation du problème, il peut donc être lancé sur un problème résolu comme non résolu. Il peut servir à : vérifier qu'on a bien construit la géométrie qu'on souhaitait dans le cas où on a utilisé Grasshopper pour construire la géométrie ; vérifier un problème qu'on a importé avec l'outil d'importation de .abi.

On peut par exemple observer la figure 8, qui illustre la configuration du problème G001 de nos documentations, lu précédemment en figure 3. On peut y voir le composant de lecture, ainsi que le résultat dans Rhino. Il s'agit d'une prévisualisation à l'aide des composants Rhino, mais on constate que le composant génère des surfaces qui sont utilisables comme tout objet de ce type par Grasshopper : nous avons ici choisi une

manière de les visualiser, mais nous pourrions sinon faire un deuxième traitement suivant nos souhaits. On observe de plus que le out fournit des indications sur le nombre de chaque type de surfaces obtenues. Notez qu'il s'agit d'un cluster, qui contient une architecture interne un peu complexe, mais que nous trouvons beaucoup plus maniable sous ce format.

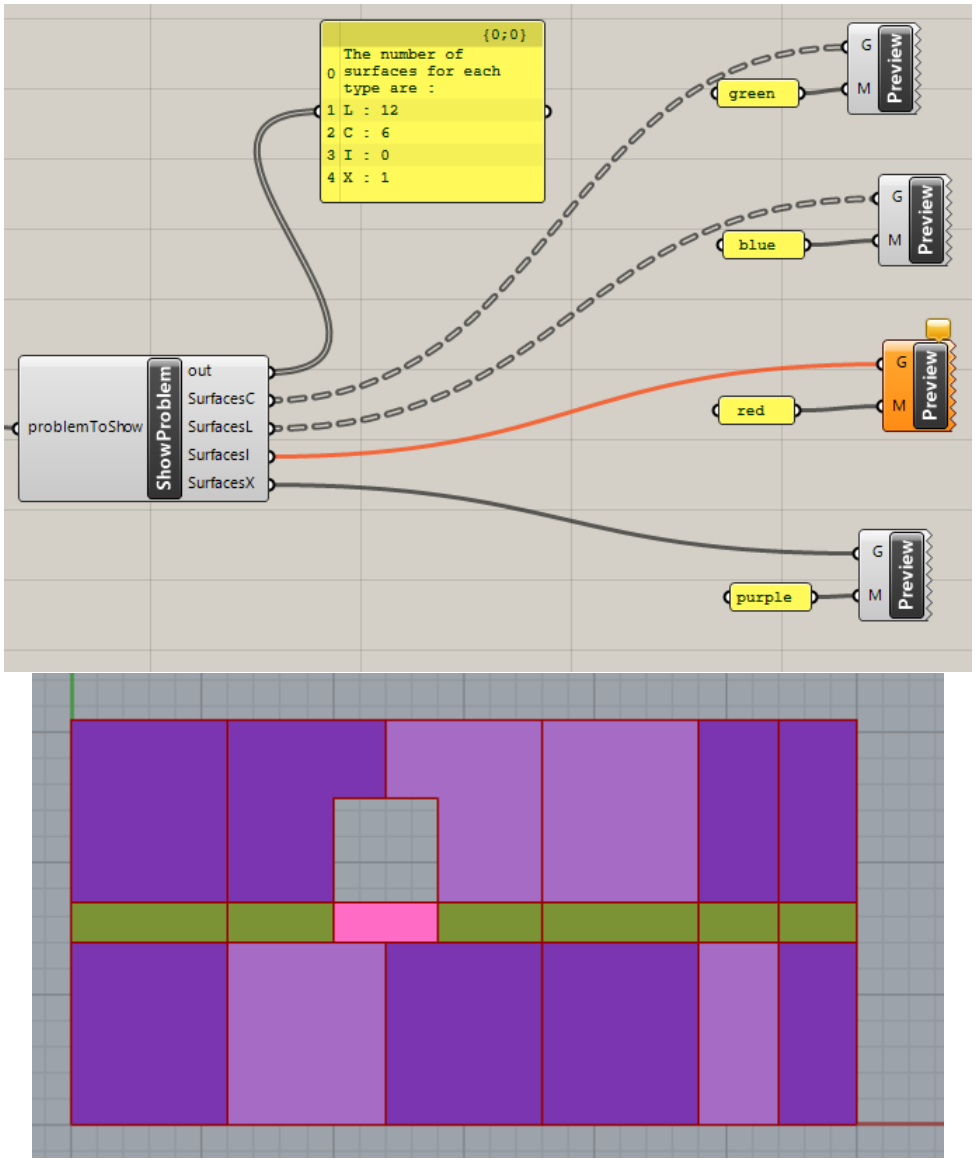

FIGURE 8 – Affichage de la géométrie du problème du fichier exemple G001

Le deuxième composant visualise les solutions. Il fonctionne de manière très similaire au précédent, avec un changement sur les types en sortie, ainsi qu'une entrée supplémentaire afin de choisir un indice de solution. Vous pouvez observer en figure 9 le choix de la première solution de G001_solved.abi, et en 10 la deuxième solution. On remarque que la géométrie ne semble pas parfaitement conservée, mais c'est un problème qui provient de l'algorithme premier et non de notre solution.

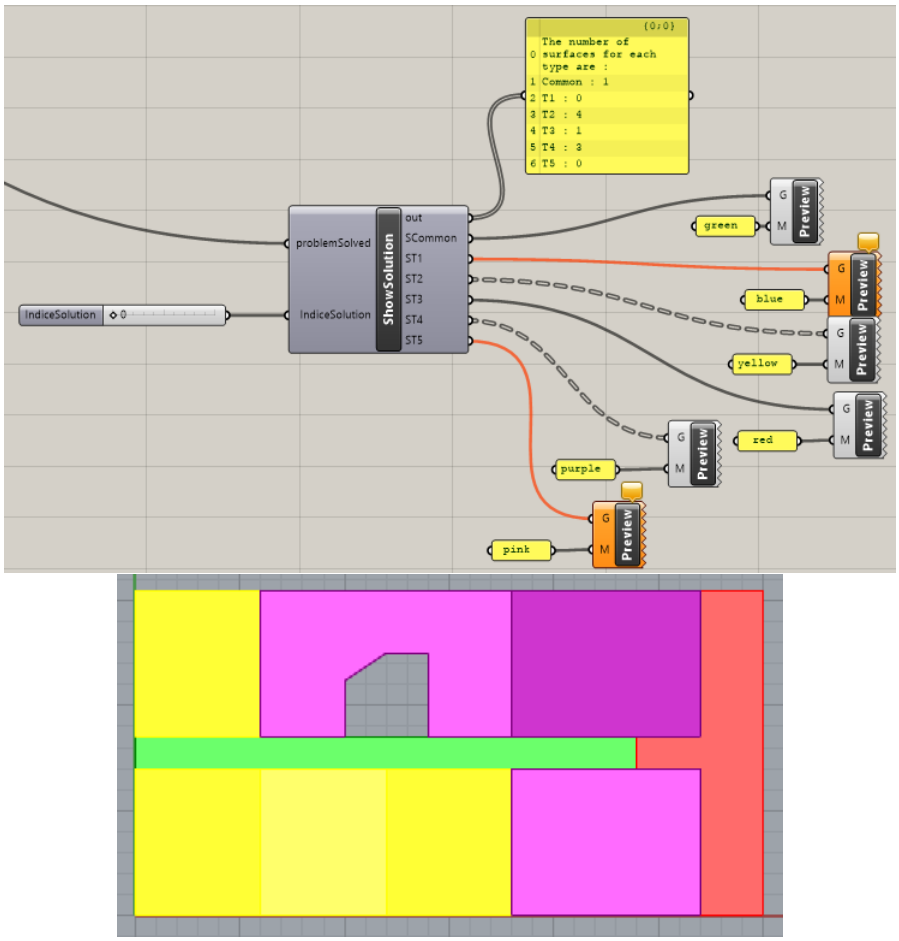

FIGURE 9 – Affichage de la solution 1 du problème du fichier exemple G001_solved.abi

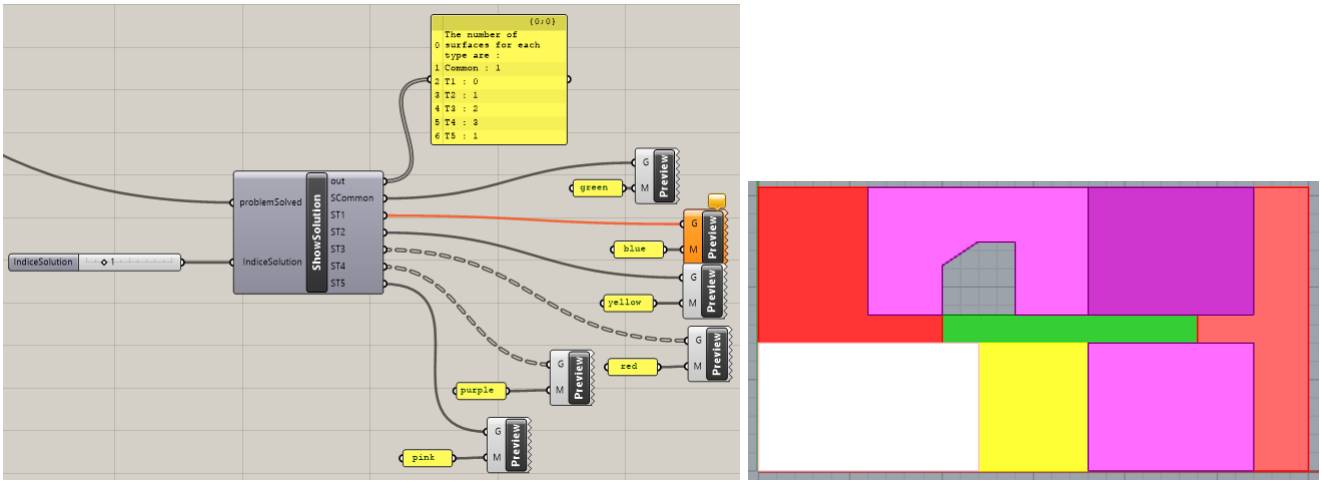

FIGURE 10 – Affichage de la solution 2 du problème du fichier exemple G001_solved.abi

3.4 Catégorie 4 : Résolution

Cette catégorie comprend un unique composant : il s'agit de celui qui fait le lien avec l'algorithme de résolution et le modèle pris en entrée. Ce bloc, visible en figure 11, est de nouveau avec un mécanisme permettant de court-circuiter l'exécution continue de Grasshopper, cette fois-ci car l'exécution prend du temps et qu'il faut éviter de la lancer plusieurs fois. Il s'agit du seul bloc où l'obtention d'une sortie n'est pas instantané, ce qui peut s'avérer perturbant mais cela ne signifie pas que cela ne marche pas. Cela peut aussi provoquer une fermeture de la fenêtre de Grasshopper, et la solution est calculée lorsqu'on réouvre la fenêtre. Il s'agit donc plutôt d'une réduction de la fenêtre que d'une véritable fermeture.

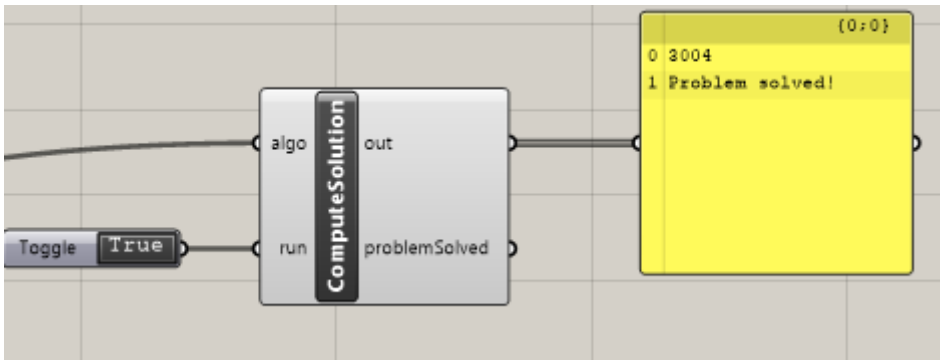

FIGURE 11 – Composant pour la résolution de problème

4 Bugs connus et limites

Pour certaines configurations données, on observe une déformation de la géométrie des différentes surfaces. Les résultats affichés sont alors à prendre avec beaucoup de recul !

La source de ce problème semble venir de l'ordre des points que Grasshopper utilise pour construire les éléments ; en cas de besoin, il est toujours possible d'exporter le fichier `.abi` depuis Grasshopper, et d'utiliser ensuite les outils d'Abita+ ou AbitaPy directement pour calculer les solutions.

Liste des figures