Titre: Title:	Génération des séquences d'assemblage d'un produit à partir de sa modélisation dans CATIA et AutoCAD
Auteur: Author:	Pierre-Emmanuel Coulon
Date:	2002
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Coulon, P.-E. (2002). Génération des séquences d'assemblage d'un produit à partir de sa modélisation dans CATIA et AutoCAD [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7039/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7039/
Directeurs de recherche: Advisors:	Kalyan Ghosh
Programme: Program:	Unspecified

UNIVERSITÉ DE MONTRÉAL

GÉNÉRATION DES SÉQUENCES D'ASSEMBLAGE D'UN PRODUIT À PARTIR DE SA MODÉLISATION DANS CATIA ET AUTOCAD

PIERRE - EMMANUEL COULON
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INDUSTRIEL)
NOVEMBRE 2002

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

GÉNÉRATION DES SÉQUENCES D'ASSEMBLAGE D'UN PRODUIT À PARTIR DE SA MODÉLISATION DANS CATIA ET AUTOCAD

présenté par : COULON Pierre - Emmanuel

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. TRÉPANIER Martin, Ph.D., président

M. GHOSH Kalyan, D.Eng, membre et directeur de recherche

Mme. CLÉROUX Louise, Ph.D., membre

À ma famille,

À mes amis.

REMERCIEMENTS

J'aimerais exprimer ma reconnaissance à tous ceux qui ont contribué à la réalisation de ce travail et qui m'ont soutenu dans ma démarche, plus particulièrement :

Kalyan GHOSH, D.Eng, mon directeur de recherche pour l'aide qu'il m'a fournie et pour la part d'aide financière qu'il m'a accordée.

Sankha DEB et Eugen COMAN, étudiants en doctorat au département de mathématiques et de génie industriel pour leurs conseils.

Isabelle BASTIEN qui m'a permis de mieux comprendre le fonctionnement du logiciel CATIA.

Alain ROBIDOUX qui m'a beaucoup aidé à résoudre plusieurs problèmes informatiques.

RÉSUMÉ

Les opérations d'assemblage constituent souvent la majeure partie des coûts de fabrication d'un produit. Le choix d'une bonne séquence d'assemblage est donc une décision centrale tant lors du développement de nouveaux produits que pour améliorer les gammes existantes. Il serait très utile pour les industriels de disposer d'un logiciel capable de générer rapidement et de façon automatique l'ensemble des séquences d'assemblage d'un produit quelconque. Pour nous dans ce projet, une séquence d'assemblage est définie par l'ordre et la direction dans laquelle on assemble chacun de ses composants.

Ces dernières années de nombreuses études ont été menées pour générer des séquences d'assemblage et les évaluer. Nous avons réussi à obtenir ces résultats directement à partir des modèles avec lesquels la majorité des industriels travaille, c'est à dire avec les dessins des produits dans CATIA ou AutoCAD. Le logiciel développé se nomme Polyassemblage. Il génère les séquences d'assemblage d'un produit à partir de sa modélisation solide, et les évalue. Il est compatible avec CATIA V5 sous Windows et AutoCAD (les versions supérieures à 14).

Polyassemblage a été programmé en Visual Basic. Il est donc de type Windows, avec des fenêtres et des barres de menus dont l'utilisation est connue de tous. Il est convivial et facile à utiliser. Enfin d'après les essais effectués sur des assemblages d'une dizaine de composants, il donne de très bons résultats et permet d'arriver rapidement aux meilleures séquences. Polyassemblage utilise les interfaces d'automatisation offertes par CATIA et AutoCAD pour extraire les informations géométriques de l'assemblage. Ces deux interfaces sont assez différentes, mais elles permettent toutes les deux à Polyassemblage d'agir sur l'ensemble des éléments du dessin. Polyassemblage peut alors rechercher des chemins d'assemblage en déplaçant pas à pas chaque solide dans une direction donnée, et à chaque pas en vérifiant que ce solide n'est pas entré en collision avec les autres solides du dessin. Avec ces données, Polyassemblage peut générer les séquences d'assemblage. L'algorithme est basé sur la méthode de Ghosh et Gottipolu (1995). Le programme assemble les composants en essayant toutes les combinaisons possibles. Il génère ainsi tous les états possibles, et le produit se construit progressivement. Avec l'algorithme original, la génération des séquences d'assemblage n'était

possible que lorsque les composants s'assemblaient selon les axes X, Y ou Z. Aguilar (1996) l'a amélioré pour autoriser l'assemblage de composants dans des directions quelconques du plan. Dans ce projet, nous avons rendu possible la génération de séquences avec toutes les directions de l'espace. Enfin, pour évaluer les séquences trouvées, le logiciel utilise des critère basés sur les informations contenues dans le dessin : regroupement des directions d'assemblage identiques pour minimiser les réorientations du produit, préférence de débuter par l'élément le plus gros, etc. Les critères d'évaluation se divisent en deux familles :

- Ceux pour réduire le nombre de séquences en éliminant celles qui ne respectent pas certaines contraintes;
- Ceux pour noter et classer les séquences restantes.

Il existe un certain nombre de restrictions sur la géométrie de l'assemblage pour que le logiciel fonctionne correctement. Il existe donc encore de grandes possibilités d'amélioration pour étendre son champ d'applications et le rendre ainsi un jour, pourquoi pas, indispensable dans l'industrie.

ABSTRACT

Assembly operations account for a large proportion of the manufacturing cost. The choice of a good assembly sequence is a very important step in development of new products, or improvement of the existing production processes.

It would be very useful for industry to make use of a software that is able to generate and evaluate automatically all the feasible assembly sequences of any product, i.e. to find the order and direction of assembling each of its components. Over the years, several studies have been undertaken to generate and evaluate assembly sequences. In this project, we have obtained the above directly from the assembly drawing in AutoCAD or CATIA. We have chosen these two CAD softwares because they are very commonly used in industry.

The application that we have developed is called Polyassemblage. It is able to generate and evaluate all feasible assembly sequences of a product given its solid representation. It is compatible with CATIA V5 on Windows and AutoCAD (releases after 14). Polyassemblage was written in Visual Basic. It looks like any other Windows based application. It has windows and menu bars, so everybody knows how it works. It is user friendly and gives good results for the examples we tested (assemblies with around 10 components). Polyassemblage uses the automation interfaces provided with CATIA and AutoCAD to extract the information about the geometry of the assembly. These interfaces are very different, but both allow Polyassemblage to act on the elements of the drawing.

The program can look for means of disassembly: it moves each element step by step in a given direction and checks that it does not collide with any of the other solids. With this information the program is able to generate all the assembly sequences. It uses the algorithm developed by Ghosh and Gottipolu (1995). The program tries to build the subassemblies starting with the individual components, and subsequently adds a component or a previously formed subassembly, until it gets the final assembly. With the original algorithm, it was possible to generate assembly sequences only if the components were assembled in the X, Y or Z directions.

Aguilar (1996) improved it by allowing the insertion of components in any planar direction. In this project, it is possible to generate assembly sequences with any direction in the space.

Finally, to evaluate the sequences that have been found, Polyassemblage uses criteria based on the geometry of the assembly: minimisation of the number of reorientations, requirement to begin with the largest solid, etc. There are two families of criteria:

- Those to reduce the number of sequences by eliminating the sequences which do not respect the constraints;
- Those to evaluate and tabulate the results.

The geometry of the assembly has to respect some restrictive conditions so that Polyassemblage works correctly. So there are still possibilities to improve it and, one day, to make it essential to the industry!

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

INTRODUCTION

Les pressions toujours plus fortes de la concurrence obligent les entreprises à développer des produits toujours mieux adaptés au marché, plus performants et moins chers. L'ingénierie simultanée a été présentée ces dernières années comme une solution pour atteindre ce but. Elle permet de concevoir des nouveaux produits plus rapidement et mieux adaptés aux besoins des consommateurs. Les équipes de développement sont devenues des équipes pluridisciplinaires. Elles sont constituées de membres de tous les secteurs de l'entreprise : ingénieurs de développement bien sûr, mais aussi membres de la production, du service des ventes, etc. Le développement des nouveaux produits est ainsi réalisé en tenant compte à tous les niveaux de toutes les contraintes, et en particulier pour ce projet des coûts de fabrication.

L'assemblage d'un produit intervient dans 70 à 80% des coûts totaux de production (Boothroyd, 1994). Pour obtenir un produit à meilleur coût, il est très important de savoir choisir une bonne séquence d'assemblage, c'est à dire de connaître l'ordre et la direction d'assemblage des composants qui minimise les coûts d'assemblage. Les équipes pluridisciplinaires doivent tenir compte de ce problème très tôt dans le processus de développement. Beaucoup de travaux ont été réalisés ces dernières années pour générer toutes les séquences d'assemblage d'un produit de manière rapide et automatique. Les programmes développés sont généralement aussi capables d'évaluer ces séquences, et aident à identifier les meilleures. Le plus souvent les données nécessaires pour faire fonctionner ces programmes sont fournies directement par l'utilisateur et portent sur la géométrie de l'assemblage et sa structure.

Il serait utile d'obtenir ces résultats directement à partir du dessin d'assemblage dans son format d'origine, c'est-à-dire obtenu à partir d'un logiciel de Conception Assistée par Ordinateur (CAO) répandu dans l'industrie. Nous proposons dans ce projet un programme qui réalise cette tâche : il génère et évalue toutes les séquences d'assemblage d'un produit à partir de son dessin dans CATIA ou AutoCAD. Le résultat est presque immédiat, donc il peut être utilisé en temps réel par l'équipe pluridisciplinaire.

Ce programme, nommé Polyassemblage, ne peut pas remplacer l'expertise humaine ni ses connaissances. Mais il peut apporter une bonne idée de ce à quoi ressemblera la séquence d'assemblage finale, et mettre tôt en évidence des problèmes qui pourront gêner l'assemblage du produit. Il n'est donc pas là pour remplacer l'ingénieur qui décide des gammes d'assemblage, mais pour le supporter dans sa démarche. L'outil développé fournit un élément de réflexion supplémentaire pour encourager une décision rationnelle et éclairée puisque celle-ci sera basée sur des données objectives et non subjectives.

Polyassemblage a été développé en Visual Basic 6.0. Il utilise les interfaces d'automatisation offertes par les plus récentes versions d'AutoCAD et de CATIA pour extraire les informations sur la géométrie et la structure des composants. Pour générer les séquences d'assemblage à partir de ces trajectoires l'algorithme de Ghosh et Gottipolu (1995) a été programmé. Cet algorithme a été amélioré par Aguilar (1996) et adapté au programme dans ce projet. Enfin, Polyassemblage est capable d'évaluer et de classer les séquences trouvées selon plusieurs critères de base. Ce tri facilite grandement le choix d'une séquence finale par l'utilisateur.

Dans ce rapport, nous allons présenter les différents aspects théoriques et pratiques qui décrivent le logiciel. Le plan du rapport suit les étapes franchies pour mettre au point Polyassemblage :

- Dans le chapitre 1, nous allons tout d'abord présenter une série de travaux qui ont été effectués à travers le monde ces dernières années dans le domaine de la génération des séquences d'assemblage.
- Dans le chapitre 2, nous allons montrer comment il est possible, d'un point de vue informatique, d'utiliser les fonctionnalités de CATIA et d'AutoCAD depuis un programme extérieur, dans notre cas écrit en Visual Basic.
- Dans le chapitre 3, nous allons expliquer précisément comment fonctionne l'algorithme pour générer les séquences d'assemblage.
- Dans le chapitre 4, nous allons montrer comment trouver des chemins d'assemblage ou de désassemblage pour les différents composants. Ces données sont nécessaires pour générer les séquences d'assemblage.

- Ensuite dans le chapitre 5, nous présenterons les différents critères programmés pour trier et classer les séquences trouvées précédemment.
- Dans le chapitre 6, nous présenterons d'autres aspects de Polyassemblage et des outils supplémentaires qui rendent le programme plus convivial, plus facile à utiliser et qui simplifie la compréhension des résultats et le choix d'une séquence finale.
- Enfin dans le chapitre 7, nous montrerons les résultats obtenus avec différents exemples de la vie courante : un alternateur d'automobile et le système de frein d'un vélo. Nous accompagnerons ces résultats d'observations sur le fonctionnement du programme et les temps de calculs.
- Nous conclurons avec un bilan des performances et des limites du programme. Nous parlerons aussi des différentes voies d'amélioration qu'il sera intéressant d'explorer dans le futur.

CHAPITRE 1 – REVUE DE LA LITTÉRATURE

1-1 Introduction

La sélection d'une séquence d'assemblage efficace est une activité importante dans la planification du processus d'assemblage. Beaucoup de recherches ont été effectuées et sont encore effectuées pour permettre de générer les séquences possibles d'un produit et de les évaluer.

Généralement, le processus est organisé en 3 étapes. Ces étapes sont plus ou moins indépendantes selon l'approche développée.

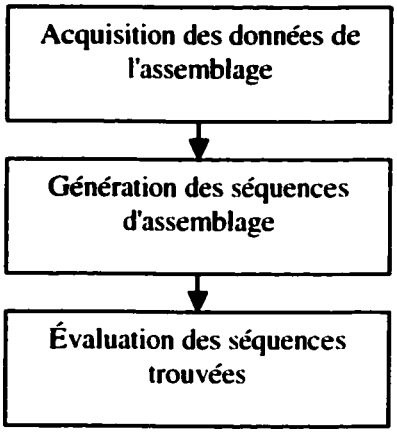


Figure 1.1 : Étapes du processus de génération des séquences d'assemblage

L'acquisition des données : les données à recueillir doivent permettre de générer toutes les séquences d'assemblage. Beaucoup d'informations sont nécessaires et elles dépendent de l'algorithme programmé : on peut avoir besoin des relations de précédence (définies plus bas), du type de liaison entre les composants, de la description géométrique des solides, etc.

La génération des séquences d'assemblage : un très grand nombre d'approches différentes coexistent, comme nous allons le voir plus bas.

L'évaluation des séquences trouvées et leur sélection : cette étape peut être réalisée en même temps que la génération des séquences (en particulier avec certains algorithmes génétiques), mais elle en est le plus souvent indépendante.

1-2 Acquisition des données sur l'assemblage

1-2.1 Acquisition manuelle

Dans la plupart des articles identifiés, les données sont entrées manuellement par l'utilisateur. La collecte manuelle des données est plus facile à mettre en œuvre dans la mesure où elle met de côté tout l'aspect lié à l'analyse automatique de la géométrie et de la structure de l'assemblage. Seul l'algorithme de génération des séquences est développé.

1-2.2 Acquisition mixte

Dans d'autres cas, la collecte des données est mixte :

- Extraction automatique d'un certain nombre de renseignements géométriques directement à partir du dessin de l'assemblage;
- Indication des autres renseignements par l'utilisateur (par exemple au sujet du processus de fabrication, des outillages nécessaires ou du type de liaison entre les composants).

Dans ces cas, l'assemblage est modélisé avec un logiciel de CAO particulier. Nous pouvons citer par exemple :

- L'interface graphique utilisée par Mascle et Balasoiu (2001) se nomme « *3D Tool Kit* » . Cet outil de modélisation utilise la représentation *B-rep* (*Boundary Representation*, représentation des solides par leurs frontières) qui est bien adaptée à l'algorithme développé.
- Le programme présenté par Romney et al. (1995) pour la génération des séquences est lui-même capable de modéliser les solides.

1-2.3 Acquisition automatique et directe

Dans aucun des articles recensés, l'extraction des données n'a été faite directement depuis un logiciel de CAO répandu dans l'industrie comme AutoCAD, CATIA ou ProEngineer. Les approches développées sont donc en général peu pratiques à utiliser dans l'industrie.

Les auteurs Pandey et Sarvananthen (1999) ont utilisé les données contenues dans un fichier ".dxf" (créé par AutoCAD), mais le programme a été limité aux assemblages qui possèdent une symétrie centrale autour d'un axe, en l'occurrence l'axe X.

1-3 Génération des séquences d'assemblage

Il existe une très grande variété d'approches pour générer les séquences d'assemblage. Elles se différentient par exemple par :

- Les informations nécessaires pour générer les séquences : utilisation de la géométrie des solides (Romney et al. (1995)), des connecteurs (Tsen et Li (1999)), des relations de précédence (Bourjault (1984))...
- La représentation interne des données : utilisation de matrices (Ghosh et Gottipolu (1995)), de graphes (Senin et al. (2000)), stockage des séquences sous forme « génétique » (Fujimoto et Sebaaly (2000))...
- L'approche du point de vue informatique : éventuellement recours à l'intelligence artificielle (Choi et al. (1998)).
- La complétude des résultats : les approches génèrent toutes les séquences (Ghosh et Gottipolu (1995)) ou seulement quelques-unes (ce qui est moins coûteux en temps, comme Senin et al. (2000)).

Ces différents aspects sont développés dans les parties suivantes. Nous allons tout d'abord montrer les schémas les plus courants pour représenter la structure des produits et leurs séquences d'assemblage. Ensuite nous décrirons les différentes approches qui existent pour générer ces séquences d'assemblage.

1-3.1 Représentations de la structure des produits et de leurs séquences d'assemblage

Le diagramme de liaison

Le diagramme de liaison permet de représenter la structure d'un assemblage. Les nœuds représentent les composants, et les traits indiquent les contacts entre eux. Le diagramme de liaison d'une roue de vélo est montré à la figure 1.2. Les rayons de la roue et leurs vis ont été considérés comme des composants uniques pour simplifier le diagramme.

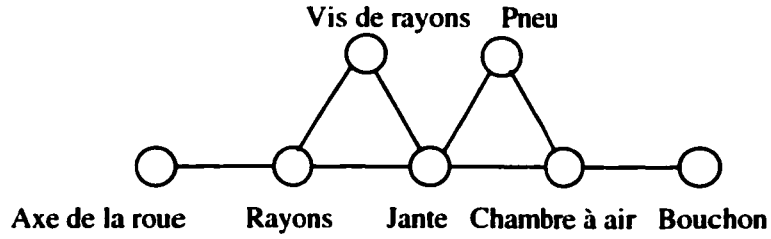


Figure 1.2 : Diagramme de liaison pour une roue de vélo

Historiquement, Bourjault (1984) a été l'un des premiers à utiliser ce diagramme pour générer des séquences d'assemblage. Il a pu le faire avec les relations de précédence du produit. Les relations de précédence sont des relations qui indiquent quelles liaisons doivent être réalisées en priorité pour permettre aux autres de se faire (par exemple il faut mettre la chambre à air dans le pneu avant de monter le pneu sur la jante). De Fazio et Whitney (1987) ont ensuite amélioré sa méthode et réduit le nombre de questions posées de 2^n à 2n (n est le nombre de composants).

Le « AND/OR Graph »

Pour représenter les séquences d'assemblage Homem de Mello et Sanderson (1990) ont proposé un graphe nommé *AND/OR Graph*. Ce graphe représente l'ensemble des séquences d'assemblage d'un produit.

Chaque nœud représente un état, c'est-à-dire un sous-assemblage intermédiaire constitué d'un ou de plusieurs composants. Ce graphe devient vite très grand en taille car le nombre de nœuds augmente de façon exponentielle avec le nombre de composants. Par contre ses principaux intérêts sont :

- Il regroupe à lui seul toutes les séquences possibles;
- Il est possible d'identifier très rapidement les sous-assemblages impliqués dans les séquences d'assemblage.

Ci-dessous se trouve un exemple de *AND/OR Graph* publié par Senin et al. (2000).

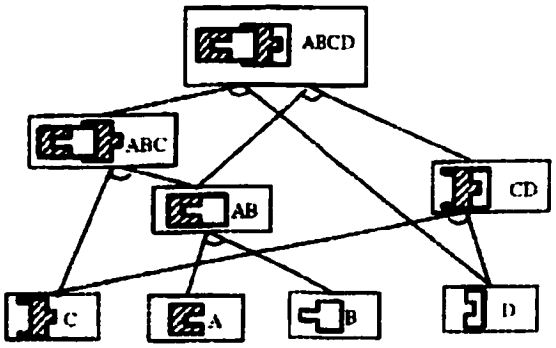


Figure 1.3 : *And/Or Graph* pour l'assemblage {A, B, C, D}

Le « Cluster graph »

Les auteurs O'Shea et al. (2000) ont présenté un autre type de graphe pour représenter les séquences d'assemblage. Le graphe est structuré en niveaux qui correspondent chacun à une « enveloppe » du produit.

- Niveau 1 : liste des composants qu'on peut désassembler directement,
- ...
- Niveau n : liste des composants qu'on peut désassembler lorsque ceux des niveaux antérieurs ont été ôtés,

Nous pouvons illustrer la notion de *cluster graph* avec l'exemple ci-dessous.

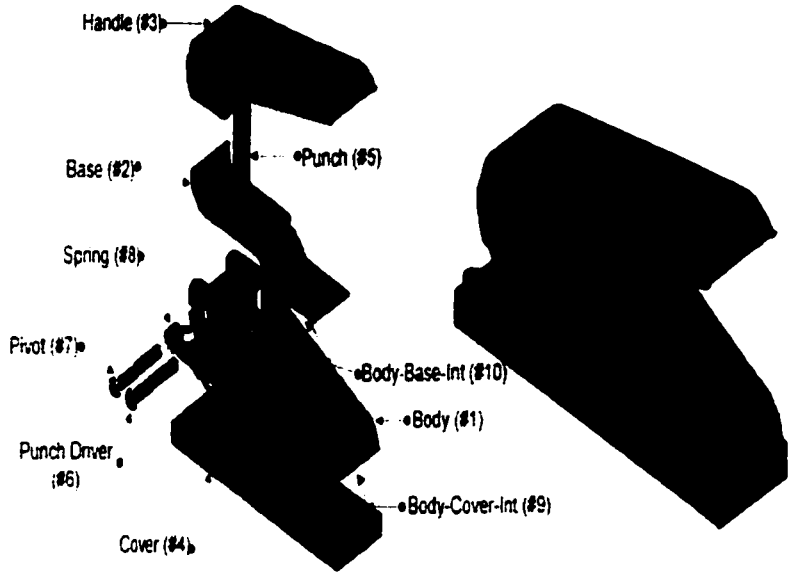


Figure 1.4 : Structure de l'agrafeuse

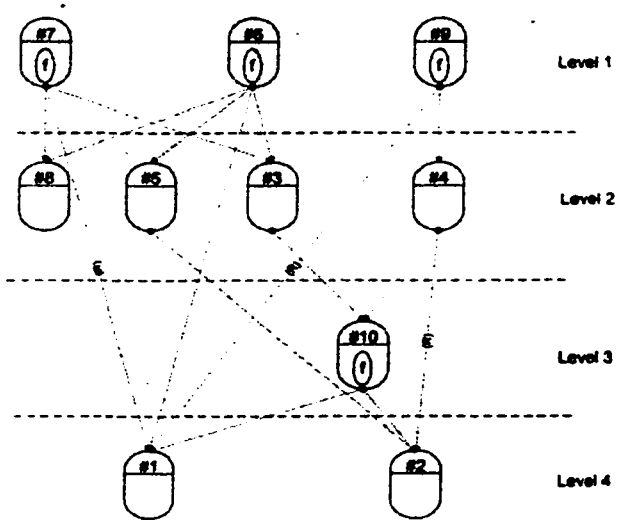


Figure 1.5 : *Cluster Graph* pour l'agrafeuse

Ce graphe peut contenir aussi d'autres informations : liste des contraintes de précédence, procédure pour opérer le désassemblage d'un composant, liste des outils et des

équipements nécessaires, etc. Il permet de lire facilement les séquences d'assemblage et de les comparer entre elles.

Autres graphes

Il existe beaucoup d'autres représentations possibles :

- Lanham et Dialami (2001) ont défini *l'Assembly State Vector* (vecteur qui décrit chaque étape de la séquence d'assemblage). Ce vecteur permet de coder de manière unique chaque séquence. Avec cette représentation les ingénieurs de production peuvent opérer facilement des « micro changements » à l'intérieur d'une séquence sans que cela n'ait trop de répercussions sur la « macro séquence ».
- Zha et al. (1998) ont aussi cité le graphe de connectivité, le graphe des contraintes d'assemblages, et bien d'autres encore.

Les différents graphes sont des outils pour visualiser la structure d'un produit ou sa séquence d'assemblage. Le graphe le plus utile est donc celui qui s'adapte le mieux à l'algorithme développé dans la recherche. De nombreux graphes sont spécifiques à certaines approches, et tous ne peuvent pas être décrits dans ce projet.

1-3.2 Les systèmes experts

Nous allons présenter dans cette partie les algorithmes de génération et d'évaluation des séquences d'assemblage basés sur les systèmes experts. Les systèmes experts sont une branche de l'intelligence artificielle. Ils permettent de stocker l'expertise humaine dans des bases de données pour la réutiliser. Les algorithmes mis en œuvre avec les systèmes experts sont actuellement les plus proches des raisonnements humains.

Choi et al. (1998) ont présenté la base de données qui contient toutes les informations relatives à l'assemblage étudié. La *KALG* (*Knowledge Assembly Liaison Graph*) contient la liste des liaisons et leur type, la liste des contraintes géométriques et de stabilité, les différents coûts,

etc. Ces informations sont enregistrées sous la forme suivante (où c1, c2 et c3 sont des composants de l'assemblage) :

- Renseignements sur la structure de l'assemblage : "*connect (c1, c3)*" ; "*before (c1, c2)*" ; "*right (c2, c3)*"...
- Problèmes liés à certains états ou séquences : "*unstable ([c1, c2, c3])*"...
- Renseignements pour l'évaluation des séquences trouvées : "*time_consuming (c1, [c2, c3])*", "*cost_effective (c1, c2)*"...

À partir de ces données, le programme écrit en Prolog détermine automatiquement les séquences possibles et les évalue.

Osetrov (1998) a utilisé la logique des prédicats pour générer les séquences d'assemblage. Les données à indiquer initialement au programme sont, par exemple, la géométrie de l'assemblage, le nom et la position des composants, etc. Ce programme est aussi capable de trouver des séquences d'assemblage non linéaires, c'est-à-dire qui passent par des sous-assemblages.

1-3.3 Regroupement en familles de produits

Dans cette partie, les algorithmes présentés utilisent les analogies entre les produits d'une même famille pour obtenir des bonnes séquences avec un minimum de calculs.

Tous les éléments d'une même famille se ressemblent. Ils ont la même structure et souvent les mêmes composants. Les séquences d'assemblage sont donc nécessairement voisines. Pour chaque produit, il est possible de dériver la séquence d'assemblage de la séquence « modèle » de la famille.

Wolter et Chakrabarty (1997) ont présenté une « librairie de structures prédéfinies ». Lorsqu'un produit a des ressemblances avec un produit déjà enregistré, il n'est pas nécessaire de recalculer sa séquence optimale car elle est similaire à une séquence enregistrée. Avec ce procédé d'autres informations sur la gamme d'assemblage sont aussi disponibles, telles que les temps d'assemblages, les outils nécessaires, etc.

La figure ci-dessous donne un exemple de diagramme de liaison représentant la structure de deux produits d'une même famille.

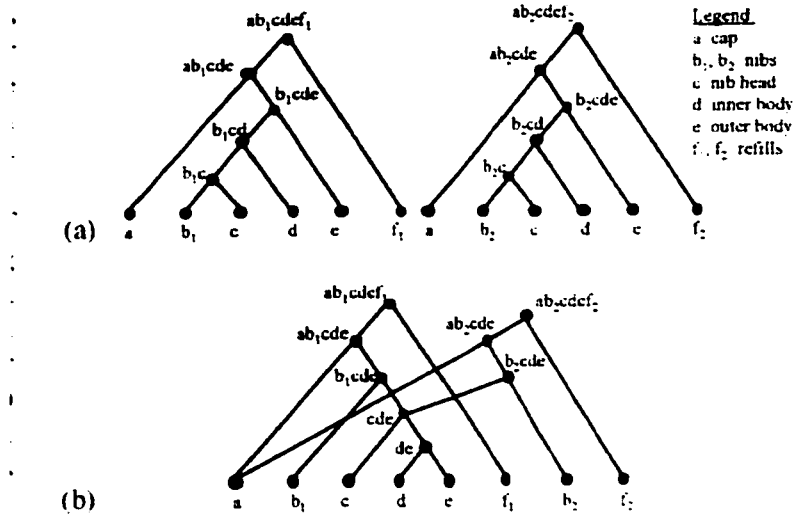


Figure 1.6 : Diagramme de liaison pour 2 stylos

a) Deux diagrammes séparés b) Un diagramme commun

Gupta et Krishnan (1998) ont développé un nouveau diagramme pour obtenir des sous-assemblages communs à toute la famille, le PFID (*Product Family Interconnection Diagram*). Les noeuds du diagramme représentent les composants de tous les éléments de la famille, et ses arcs représentent leurs liaisons. Le programme supprime les arcs qui ne sont pas associés à tous les éléments de la famille. Il ôte ensuite les nœuds déconnectés. Si les sous graphes restants peuvent être assemblés en respectant toutes les relations de précédence, alors ils représentent des bons sous-assemblages communs.

Fouda et al. (2001) ont présenté un algorithme pour générer un unique *AND/OR Graph* par famille de produits. Chacun des éléments de la famille possède des composants propres qui apparaissent aussi dans le *AND/OR Graph*.

Enfin Martinez et al. (2000) ont proposé de créer un produit « virtuel » qui ressemble le plus possible à tous les autres éléments de la famille de produits.Une fois ce produit choisi, le

logiciel ISASG (*Integrated Software for Assembly System Generation*, Logiciel intégré pour la génération des séquences d'assemblage) génère toutes les séquences d'assemblage possibles et l'utilisateur choisit la meilleure séquence à l'aide de critères programmés.

- Minimum de réorientations,
- Modularité,
- Temps d'assemblage.

1-3.4 Utilisation des connecteurs

Les connecteurs sont des composants qui servent à faire tenir d'autres composants de manière stable (vis, rivets, etc.). C'est d'ailleurs généralement leur seule utilité. Il est possible de décrire la structure de la plupart des assemblages uniquement à partir des connecteurs. Des recherches existent donc qui utilisent ces connecteurs pour générer les séquences d'assemblage. Ces composants sont alors plus considérés comme des caractéristiques de l'assemblage, que comme des composants à part entière.

L'algorithme de génération des séquences d'assemblage présenté par Tsen et Li (1999) est entièrement basé sur l'utilisation des connecteurs. Les trois étapes de l'algorithme sont :

- Définition des connecteurs : chaque connecteur représente une connexion et a pour paramètres les composants impliqués dans la connexion. Chaque composant de l'assemblage est associé à un et un seul connecteur.
- Recherche d'une séquence d'assemblage linéaire : le programme crée les séquences d'assemblage possibles en se basant sur la liste des connecteurs. Ci-dessous se trouve une illustration de cette approche.

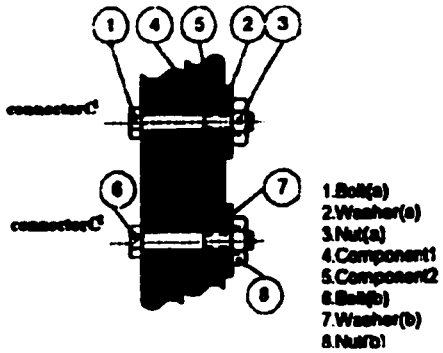


Figure 1.7 : Assemblage adapté à l'approche des connecteurs

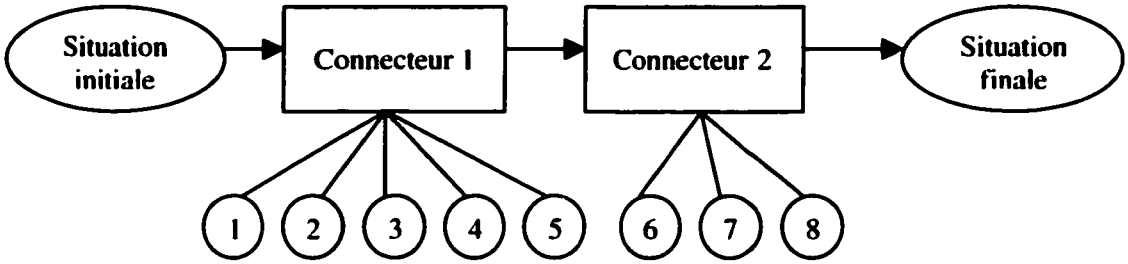


Figure 1.8 : Graphe d'une séquence d'assemblage basé sur les connecteurs

- Recherche d'une séquence non linéaire : à partir des séquences obtenues précédemment l'algorithme cherche quels connecteurs pourraient être assemblés en parallèle.

Van Holland et Bronsvoort (1996) n'ont pas basé la génération des séquences sur les connecteurs, mais ils les ont utilisés pour trouver des relations de précédence supplémentaires. Le but est de faciliter la génération des séquences et de réduire leur nombre final. Les données à indiquer au programme se regroupent en deux catégories.

- « Caractéristiques de tenue » : liste des préhenseurs possibles, positions de saisie des composants...
- « Caractéristiques des connexions » : liste des composants tenus par la connexion, points d'insertion, types de connexion, directions d'assemblage...

L'approche présentée par Zhao et Masood (1999) est aussi une approche mixte. Les données de base de l'algorithme sont stockées dans 4 graphes différents qui indiquent :

- La liste des composants et des liaisons, les types de liaison, les directions de désassemblage, etc;
- La liste des connecteurs, leur type et le nombre de composants impliqués dans chaque connexion;
- Des informations sur le processus de production (calibrages, tests, etc.);
- Des informations sur les tolérances de l'assemblage.

L'identification de la meilleure séquence d'assemblage se fait ensuite en 2 étapes :

1) Pour chaque composant, calcul du « coefficient de contrainte »;
2) Recherche de la séquence qui a la plus petite somme des coefficients de contrainte (c'est la séquence optimale).

1-3.5 Le NDBG (Non Directional Blocking Graph)

Le NDBG (Graphe des directions non bloquantes) est une représentation introduite par Wilson (1992) qui contient l'ensemble des directions de désassemblage localement pour l'ensemble des composants d'un produit. Cette approche est limitée aux composants polyédriques.

Romney et al. (1995) ont présenté l'utilisation de ce concept pour développer STAAT, un programme qui permet de trouver des séquences d'assemblage et de désassemblage. Le programme construit le NDBG du produit à partir du dessin tridimensionnel de l'assemblage, puis il en génère les séquences possibles.

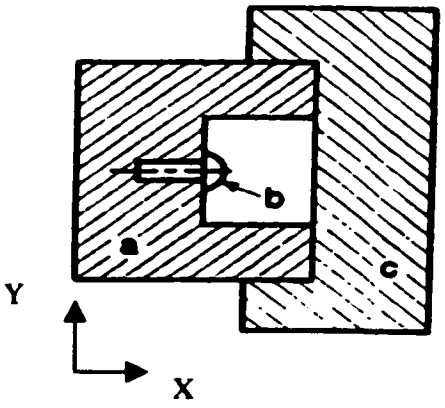
Latombe et Wilson (1995) ont pu générer des séquences d'assemblage pour des produits comportant des tolérances. L'algorithme est limité à un certain nombre de cas. Il ne fonctionne que pour les assemblages en deux dimensions et les tolérances sont données uniquement sur la longueur des côtés. Deux cas se présentent :

- La procédure d'assemblage / désassemblage est possible à coup sûr. Les séquences sont alors générées à l'aide d'un NDBG modifié, le « strong NDBG ».
- Pour certaines dimensions l'assemblage est impossible. Le programme cherchera alors dans quels cas il est possible d'assembler le produit et dans quels cas il ne l'est pas. L'algorithme utilise une autre version du NDBD, appelée « weak NDBG ».

1-3.6 Utilisation des matrices de contact et de translation

L'intérêt des approches qui utilisent les matrices de contact et de translation est qu'elles sont rigoureuses d'un point de vue mathématique et faciles à comprendre et à programmer.

Ghosh et Gottipolu (1995) ont développé un algorithme pour générer des séquences d'assemblage à partir de ces deux types de matrices. À chaque paire de composants ($Comp_i$, $Comp_j$) sont associées une valeur de contact ($C(Comp_i, Comp_j) = 1$ si les composants se touchent, 0 sinon) et une matrice de translation T. $T(Comp_i, Comp_j)$ indique si $Comp_j$ peut se déplacer sans rentrer en collision avec $Comp_i$ dans chacune des 6 directions principales de l'espace : X+, Y+, Z+ et X-, Y-, Z-.



Matrices de contact :

$C(a, b) = 1$

$C(a, c) = 1$

$C(b, c) = 0$

Matrices de translation

$T(a, b) = (1, 0, 0, 0, 0, 0)$

$T(a, c) = (1, 0, 0, 0, 0, 0)$

$T(b, c) = (1, 1, 1, 0, 1, 1)$

Figure 1.9 : Illustration de l'algorithme de Ghosh et Gottipolu

À partir de ces matrices, le programme est capable de générer toutes les séquences d'assemblage du produit. L'algorithme sera présenté en détails dans le chapitre 3. Aguilar (1996)

a amélioré la méthode en permettant de générer les séquences aussi lorsque des composants doivent être assemblés selon des directions quelconques du plan.

Eng et al. (1999) ont présenté un programme qui utilise une variante de cette méthode. Leurs matrices contiennent 12 données : 6 degrés de liberté pour les translations et 6 pour les rotations. À l'aide de ces matrices, ils ont pu déterminer la « meilleure » séquence d'assemblage d'un produit. À chaque étape le programme cherche quel composant est le meilleur candidat à désassembler selon les critères suivants :

- Facilité à manipuler,
- Regroupement des composants de même type (pour éviter les changements d'outils),
- Stabilité du sous-assemblage obtenu.

L'article suppose implicitement qu'en enlevant chaque fois le meilleur composant on obtient la meilleure séquence de désassemblage, et donc la meilleure séquence d'assemblage.

Enfin, S. Smith et G. Smith (2001) ont perfectionné la méthode en donnant d'autres valeurs possibles aux éléments de la matrice de contact :

- 0 si il n'y a pas de contact,
- 1 si il y a un simple contact,
- 2 si le contact est « fort », c'est-à-dire si les composants peuvent tenir de manière stable entre eux.

Cette information supplémentaire permet de ne générer que des séquences qui contiennent des assemblages stables.

1-4 Évaluation des séquences

Lorsqu'on ne tient compte que des contraintes géométriques, on obtient facilement un grand nombre de séquences d'assemblage pour le produit. Il est donc nécessaire de développer des procédures pour le réduire.

Certaines approches consistent à évaluer complètement toutes les séquences pour ensuite faire le bon choix :

- En supprimant les séquences qui ne correspondent pas à certains critères;
- En évaluant et triant les séquences restantes, qui peuvent encore être nombreuses !

D'autres consistent à trouver simplement des séquences presque optimales. Dans ces cas, toutes les séquences ne sont pas évaluées (Güngör et Gupta (2001), et plus généralement tous les algorithmes génétiques).

Nous allons présenter ci-dessous les critères les plus répandus pour évaluer les séquences d'assemblage. Ensuite nous décrirons plusieurs approches qui permettent d'identifier les meilleures d'entre elles.

1-4.1 Différents critères d'évaluation

Les critères d'évaluation qu'on rencontre le plus fréquemment dans la littérature sont regroupés en deux catégories : les facteurs qualitatifs et les facteurs quantitatifs. Certains de ces facteurs ont été déjà vus. Zha at al. (1998) en ont proposé une liste non exhaustive.

Parmi les facteurs qualitatifs on trouve :

- La fréquence des réorientations;
- La stabilité des sous-assemblages;
- La modularité des sous-assemblages.

Et parmi les facteurs quantitatifs :

- Le temps total incluant les opérations d'assemblage, de manutention, de changement d'outils;
- Le coût total incluant la main d'œuvre, les outillages, etc.

Souvent les critères d'évaluation sont intégrés au programme qui génère les séquences d'assemblage. Il existe aussi des programmes dédiés à l'évaluation des séquences.

1-4.2 Le programme Archimède

Décrit par Jones et al. (1998), le programme permet de sélectionner les séquences d'assemblage qui répondent à une liste de contraintes fixées par l'utilisateur.

Ces contraintes sont de deux types :

- Contraintes d'interdictions (par exemple, interdiction de certains sous-assemblages ou certaines opérations);
- Contraintes d'obligation (forcer une direction d'assemblage, passer par un sous-assemblages spécifique, etc.).

L'utilisateur affine la liste des contraintes au fur et à mesure que le projet avance.

Lorsque toutes les contraintes sont fixées, l'utilisateur choisit la ou les séquences finales par optimisation :

- Maximisation (de la linéarité de la séquence...);
- Minimisation (des coûts, du nombre de réorientations nécessaires...).

Le choix final revient toujours à l'utilisateur. Nous sommes encore très loin de pouvoir modéliser tous les critères qui entrent en compte pour trouver la meilleure séquence (dont l'expérience de travail). Les outils actuels restent des outils qu'il est nécessaire de bien comprendre pour les utiliser de manière pertinente.

1-4.3 Programmation d'un réseau de Pétri

Les réseaux de Pétri sont des outils d'optimisation qui ont été programmés par Yee et Ventura (1999) pour évaluer les séquences d'assemblage.

L'algorithme détermine tout d'abord le réseau de Pétri à partir du *AND/OR Graph* et des informations qui lui permettront d'optimiser les séquences d'assemblage.

Ces données sont stockées sous forme matricielle, et le problème revient à un problème d'optimisation mathématique : il faut minimiser la fonction de coût tout en respectant les contraintes fixées par l'utilisateur.

1-4.4 Les algorithmes génétiques

Ce domaine de l'intelligence artificielle est de plus en plus utilisé pour évaluer les séquences d'assemblage. Les résultats obtenus jusqu'à maintenant indiquent que les algorithmes génétiques sont légèrement moins fiables que l'approche combinatoire pour trouver la meilleure séquence, mais ils sont capables d'identifier des solutions très proche en un temps record lorsque la taille du problème augmente.

Principe de fonctionnement

Comme indiqué par Senin et al. (2000), les algorithmes génétiques sont des méthodes d'optimisation qui adoptent des stratégies de recherche imitant les mécanismes de la sélection naturelle. De même que les êtres vivants ont leurs caractéristiques physiques codées dans leurs gènes, les algorithmes génétiques codent chaque séquence d'assemblage dans le génome d'un individu virtuel.

Initialement, on part d'un petit nombre de séquences d'assemblage générées aléatoirement, mais toutes possibles. L'ordinateur fait ensuite les opérations suivantes :

- *Reproduction :* à chaque itération, il génère une nouvelle séquence à partir de deux séquences parentes. La séquence générée aura peut-être les pires caractéristiques des deux parents, auquel cas elle sera oubliée par le programme. Elle en aura peut-être les meilleures, auquel cas nous nous serons rapprochés de la solution optimale ;

- *Chevauchement :* après la reproduction l'ordinateur mélange certaines portions des meilleures séquences filles entre elles. De nouveau si le résultat est pire, il

sera aussi oublié. Sinon nous nous serons encore approchés de la meilleure solution;

- *Mutation :* le programme modifie aléatoirement certaines séquences. Le but est d'éviter d'échouer dans des optimums locaux (au cas où certaines portions de la meilleure séquence n'étaient pas présentes dans la population initiale).

Pour évaluer les séquences générées, il existe une fonction appelée « fonction d'évaluation ». Cette fonction permet d'évaluer les séquences utilisées et de savoir si les nouvelles séquences générées sont meilleures ou moins bonnes que les séquences d'origine, selon une liste de critères définis dans le programme.

Après un certain nombre d'itérations, l'algorithme s'arrête et le meilleur individu trouvé selon la fonction d'évaluation représente la séquence définitive.

L'algorithme fondamental des algorithmes génétiques est présenté ci-dessous.

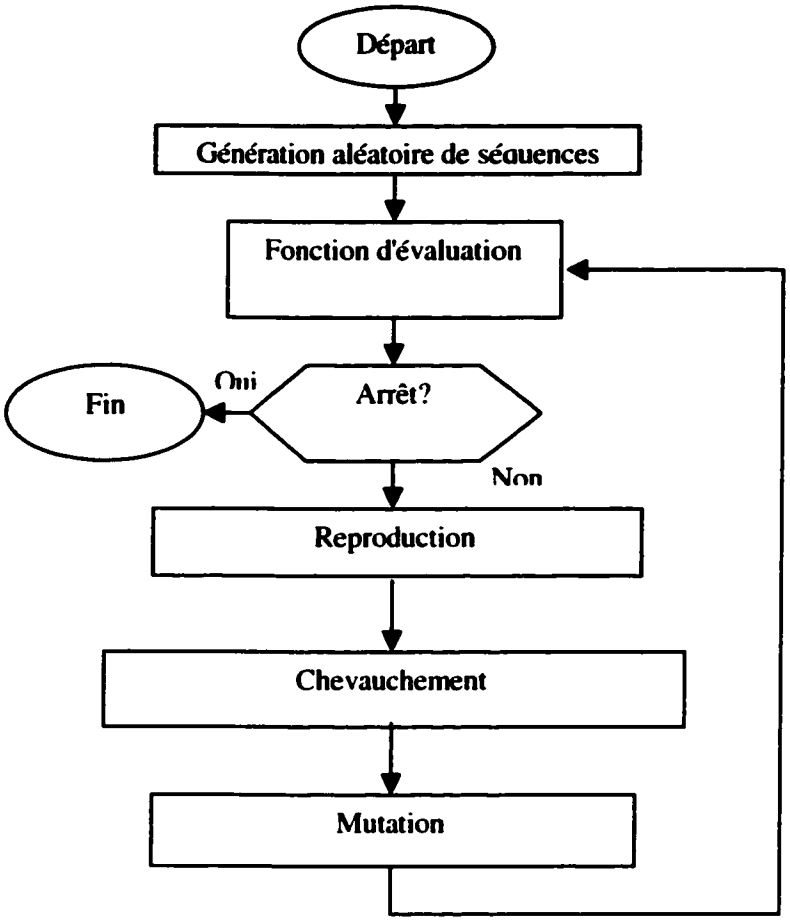


Figure 1.10 : Algorithme génétique pour l'évaluation des séquences d'assemblage

Variantes

Senin et al. (2000) ont essayé plusieurs méthodes. Ils ont entre autres :

- Remplacé complètement à chaque génération l'ancienne génération;
- Introduit des restrictions basées sur la ressemblance entre les individus.

Fujimoto et Sebaaly (2000) ont introduit le concept de blocs. L'algorithme vérifie si les meilleurs individus d'une génération peuvent être composés de sous-assemblages avec un opérateur de blocs ("block operator").

De Lit et al. (2001) comparent les individus en utilisant une méthode d'aide à la décision multi – critères appelée Prométhée II. L'utilisation de cette méthode évite de fusionner les nombreux critères techniques d'évaluation en une seule fonction d'évaluation, comme c'est généralement le cas.

Enfin, Dini et al. (1999) ont réussi à aussi générer les séquences d'assemblage avec un algorithme génétique. Le codage des gènes n'est pas exactement le même que dans les exemples précédents. Les auteurs ont ajouté :

- Des gènes correspondant aux directions d'assemblage,
- Des gènes correspondant aux outils utilisés pour tenir les composants instables.

Ces gènes ont permis d'évaluer les séquences en même temps qu'elles étaient générées. L'échantillon initial était constitué de suite d'opérations d'assemblage qui se sont ordonnées au fil des itérations pour constituer des séquences d'assemblage faisables.

1-5 Conclusion

Il existe un très grand nombre d'approches pour générer et évaluer les séquences d'assemblage. Ces approches sont plus complémentaires que concurrentes car elles ont toutes leurs avantages et leurs limites.

Pour faire avancer la recherche dans ce domaine, il nous a semblé intéressant de créer un programme pour générer les séquences d'assemblage directement à partir du dessin d'assemblage dans un logiciel de CAO répandu dans l'industrie.

Pour générer les séquences d'assemblage, nous avons choisi l'approche qui utilise les matrices de contact et de translation pour les raisons suivantes :

- L'algorithme de Ghosh et Gottipolu permet d'obtenir toutes les séquences d'assemblage géométriquement possibles ;
- Il est très facile à programmer, sans nécessiter de bonnes connaissances en Génie Informatique (comme pour les systèmes experts) ;
- Il permet d'identifier très facilement les sous-assemblages intermédiaires qui mènent au produit final (voir explications au chapitre 3) ;
- Il permet de générer les séquences d'assemblage à partir de la géométrie des solides directement (l'utilisation des connecteurs oblige l'utilisateur à les identifier manuellement et le NDBG se borne aux composants polyhédriques).

La principale limite de l'algorithme est qu'il nécessite de longs temps de calculs lorsque le nombre de composants augmente. Mais la puissance de calcul des ordinateurs étant toujours plus importante, il me semble que cette faiblesse est négligeable par rapport à ses forces.

Pour évaluer les séquences trouvées, nous avons préféré l'approche « classique ». Nous avons évalué toutes les séquences trouvées et nous laissons le soin à l'utilisateur de choisir la séquence finale. Aussi plus longue en temps de calculs, cette approche permet cependant d'obtenir à coup sûr la meilleure séquence possible.

CHAPITRE 2 – ACQUISITION DES DONNÉES GÉOMÉTRIQUES DE L'ASSEMBLAGE

2-1 Introduction

Comme nous l'avons déjà dit plus haut, il nous a semblé utile d'obtenir les données géométriques de l'assemblage de façon entièrement automatique pour générer les séquences d'assemblage. Nous allons expliquer dans ce chapitre comment nous avons fait.

En même temps que Polyassemblage fonctionne, le dessin d'origine de l'assemblage doit être ouvert avec une version de AutoCAD supérieure à 14 ou CATIA V5, sous Windows. Polyassemblage est capable d'entrer en communication avec le logiciel de CAO et de lui faire faire à peu près tout ce qu'il veut :

- Obtenir des informations sur les propriétés des éléments géométriques (volume, dimensions...),
- Rendre des objets visibles ou invisibles et les déplacer,
- Faire des tests de collision pour la recherche de chemins de désassemblage.

Ces fonctions sont nécessaires pour générer les séquences d'assemblage, et elles sont déjà programmées dans les logiciels de CAO. Il nous a semblé important d'essayer de les utiliser plutôt que de développer des nouveaux outils indépendants.

Tout d'abord, nous allons présenter le fonctionnement « classique » des logiciels de CAO, et expliquer pourquoi nous avons besoin d'automatiser certaines procédures. Ensuite nous expliquerons les principales possibilités qui existent pour automatiser des procédures avec AutoCAD et CATIA. Nous pourrons alors comparer ces possibilités et sélectionner la meilleure.

2-2 Automatisation

2-2.1 Nécessité d'automatisation

Fonctionnement classique d'une application

Un logiciel est un programme qui reçoit des informations et des ordres de l'utilisateur, via le clavier et la souris. Il réagit en fonction de ces données et retourne un résultat à l'écran (ou l'imprimante) comme indiqué sur la figure 2.1 :

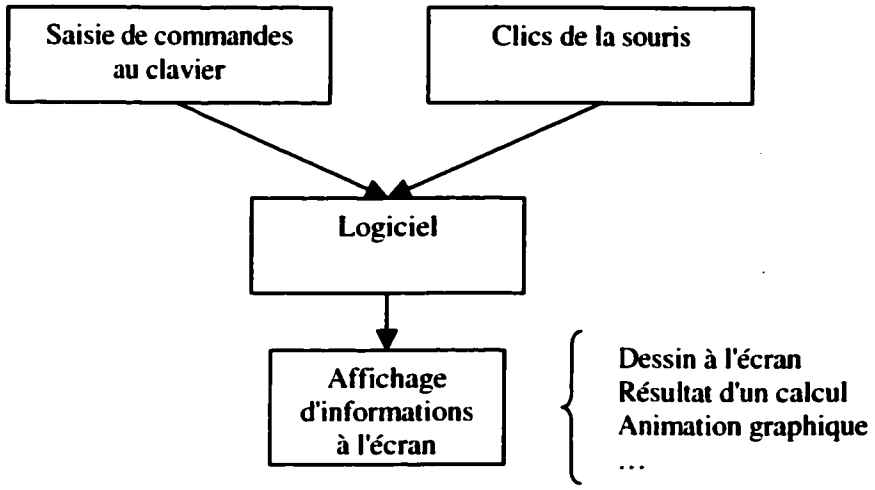


Figure 2.1 : Utilisation courante des logiciels de CAO

2-2.2 Nécessité d'automatisation

Il est extrêmement utile de pouvoir automatiser la saisie de commandes et la lecture des résultats. Pour l'exemple aussi simple que celui présenté sur la figure 2.2, le nombre de tests de collision nécessaires à Polyassemblage pour générer les séquences d'assemblage se compte en centaines, voire milliers... Impossible de tous les faire à la main !

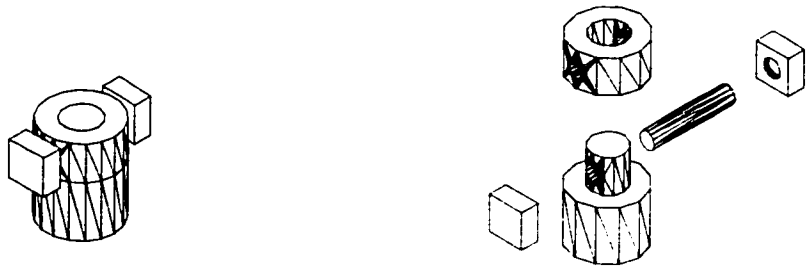


Figure 2.2 : Exemple simple d'assemblage analysé par Polyassemblage

Il existe plusieurs types d'approches pour automatiser des procédures avec Catia et AutoCAD. Nous allons voir dans la partie suivante quels ont été les critères qui ont été pris en compte pour développer Polyassemblage.

2-2.2 Critères de sélection

Contraintes nécessaires

Quelle que soit la solution retenue, elle devait absolument répondre aux deux contraintes suivantes :

- Capacité à envoyer une grande série d'ordres aux logiciels de CAO (en particulier des ordres pour déplacer des solides dans l'espace) ;
- Capacité à lire et comprendre les indications émises par les logiciels de CAO.

Critères pour choisir la meilleure approche

A partir du moment où les deux contraintes ci-dessus sont respectées, il est possible de développer un logiciel tel que Polyassemblage. Mais il existe de nombreux critères de sélection qui permettent de déterminer, à mes yeux, la meilleure approche.

Sur le plan technique, idéalement la solution retenue doit :

- Être compatible avec AutoCAD et Catia. Sinon, il faudra créer deux versions différentes de Polyassemblage, l'une adaptée à AutoCAD et l'autre à Catia ;
- Permettre d'obtenir des renseignements complémentaires pour améliorer les performances du programme (comme la taille des solides, leur nom, etc.) ;

Pour faciliter le travail des utilisateurs, il faut de plus :

- Que le résultat soit facilement compréhensibles par l'utilisateur (représentation graphique disponible, logiciel facile d'utilisation, etc.) ;
- Que l'utilisateur puisse utiliser Polyassemblage sans connaître nécessairement bien les logiciels de CAO et le langage de programmation utilisé.

Enfin, pour que le logiciel soit facile à développer et à améliorer par la suite, nous préférons utiliser un langage de programmation facile à utiliser, bien adapté à la conception logicielle et bien adapté à la « communication » avec les logiciels de CAO.

2-3 Revue des outils disponibles

2-3.1 Automatisation d'AutoCAD

Les macros

Les macros sont des outils simples qui existent avec la plupart des logiciels (de CAO ou non). Elles permettent de répéter une suite d'opérations de manière automatique. Le problème est que la suite des opérations est toujours la même. La macro ne peut pas s'adapter aux cas particuliers comme peut le faire un langage de programmation et ses possibilités sont très limitées.

AutoLISP

AutoLISP est un langage de programmation qui dérive de LISP. Il a été mis au point spécialement pour AutoCAD et existe depuis la version 10. Avec l'arrivée d'AutoCAD 2000, AutoLISP est devenu Visual LISP mais le langage lui-même est resté à peu près inchangé. La principale différence est que l'environnement de programmation a été rendu plus visuel.

AutoLISP, comme LISP est un langage basé sur les listes. Il est très bien adapté à AutoCAD. Par exemple un point peut être stocké dans une unique variable constituée de la liste des 3 coordonnées (X, Y, Z). Ci-dessous se trouvent deux exemples de code AutoLISP.

Création du point *pt* de coordonnées (X, Y, Z) :

```
(setq pt (list X Y Z))
```

La ligne suivante appelle la commande *move* de AutoCAD pour déplacer *element* selon le vecteur défini par l'origine et le point *pt* :

```
(command "move" element "" "0,0,0" pt)
```

Les routines écrites en AutoLISP sont appelées depuis AutoCAD, au même titre que n'importe quelle autre fonction. D'ailleurs parmi les fonctions existantes plusieurs sont programmées dans ce langage.

AutoLISP est très utile pour personnaliser des fonctions d'AutoCAD, ou en créer de nouvelles. Il aurait été très bien adapté pour toute la partie acquisition de données de Polyassemblage. Mais il est très peu pratique à utiliser dès qu'on souhaite développer un programme complexe comme celui pour générer des séquences d'assemblage. Par exemple la boucle *for...next* n'est même pas définie en AutoLISP !

ActiveX

Depuis la version 14, il existe dans AutoCAD une interface nommée ActiveX (la technologie ActiveX est plus ancienne et portait auparavant le nom de OLE). Cette interface

fournit un mécanisme pour permettre la manipulation d'objets AutoCAD par des programmes appartenant au « monde extérieur ». Elle met ces objets à la disposition d'autres applications, comme l'indique la figure ci-dessous extraite du menu d'aide d'AutoCAD.

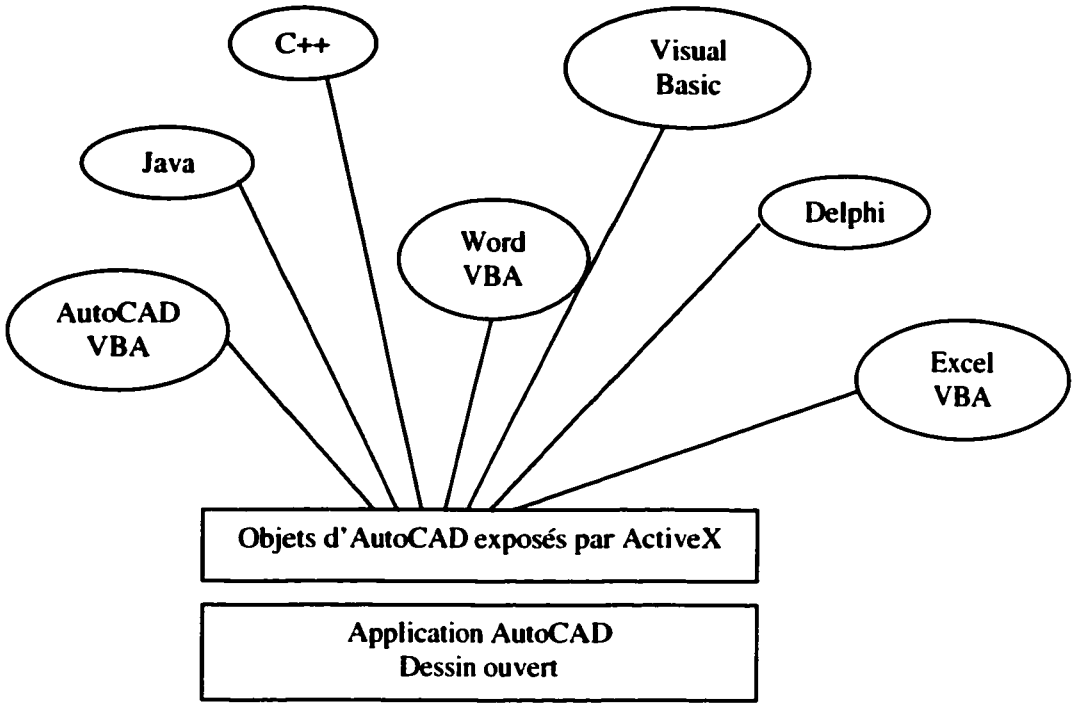


Figure 2.3 : Applications compatibles avec la technologie ActiveX

Les objets sont la base de toute application qui utilise ActiveX. Chaque objet exposé représente une partie d'AutoCAD. Il y a de nombreux types d'objets dans l'interface ActiveX d'AutoCAD :

- Les objets graphiques tels que les lignes, les arcs, les solides et les côtes;
- Les paramètres de style comme les types de ligne ou les styles des textes;
- Les structures graphiques tels que les calques, les groupes et les blocs;
- Les types d'affichage du dessin comme les différentes vues;
- Le fichier ouvert et l'application AutoCAD sont aussi considérés comme des objets.

La liste complète de ces types d'objets et leur hiérarchie pour AutoCAD se trouvent dans l'annexe 1.

2-3.2 Automatisation de CATIA

CATIA ne possède pas d'interface ActiveX comme AutoCAD, mais propose aussi un portail d'automatisation. Comme indiqué dans le menu d'aide de CATIA, il est donc aussi possible de programmer des procédures d'automatisation entre autres :

- En Basic Script lorsque CATIA est installé sur une station UNIX
- En Visual Basic lorsque CATIA est installé sous Windows.

De même que pour AutoCAD (vois ci-après), la fonction qui capture l'application CATIA sous Windows est la suivante :

```
Set CATIA = GetObject( , "CATIA.Application")
Set document = CATIA.ActiveDocument
Set produit = document.Product
Set dessin_assemblage = produit.Products
Set solide = dessin_assemblage.Item(i)
```

L'objet *document* représente le dessin ouvert. L'objet *produit* représente le produit racine du dessin, et l'objet nommé *dessin_assemblage* représente l'ensemble des composants qui constituent ce produit. L'objet *solide* tel que défini ci-dessus représente l'élément qui se trouve à la $i^{ème}$ position dans l'arbre qui représente la structure de *produit*.

Comme AutoCAD, CATIA expose donc un certain nombre d'objets au monde extérieur à travers son portail d'automatisation. Ces objets permettent de retrouver la plupart des fonctions de CATIA, mais pas toutes. Il existe par exemple une fonction qui permet d'envelopper les solides dans une « *Bounding Box* » (cette notion est décrite plus précisément dans le chapitre 4). Mais à notre connaissance, il n'est pas possible de l'automatiser.

Dans l'annexe 1 se trouve aussi une liste générale des types d'objets exposés par CATIA au « monde extérieur ».

2-4 Comparaison et sélection

2-4.1 Sélection pour AutoCAD

Comme indiqué plus haut, les macros et le langage AutoLISP ne sont pas bien adaptés au développement de Polyassemblage. En particulier, AutoLISP ne peut pas s'intégrer à un langage de programmation plus général. Nous ne pourrions donc pas développer d'application vraiment efficace et pratique à utiliser. Nous allons donc discuter des différents langages de programmation qui permettent d'utiliser l'interface ActiveX d'AutoCAD.

Word VBA, Excel VBA

Ces deux langages sont intégrés à Word et Excel et sont très pratiques à utiliser lorsqu'on cherche à faire interagir de manière automatique ces deux logiciels avec AutoCAD. Ce n'est pas notre cas, donc nous n'allons pas développer plus en détails cette approche.

VBA (Visual Basic for Application) pour AutoCAD

Microsoft VBA est un environnement de programmation qui fournit de riches possibilités pour développer de nouvelles applications (similaires à celles de Visual Basic). La principale différence réside dans le fait que VBA « roule » dans le même espace qu'AutoCAD, ce qui fournit un environnement de programmation très rapide d'exécution lorsque VBA et AutoCAD doivent interagir. VBA permet aussi une très bonne intégration avec d'autres applications, comme Word ou Excel, au point qu'AutoCAD pourrait même contrôler ces applications !

VBA aurait été un excellent langage de programmation pour ce projet. Cependant deux aspects de VBA sont gênants pour nous :

- VBA est basé sur certaines applications Windows, et CATIA n'en fait pas partie. Nous n'aurions donc pas pu écrire un programme en VBA exécutable depuis CATIA.
- VBA est développé à l'intérieur d'AutoCAD et est lancé depuis AutoCAD. Il nous semble important d'avoir un programme indépendant des logiciels de CAO, en particulier pour les phases de génération et d'évaluation des séquences qui ne les font à peu près plus intervenir.

Delphi et Java

Ce deux langages de programmation sont relativement peu connus par la plupart des étudiants de Génie Industriel, dont moi-même. Il me semblait important de développer Polyassemblage dans un langage de programmation répandu. Ainsi, le code développé sera facile à comprendre et à améliorer pour les futurs étudiants qui participeront aux développements futurs du logiciel.

C++

C++ aurait pu être un excellent langage de programmation pour Polyassemblage. Il est très performant, très répandu et très bien connu de tous les programmeurs. Mais il a, dans notre cas, deux gros défauts :

- Il est beaucoup plus difficile de développer un affichage graphique des résultats en C++ qui soit accueillant, que avec un autre langage de programmation tel que Visual Basic. Avec ce dernier, par exemple, l'affichage des fenêtres et des barres de menus se fait entièrement automatiquement et le concepteur du logiciel peut de consacrer aux problèmes liés à sa spécialité : la génération des gammes d'assemblage.
- Il est aussi beaucoup plus difficile d'utiliser l'interface ActiveX avec le C++. Le langage de base de ActiveX est VBA. Ce langage est très proche de Visual Basic, et la transposition de VBA en Visual Basic est très simple, contrairement à la transposition de VBA en C++.

Visual Basic

Comme vu sur la figure 2.3, les objets d'AutoCAD sont aussi disponibles pour les programmes écrits en Visual Basic. Il est donc possible de créer un programme en Visual Basic parfaitement autonome qui, lorsqu'il en a besoin, va pouvoir agir sur les objets exposés par AutoCAD.

Comme indiqué dans le menu d'aide d'AutoCAD, la fonction en Visual Basic qui « capture » l'application AutoCAD est :

```
Set Acad = GetObject( , "AutoCAD.Application")
```

L'objet *Acad* défini ci-dessus représente l'application AutoCAD au complet. À travers lui, le programme accède à tous les objets du dessin :

```
Set document = acad.ActiveDocument
Set dessin_assemblage = document.ModelSpace
Set element = dessin_assemblage(0)
```

L'objet *document* représente le document ouvert; *dessin_assemblage* représente l'ensemble des éléments du dessin. Enfin *element* représentera le premier élément du dessin (une ligne, une surface ou un solide).

Avec ce système, le nouveau mode d'utilisation d'AutoCAD est celui indiqué sur la figure ci-dessous.

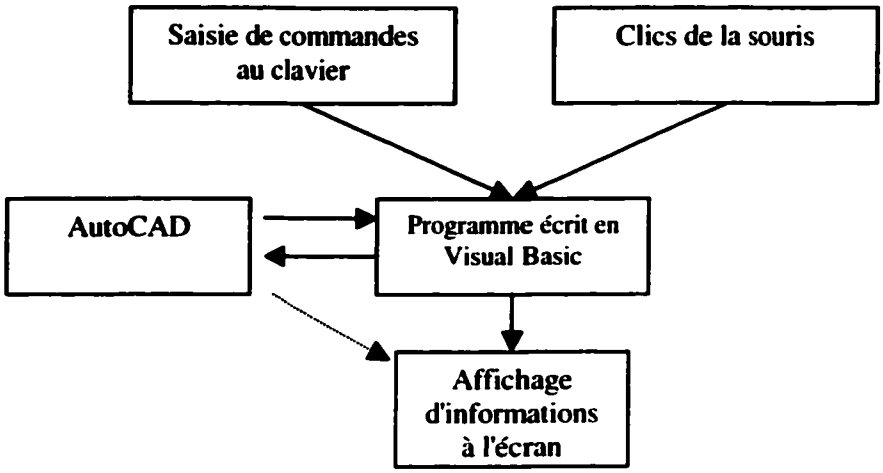


Figure 2.4 : Fonctionnement d'AutoCAD avec ActiveX

AutoCAD peut ici aussi afficher des informations à l'écran (comme dans la figure 2.1), mais il ne le fera que si le programme en Visual Basic le lui demande.

Cette solution est moins efficace du point de vue temps de calculs qu'un programme écrit en VBA. Mais nous avons choisi de développer Polyassemblage en Visual Basic car ses autres avantages pour nous sont plus importants.

2-4.2 Sélection pour Catia

Vu qu'on souhaite de préférence avoir le même programme pour Catia et pour AutoCAD, et que l'interface d'automatisation de Catia sous Windows ne fonctionne qu'avec Visual Basic, nous n'avons pas d'autre choix que d'utiliser ce langage pour la programmation de Polyassemblage. C'est d'ailleurs une solution très avantageuse pour nous.

2-4.3 Les objets avec la programmation en Visual Basic

Cette partie explique de manière détaillée comment sont structurés les objets en utilisant le langage Visual Basic et comment ils doivent être utilisés. Elle met aussi en évidence des différences importantes qu'il y a parfois entre CATIA et AutoCAD dans ce domaine. Il m'a

semblé important de l'intégrer dans ce chapitre car ce domaines de l'informatique n'est pas toujours bien connu des non-informaticiens.

Propriétés et méthodes

Chaque objet est associé à des propriétés et des méthodes. Les propriétés décrivent les attributs des objets considérés, et les méthodes sont des actions qui peuvent être effectuées sur ces objets. Dès qu'un objet est créé ou identifié, il est donc possible de le « consulter » ou le modifier selon ses besoins.

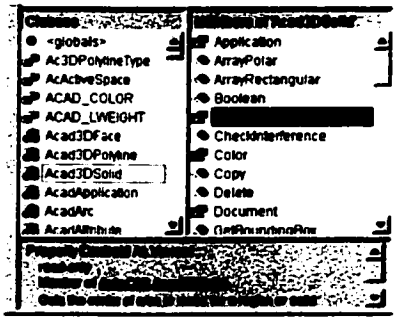


Figure 2.5 : Liste des méthodes et des propriétés des objets de la classe *3DSolid* de AutoCAD

Certaines propriétés sont accessibles seulement en lecture (volume d'un solide par exemple), et d'autres peuvent être modifiées (couleur de trait, style d'affichage, etc.). Les méthodes sont constituées de deux groupes : les fonctions *(functions)* et les sous-routines *(subs)*. Les premières renvoient un résultat (nombre, booléen, solide, etc.) mais pas les secondes.

Dans AutoCAD, voici quelques propriétés et méthodes pour un élément nommé *solide* de la classe *3DSolid* :

- Parmi les propriétés :
  - La couleur : **solide.Color = 5** change la couleur de l'objet *solide* à la couleur numéro 5;
  - La visibilité : **solide.Visible = True** rend *solide* visible;
  - Le volume : **solide.Volume** retourne le volume de *solide*.
- Parmi les méthodes :

- **solide.Move (pt_départ, pt_arrivée)** est la *sub* qui déplace *solide*.
- **solide.CheckInterference(autre_solide, True)** est la fonction qui retourne le solide qui représente l'intersection de *solide* et *autre_solide*.

Il est aussi possible d'agir à un niveau plus général. Dans CATIA, voici quelques propriétés et méthodes pour l'objet *document* défini plus haut :

- Parmi les propriétés :
  - **Document.FullName** indique le nom du dessin ouvert;
  - **Document.Saved** indique si le dessin a été sauvegardé.
- Et un exemple de méthode :
  - **Document.Save** effectue la sauvegarde du dessin.

La frontière entre propriétés et méthodes est parfois floue car une propriété pourrait aussi bien être programmée comme une méthode et vice versa. De même certaines sous-routines pourraient être programmées comme des fonctions et inversement. Le paragraphe suivant illustre ce phénomène en comparant des structures d'objets différentes qui permettent d'obtenir le même résultat avec CATIA et AutoCAD.

Comparaisons entre CATIA et AutoCAD

AutoCAD est un logiciel de CAO simple à utiliser et il n'est pas dédié à la modélisation solide en particulier. Les objets exposés sont donc plus basiques et assez simples à utiliser. Par contre CATIA est un logiciel de CAO dédié à la modélisation solide. Il est très puissant et possède un éventail de fonctions extrêmement étendu. La structure des objets exposés est nécessairement plus complexe.

Les fonctions présentées ci-dessous ont toutes été obtenues des menus d'aide de CATIA et AutoCAD.

Déplacement de solides

Avec AutoCAD la méthode est :

```
solide.Move (pt_départ, pt_arrivée)
```

Avec CATIA :

```
solide.Move.Apply vecteur_deplacement
```

Pour déplacer des éléments dans le dessin, la différence est minime. Mais pour faire un test d'interférence, la différence est plus importante.

Test d'interférence

Avec AutoCAD, comme vu précédemment la méthode est :

```
solide.CheckInterference(autre_solide, True)
```

Cette fonction crée un nouveau solide qui est l'intersection des deux. Si ce solide est vide alors il n'y a pas d'interférence entre *solide* et *autre_solide*, sinon il y en a une.

Avec CATIA, nous devons d'abord créer un nouvel objet de classe *Clash* (collision). Nous devons ensuite définir ses propriétés, et utiliser une méthode pour en faire l'analyse. La méthode pour créer l'objet *test_interference* de type *Clash* et le rajouter à la liste des objets de type *Clash* est :

```
Set I_interference = produit.GetTechnologicalObject("Clashes")
Set test_interference = liste_interferences.Add
```

Les propriétés de départ sont :

```
test_interference.FirstGroup = solide
test_interference.SecondGroup = autre_solide
test_interference.ComputationType =
    catClashComputationTypeBetweenTwo
```

Méthode qui lance l'analyse :

```
test_interference.Compute
```

Propriété qui indique la collection des conflits identifiés :

```
Set Conflits = Test_interference.Conflicts
```

Si la collection des conflits est vide, alors les solides ne se touchent pas. Si il y a des conflits et si la valeur de ces conflits est nulle, alors il y a juste contact entre les deux solides. Si elle n'est pas nulle, alors il y a une véritable interférence :

```
If Conflicts.Count > 0 then
' Présence de conflits
    If Conflicts.Item(1).Value>0 then
        ' Interférence
    else
        ' Simple contact
```

2-5 Conclusion

Plusieurs possibilités existent pour automatiser des procédures avec CATIA et AutoCAD. Seule la programmation en Visual Basic permet d'atteindre tous les objectifs fixés car elle est très performante, autant pour l'acquisition des données géométriques que pour la programmation du reste du logiciel. De plus elle est compatible avec les deux logiciels de CAO choisis pour le projet, même si l'interface à Catia ou AutoCAD est complètement différente, et elle permet d'atteindre tous les objectifs que nous nous sommes fixés.

Le principal problème est que la manière d'utiliser les différentes fonctions de Catia et AutoCAD en utilisant les objets n'a rien à voir avec l'utilisation conventionnelle de ces deux logiciels, et elle est très peu documentée. Ceci m'a causé quelques difficultés lors du développement de Polyassemblage.

Nous allons décrire dans le chapitre suivant l'algorithme qui permet de générer toutes les séquences d'assemblage possibles d'un produit. Nous saurons alors exactement quelles données sont nécessaires pour générer ces séquences et nous pourrons les calculer.

CHAPITRE 3 – GÉNÉRATION DES SÉQUENCES D'ASSEMBLAGE

3-1 Introduction

La théorie présentée dans cette partie est basée sur l'algorithme de Ghosh et Gottipolu (1995). Elle permet de générer toutes les séquences d'assemblage d'un produit connaissant les chemins d'assemblage pour toutes les paires de composants. Ces chemins doivent être rectilignes et selon les axes X, Y et Z.

En 1996 l'algorithme a été amélioré par Aguilar pour permettre de générer des séquences à partir de directions d'assemblage quelconques du plan.

Enfin, cet algorithme a été complété dans le cadre de ce projet pour :

- Permettre la génération de séquences à partir de directions d'assemblage quelconques;
- Offrir plus de détails sur les séquences trouvées. L'algorithme de Ghosh et Gottipolu ne définit les séquences d'assemblage que par l'ordre d'assemblage des composants. Nous avons rajouté la notion de direction d'assemblage.

3-2 Algorithme de Ghosh et Gottipolu

3-2.1 Données nécessaires

Les données nécessaires pour pouvoir générer les séquences sont :

- Les relations de contact entre toutes les paires de composants (stockées dans les matrices de contact),

- Les directions de désassemblage possibles pour tous les composants (stockées dans les matrices de translation).

Ces données peuvent être acquises de manière automatique avec un logiciel de CAO (tel que PADL – 2, utilisé par Ghosh et Gottipolu), manuellement (moyen utilisé par Aguilar), ou calculées directement à partir du dessin de l'assemblage dans AutoCAD ou CATIA. Cette dernière méthode est la méthode développée dans ce projet et elle est décrite dans le chapitre 4.

Nous allons illustrer le calcul de ces matrices avec l'exemple de la figure ci-dessous.

Figure 3.1 : Assemblage pour illustrer l'algorithme de génération des séquences

L'axe X est l'axe de la tige et Z est l'axe vertical. Les différents composants de l'assemblage sont :

Base Dessus Tige Bouchon_droit Bouchon_gauche

Figure 3.2 : Liste des 5 composants de l'assemblage

Les matrices de contact

Les matrices de contact permettent de savoir quels composants sont en contact avec quels autres dans l'assemblage. Il est nécessaire de connaître ces données, car lors de l'assemblage on ne peut ajouter que des composants qui ont un contact avec l'assemblage

principal. L'algorithme de Ghosh et Gottipolu prend d'ailleurs pour hypothèse que cette condition est suffisante. Nous supposons que s'il y a un contact alors ce contact permettra au composant de tenir sur l'assemblage. Cette condition n'est bien sûr pas toujours vérifiée dans la réalité, mais le programme ne fonctionne que dans ce cas là.

Pour chaque paire de composants, la valeur de contact est 1 s'il y a contact, 0 sinon. Évidemment pour deux composants $comp_i$ et $comp_j$ on a $C(comp_i, comp_j) = C(comp_j, comp_i)$.

Les matrices de contact pour l'exemple choisi sont :

C(Base, Dessus) = 1

C(Base, Tige) = 1

C(Base, Bouchon_gauche) = 0

C(Base, Bouchon_droit) = 0

C(Dessus, Bouchon_gauche) = 0

C(Dessus, Bouchon_droit) = 0

C(Dessus, Tige) = 1

C(Tige, Bouchon_droit) = 1

C(Tige, Bouchon_gauche) = 1

C(Bouchon_gauche, Bouchon_droit) = 0

Les matrices de translation

Les matrices de translation permettent de connaître les directions de désassemblage possible pour tous les composants pris deux par deux. Les directions essayées sont les 6 directions principales de l'espace : X+, X-, Y+, Y-, Z+, Z-.

$T(comp_i, comp_j) = (libre\_X+, libre\_X-, libre\_Y+, libre\_Y-, libre\_Z+, libre\_Z-)$

La valeur *libre_X+* sera 1 si $comp_j$ peut se désassembler de $comp_i$ selon la direction X+, 0 sinon. De même pour les autres valeurs. Nous obtenons pour notre exemple :

T(Base, Dessus) = (0, 0, 0, 0, 1, 0)

T(Base, Tige) = (1, 1, 0, 0, 0, 0)

T(Base, Bouchon_gauche) = (0, 1, 1, 1, 1, 1)

T(Base, Bouchon_droit) = (1, 0, 1, 1, 1, 1)

T(Dessus, Bouchon_gauche) = (0, 1, 1, 1, 1, 1)

T(Dessus, Bouchon_droit) = (1, 0, 1, 1, 1, 1)

T(Dessus, Tige) = (1, 1, 0, 0, 0, 0)

T(Tige, Bouchon_droit) = (1, 0, 0, 0, 0, 0)

T(Tige, Bouchon_gauche) = (0, 1, 0, 0, 0, 0)

T(Bouchon_gauche, Bouchon_droit) = (1, 0, 1, 1, 1 1)

Comme avec la matrice de contact, il existe ici aussi une symétrie entre $T(comp_i, comp_j)$ et $T(comp_j, comp_i)$:

$$T(comp_i, comp_j) = (a,b,c,d,e,f) \Rightarrow T(comp_j, comp_i) = (b,a,d,c,f,e)$$

3-2.2 Génération des séquences d'assemblage

Cette section décrit un algorithme systématique qui traduit les matrices de contact et de translation en séquences d'assemblage complètes. L'idée générale est de partir de paires de composants et de leur ajouter des composants ou des sous-assemblages déjà formés au fur et à mesure, jusqu'à obtenir l'assemblage final. Si les composants sont assemblés un par un, la séquence est dite linéaire. Si on ajoute au moins une fois un sous-assemblage déjà formé de plusieurs composants, la séquence est dite non linéaire.

L'algorithme se divise en deux procédures, qui ont été décrites par Ghosh et Gottipolu (1995) selon cette structure :

Procédure 1 : génération de toutes les paires de composants.

Cette étape est relativement simple. Pour chaque composant $comp_i$ et $comp_j$, si $C(comp_i, comp_j)=1$ alors on est sûr qu'on peut créer le sous-assemblage $(comp_i, comp_j)$.

Avec notre exemple nous obtenons à la fin de cette première procédure les paires *(Base, Dessus), (Base, Tige), (Dessus, Tige), (Tige, Bouchon_droit)* et *(Tige,Bouchon_gauche)*.

Procédure 2 : génération des sous-assemblages de niveau supérieur.

Dans cette procédure on cherche à ajouter un autre composant ou un sous-assemblage déjà formé à tous les sous-assemblages existants :

Étape 1 : pour pouvoir ajouter un nouveau composant ou un sous-assemblage, il doit avoir au moins un contact avec le sous-assemblage de départ. Voici l'algorithme qui fait ce test de contact :

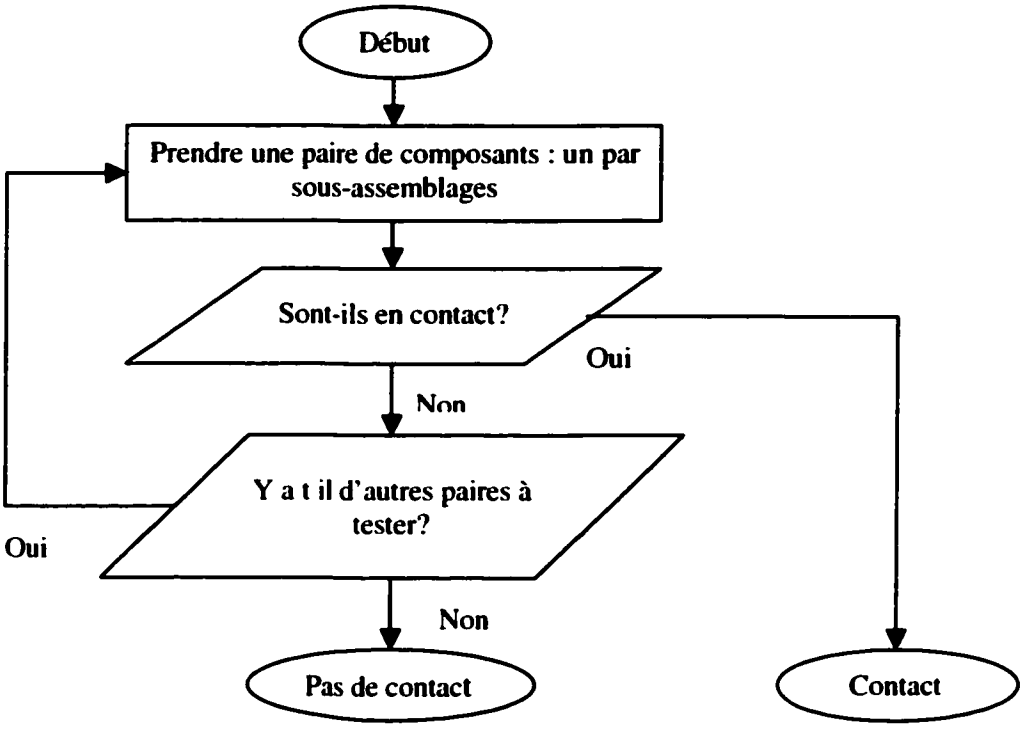


Figure 3.3 : Algorithme pour déterminer s'il y a un contact entre deux groupes de composants

La présence d'un contact est une condition nécessaire, mais pas suffisante. Par exemple si nous considérons la paire *(Base, Tige)*, nous ne pourrons pas lui assembler le composant *Dessus*. L'étape 2 fait le test pour savoir si l'assemblage est finalement possible :

Étape 2 : l'assemblage est possible s'il existe une direction d'assemblage libre pour les deux groupes de composants à assembler. Il faut donc qu'il existe une direction telle que pour

chaque composant du premier assemblage, et pour chaque composant du second, leur fonction de translation ait la valeur 1 pour la direction spécifiée.

L'algorithme qui fait ce test est le suivant :

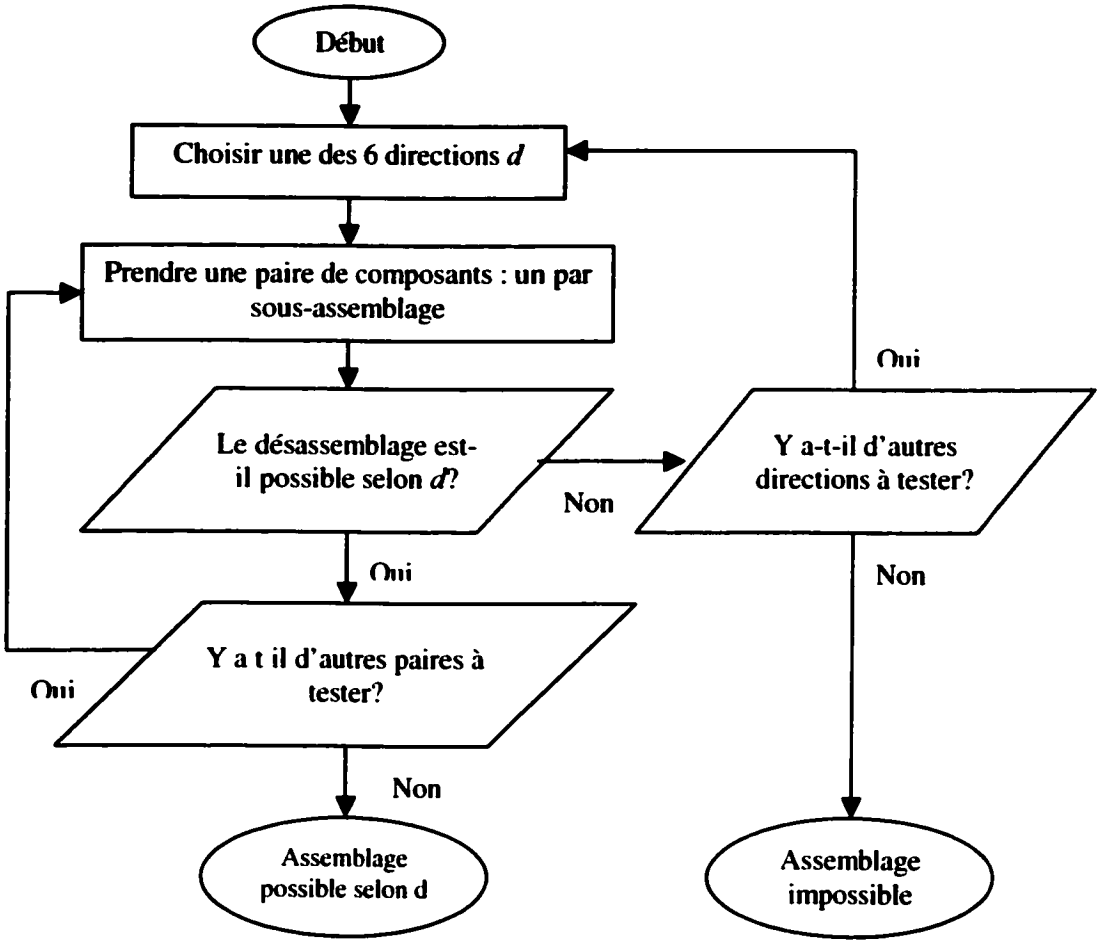


Figure 3.4 : Algorithme pour déterminer s'il est possible d'assembler deux groupes de composants

La procédure 2 se poursuit tant que toutes les combinaisons possibles n'ont pas été essayées. L'algorithme général du programme est indiqué sur la figure ci-dessous :

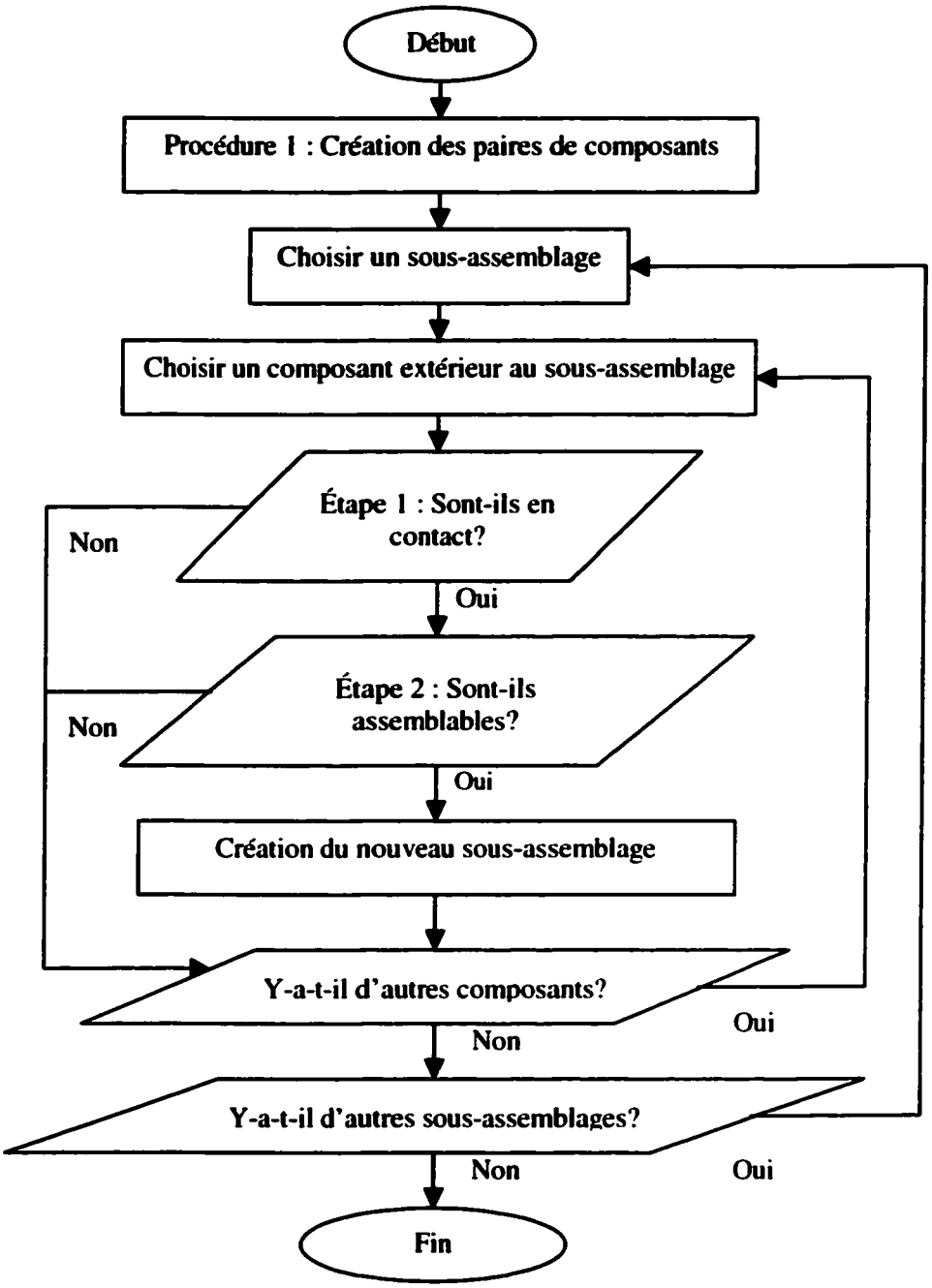
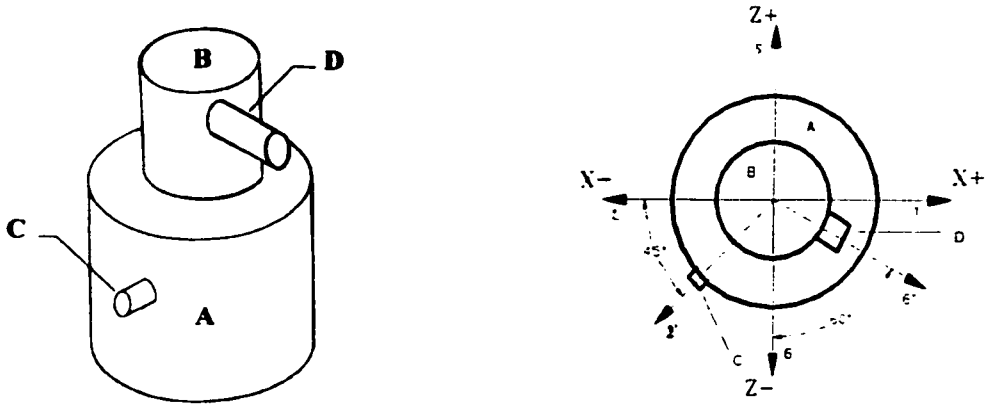


Figure 3.5 : Algorithme général de l'algorithme de Ghosh et Gottipolu

Nous ne gardons en mémoire finalement que les sous-assemblages complets, et leur séquence constitue une séquence d'assemblage possible pour le produit.

3-3 Améliorations proposées par Aguilar

Aguilar a élargi le champ d'action de l'algorithme précédent. Il a programmé la génération des séquences d'assemblage avec des directions « cylindriques », c'est à dire des directions quelconques du plan XY, YZ ou XZ.



Vue 3D

Vue de dessus

Figure 3.6 : Exemple extrait de Aguilar (1996)

Aguilar a proposé de faire tourner le repère principal autour de l'axe X, Y ou Z pour que les composants soient toujours désassemblés selon un axe principal. Cette opération permet de se ramener toujours au cas développé par Ghosh et Gottipolu. Dans l'exemple de la figure 3.6 le repère va devoir tourner autour de Y, pour que C et D puissent être désassemblés selon les directions X- et Z-.

Tableau 3.1 : Fonctions de contact et de translation avec des composants cartésiens et non cartésiens

Paire	Angle de rotation	Fonction contact	Fonction translation
AB	--	1	111110
A(C)	45	1	010000
A(D)	60	0	111011
B(C)	45	0	111111
B(D)	60	1	000001
(C)(D)	60	0	111111
BA	--	1	111101
(C)A	45	1	100000
(D)A	--	0	110111
(C)B	--	0	111111
(D)B	60	1	000010
(D)(C)	45	0	111111

Dans le tableau[1] ci-dessus, les composants « non cartésiens » (ceux dont la direction d'insertion n'est pas selon un axe principal) sont présentés entre parenthèses. Le composant qui est à droite de chaque paire est toujours celui qui bouge.

Comme l'indique Aguilar, la fonction de translation d'une paire de composants *(comp$_i$, comp$_j$)* n'est plus forcément symétrique avec celle de la paire *(comp$_j$, comp$_i$)*. L'algorithme de génération des séquences d'assemblage est aussi légèrement plus complexe, car la programmation n'est pas la même selon qu'on assemble des composants « cartésiens » (comme A ou B) avec des composants « non cartésiens » (comme C ou D) et inversement.

1. L'ordre des chiffres pour les fonctions de translation diffère ici de ceux indiqués dans le mémoire de Aguilar. La raison est qu'à l'origine Ghosh et Gottipolu ont défini la fonction de translation par *T(comp$_i$, comp$_j$) = (libre_X+, libre_Y+, libre_Z+, libre_X-, libre_Y-, libre_Z-)*. Pour simplifier la programmation de la recherche de directions d'assemblage, nous avons été amenés à modifier l'ordre des paramètres.

3-4 Autres modifications

Dans cette partie, nous proposons plusieurs modifications de l'algorithme décrit ci-dessus pour le rendre mieux adapté à la programmation.

3-4.1 Simplification de l'algorithme

Au début, l'algorithme de Ghosh et Gottipolu génère toutes les paires de solides qu'il est possible d'assembler deux à deux. C'est la procédure 1. Ensuite, au cours de la procédure 2, il construit les sous-assemblages de taille supérieure en combinant ces paires avec d'autres composants. Nous n'avons pas programmé la procédure 1. La procédure 2 construit les sous-assemblages de taille supérieure en combinant directement les éléments pris individuellement.

3-4.2 Génération des séquences avec des directions d'assemblage quelconques

Dans notre projet, le programme n'a pas besoin de faire de rotation du repère autour d'un axe principal pour se ramener à l'algorithme de Ghosh et Gottipolu. Au lieu de ne travailler qu'avec les directions X, Y et Z (comme précédemment), le programme va être capable de travailler avec n'importe quelle direction, du moment qu'elle permet d'assembler des composants.

L'algorithme de génération des séquences reste inchangé. Seule la liste des directions à essayer est modifiée dans l'algorithme de la figure 3.4. Cette méthode est illustrée ci-dessous :

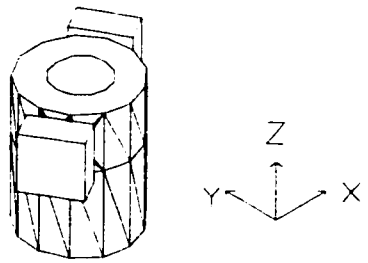


Figure 3.7 : Assemblage de la figure 3.1 réorienté dans une direction quelconque

L'ensemble des composants de l'assemblage de la figure 3.7 peut s'assembler selon deux directions :

- L'axe Z pour désassembler *Dessus* et *Base* (La direction Z+ pour désassembler *Dessus* de *Base*, et Z- pour désassembler *Base* de *Dessus*);
- L'axe d'insertion de la tige : cet axe forme un angle de 30° avec l'axe X dans le plan XY. Il est défini par le vecteur (cos 30, sin 30, 0), soit approximativement (0.866, 0.5, 0).

Les directions sont enregistrées sous la forme d'un vecteur. Nous expliquerons plus loin comment le programme les identifie automatiquement. L'ensemble des directions avec lesquelles il travaille est :

Tableau 3.2 : Liste des directions de désassemblage

Direction n°	X	Y	Z
1	0	0	1
2	0	0	-1
3	0.866	0.5	0
4	-0.866	-0.5	0

Les fonctions de translation ont presque la même définition que en III-2.1 :

$T(comp_i, comp_j)$ = (libre_dir1, libre_dir2, libre_dir3, libre_dir4, libre_dir5, libre_dir6...)

La valeur *libre_dir1* sera 1 si *$comp_j$* peut se désassembler de *$comp_i$* selon la direction n°1, 0 sinon. De même pour les autres valeurs. Nous obtenons pour notre exemple, où seulement 4 directions ont été utilisées :

T(Base, Dessus) = (1, 0, 0 ,0)

T(Base, Tige) = (0, 0, 1, 1)

T(Base, Bouchon_gauche) = (1, 1, 0, 1)

T(Base, Bouchon_droit) = (1, 1, 1, 0)

T(Dessus, Bouchon_gauche) = (1, 1, 0, 1)

T(Dessus, Bouchon_droit) = (1, 1, 1, 0)

T(Dessus, Tige) = (0, 0, 1, 1)

T(Tige, Bouchon_droit) = (0, 0, 1, 0)

T(Tige, Bouchon_gauche) = (0, 0, 0, 1)

T(Bouchon_gauche, Bouchon_droit) = (1, 1, 1, 0)

Le programme ne tient pas du tout compte des axes X et Y, car ils ne permettent l'assemblage d'aucun composant. La manière dont le programme identifie les directions 1, 2, 3 et 4 comme bonnes directions d'assemblage est décrite dans le chapitre 4.

Ici la symétrie entre $T(comp_i, comp_j)$ et $T(comp_j, comp_i)$ est conservée, nous n'avons pas besoin de recalculer les fonctions de translation pour les paires symétriques. Ceci représente un important gain de temps lors de l'exécution du programme, car le calcul des matrices de translation est une opération très longue (voir exemples au chapitre 7).

Le nombre de directions d'assemblage est borné. Dans le cas le plus défavorable, chaque composant amène deux nouvelles directions, et le nombre total de directions d'assemblage sera :

$$N_{directions} \leq 2 \times (n_{composants\_a\_assembler} - 1)$$

La génération des séquences se fera exactement selon le même principe qu'indiqué précédemment. Pour assembler deux groupes de composants, le programme vérifiera :

- s'il existe un contact entre les deux groupes,
- s'il existe une direction d'assemblage parmi les 4 qui permet d'assembler les deux groupes.

3-4.3 Application de critères pour éviter l'explosion du nombre de séquences

Pendant les différents tests, nous nous sommes rendus compte que le programme arrive à trouver des dizaines de milliers de séquences pour certains assemblages. Dans ces cas, les calculs ont été longs (1h pour générer 24000 séquences) et n'apportent pas grand-chose : l'utilisateur connaît souvent *à l'avance* certains des critères qu'il est sûr d'appliquer, et la plupart des séquences générées l'ont été inutilement.

Il nous a donc semblé utile d'appliquer certaines contraintes dès la génération des séquences. Aguilar en avait déjà proposé quelques-unes (forcer le composant par lequel la séquence débute, forcer la première liaison, etc.). Dans ce projet, nous avons programmé des contraintes basées sur la géométrie des composants.

Séquences linéaires et séquences non linéaires

L'utilisateur peut choisir dès le départ de ne générer que des séquences linéaires. Les avantages et les désavantages d'avoir des séquences non linéaires sont évoqués en détails dans le chapitre 5.

Si l'utilisateur choisit à ce niveau de ne générer que des séquences linéaires (c'est le choix fixé par défaut par le logiciel), alors la génération des séquences d'assemblage sera beaucoup plus rapide.

Par contre, il existe des cas pour lesquels il n'existe pas de séquence d'assemblage linéaire. L'utilisateur sera obligé de générer les séquences d'assemblage non linéaires pour trouver des bonnes séquences d'assemblage. La figure ci-dessous en montre un exemple :

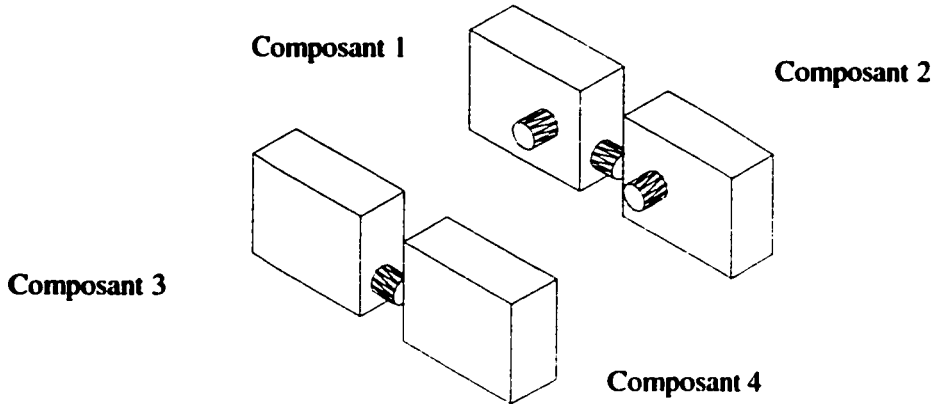


Figure 3.8 : Cas où il est nécessaire de générer les séquences non linéaires

Pour assembler ces 4 composants, nous sommes obligés d'assembler les composants 1 et 2 d'une part, et les composants 3 et 4 d'autre part avant de les assembler entre eux.

Débuter l'assemblage par l'élément le plus volumineux

Cette option n'est pas activée par défaut : l'utilisateur doit lui-même l'activer s'il le souhaite. Si elle est activée, le programme ne générera que les séquences qui commencent par l'élément le plus volumineux du dessin.

Interdire l'assemblage de composants depuis la direction Z-

Cette option n'est pas non plus activée par défaut. L'utilisateur doit lui-même l'activer s'il le souhaite. Si elle est activée, le programme ne générera que les séquences d'assemblage pour lesquelles aucun composant n'est assemblé selon la direction Z- (direction qui, en pratique, représente le bas).

Les assemblages qui se font selon cette direction sont souvent peu pratiques à réaliser car :

- les composants ainsi ajoutés sont instables tant qu'ils ne sont pas fixés définitivement;

- lorsque l'opération est manuelle, l'opérateur doit se mettre dans une position inconfortable (tête levée vers le haut, dos courbé) ou avec peu de visibilité sur ce qu'il fait.

3-4.4 Génération de séquences en tenant compte des directions d'assemblage

Avec l'algorithme de Ghosh et Gottipolu, seul l'ordre d'assemblage des composants intervient. Lorsqu'un même composant peut être assemblé dans plusieurs directions différentes, cela n'apparaît pas dans la séquence finale. Cependant la direction d'insertion des composants est importante car elle a une grande influence sur l'efficacité globale de la séquence.

Dans toute la suite du projet, nous tiendrons compte de la direction d'assemblage des composants : deux séquences pour lesquelles l'ordre d'assemblage est le même mais un composant peut s'insérer dans deux directions différentes sont deux séquences différentes. Par exemple, la séquence *(Base, Dessus, Tige, Bouchon_droit, Bouchon_gauche)* correspond en fait à deux séquences :

1) *(Base, Dessus-1, Tige-3, Bouchon_droit-3, Bouchon_gauche-4)*

2) *(Base, Dessus-1, Tige-4, Bouchon_droit-3, Bouchon_gauche-4)*

L'algorithme de la figure 3.4 est légèrement modifié. Auparavant nous avions juste besoin de savoir si nous pouvions assembler deux groupes de composants entre eux. Maintenant, nous avons besoin de connaître toutes les directions selon lesquelles les deux groupes de composants peuvent s'assembler entre eux.

Au lieu de s'arrêter dès qu'une direction d'assemblage est trouvée, le programme doit donc essayer l'ensemble des directions connues.

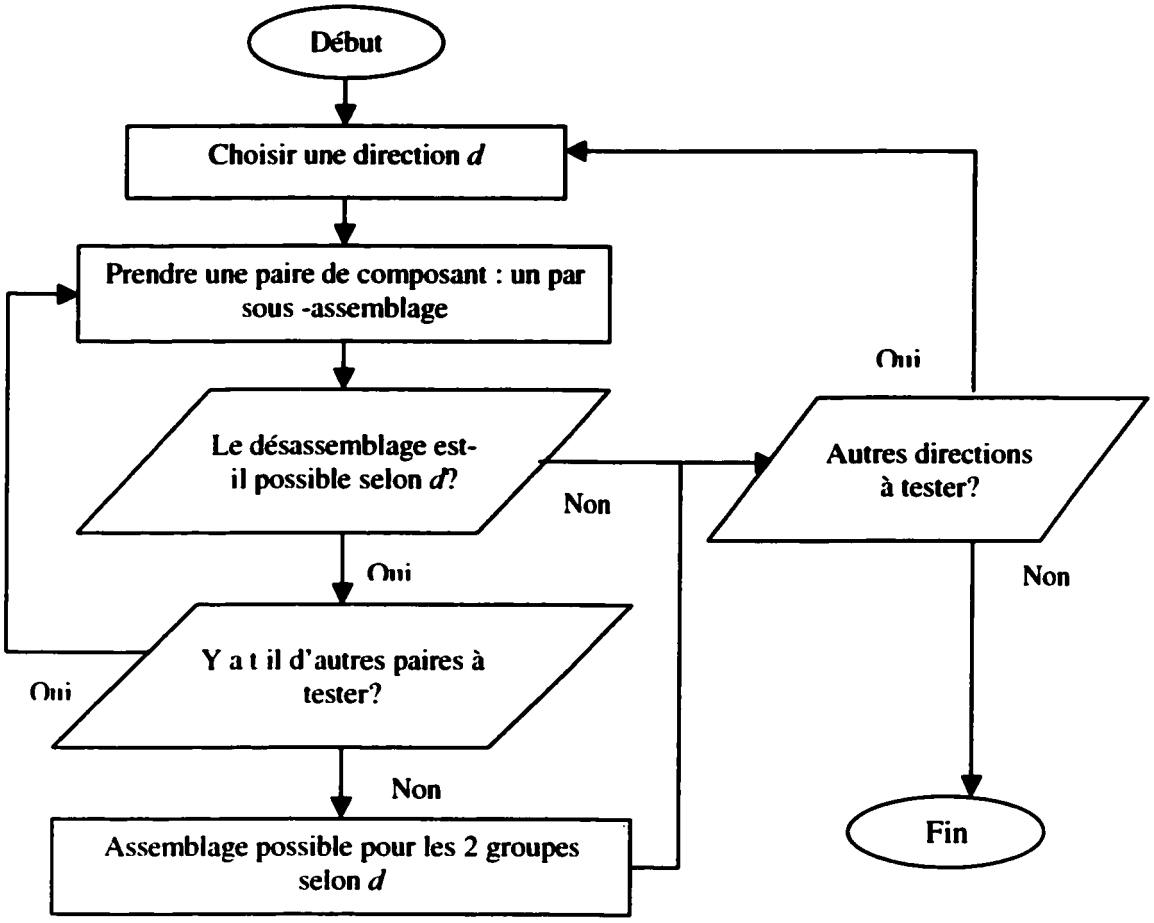


Figure 3.9 : Algorithme pour déterminer l'ensemble des directions d'assemblage possibles pour deux groupes de composants

3-5 Conclusion

Nous avons présenté dans ce chapitre un algorithme qui permet de générer toutes les séquences d'assemblage d'un produit, compte tenu d'informations très simples : les matrices de contact et de translation pour chaque paire de composants de l'assemblage, et la liste des directions d'assemblage possibles. Nous allons présenter dans le chapitre suivant comment obtenir ces données directement à partir de la géométrie de l'assemblage dans CATIA ou AutoCAD.

CHAPITRE 4 – CALCUL DES MATRICES DE CONTACT ET DE TRANSLATION ET RECHERCHE DE NOUVELLES DIRECTIONS D'ASSEMBLAGE

4-1 Introduction

Comme vu dans le chapitre 3, le programme a besoin de connaître les matrices de contact et de translation pour chaque paire de solides pour générer toutes les séquences d'assemblage du produit.

L'objectif de ce chapitre est d'obtenir ces matrices directement à partir du dessin d'assemblage. Nous allons voir comment les calculer et comment trouver des nouveaux chemins d'assemblage lorsque c'est nécessaire.

Comme indiqué dans le chapitre 2 le programme utilise l'interface ActiveX pour AutoCAD et le portail d'automatisation offert par CATIA pour interagir avec ces deux logiciels.

4-2 Calcul des matrices de translation

Ces matrices indiquent pour chaque paire de composants la liste des directions de désassemblage ou d'assemblage possibles entre eux. Il y a un certain nombre de contraintes à respecter pour pouvoir obtenir ces données :

- Les directions de désassemblage doivent être rectilignes;
- Les composants ne peuvent pas être déformés pour le désassemblage (les fils électriques, les clips et les ressorts sortent du cadre de ce projet);
- Les composants ne peuvent pas être tournés pour être désassemblés (les pas de vis sont interdits);

- Les composants ou groupes de composants sont désassemblés un par un;
- Seules les contraintes d'assemblage géométriques sont prises en compte;
- S'il est possible de désassembler une paire de composants, alors il sera possible de l'assembler de la même manière : trouver un chemin de désassemblage est équivalent à trouver un chemin d'assemblage.

Soit n le nombre de directions d'assemblage connues. Si n n'est pas nul, les matrices de translation sont stockées sous la forme :

$$T(comp_i, comp_j) = (T_1, T_2, ..., T_d, ..., T_n)$$

Avec :

T_d = *1* si $comp_j$ peut être désassemblé de $comp_i$ dans la direction n°d

0 sinon

La partie suivante indique comment obtenir la valeur de T_d pour une paire donnée $(comp_i, comp_j)$ et une direction donnée d. Chaque direction est définie par un vecteur directeur (d_x, d_y, d_z) de norme 1.

4-2.1 Calcul de T(comp$_i$. comp$_j$) pour une direction donnée

Le principe est le suivant : le programme déplace $comp_j$ selon la direction essayée, et regarde si au cours du déplacement le composant est entré en collision avec $comp_i$.

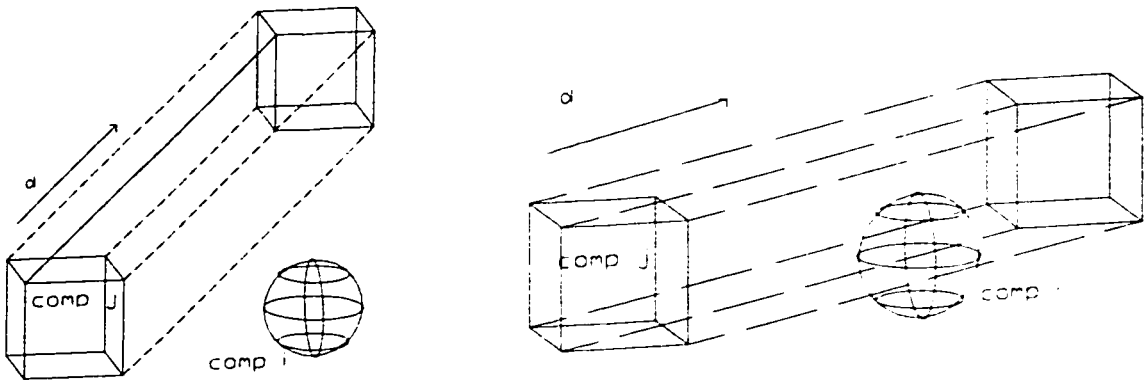
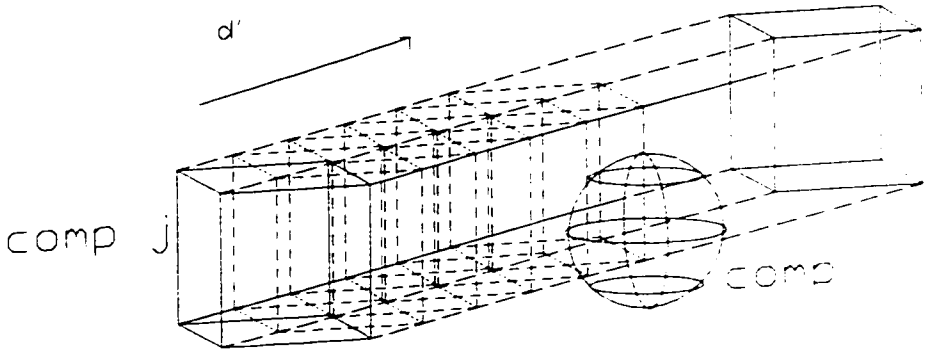


Figure 4.1 : Calcul de $T(comp_i, comp_j)$ avec deux directions d et d'

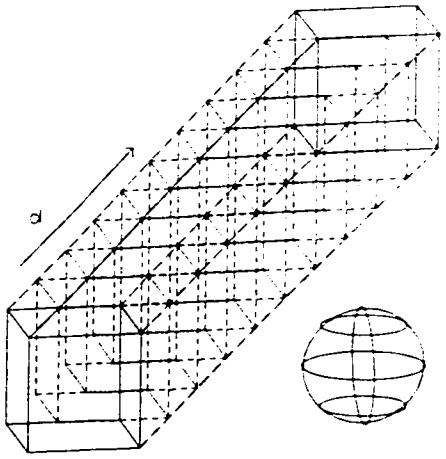
Dans le premier cas il n'y a pas de collision au cours du désassemblage. Par contre dans le second il y en a une : *d'* n'est pas une direction de désassemblage possible pour la paire *(comp$_i$, comp$_j$)*.

Fonctionnement de l'algorithme

Le programme déplace pas à pas *comp$_j$*, et s'arrête s'il se rend compte qu'après un déplacement élémentaire *comp$_i$* et *comp$_j$* sont entrés en collision :



a) Collision détectée



b) Aucune collision détectée

Figure 4.2 : Test pas à pas pour essayer les deux directions *d* et *d'*

L'algorithme qui fait le test est indiqué ci-dessous. Toutes les étapes sont expliquées en détails dans les pages suivantes.

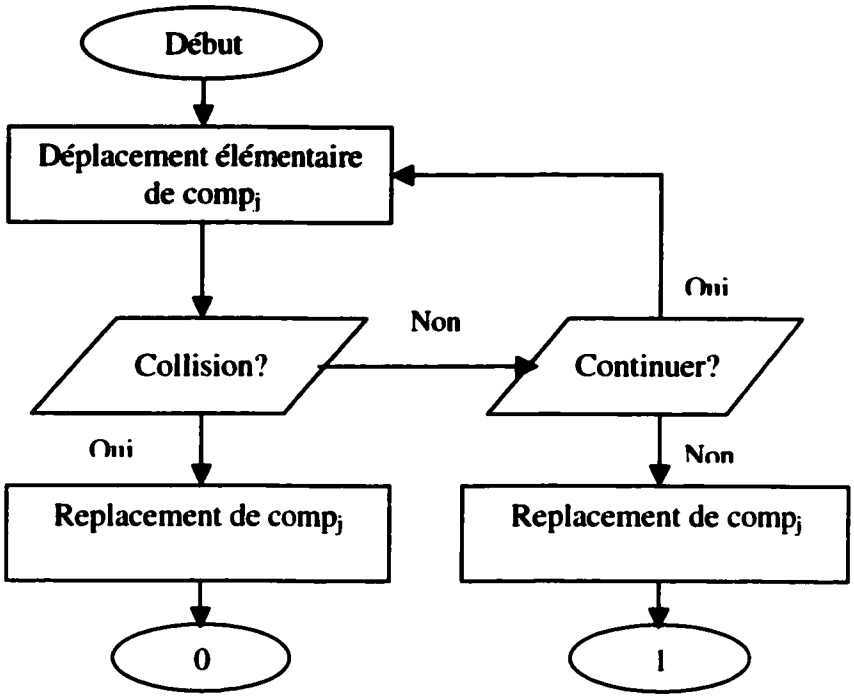


Figure 4.3 : Algorithme du test pour connaître *T(comp$_i$, comp$_j$)* selon la direction *d*

Début

Au départ, les éléments du dessin sont dans leur position initiale. Polyassemblage utilise donc directement le dessin d'origine de l'assemblage, sans modifier la position des solides au préalable.

Déplacement élémentaire de comp$_j$:

On déplace *comp$_j$* avec un pas élémentaire dans la direction *d*. Le vecteur de déplacement (V_x, V_y, V_z) est donné par :

$V_x = pas\_elementaire \cdot d_x$

$V_y = pas\_elementaire \cdot d_y$

$V_z = pas\_elementaire \cdot d_z$

Comme (d_x, d_y, d_z) est normé, le solide sera déplacé d'une distance égale exactement à *pas_elementaire*.

Plus ces déplacements sont petits et moins l'algorithme a de chances de « sauter » une collision éventuelle. Par contre, le temps de calculs est rallongé.

pas_elementaire est calculé de la façon suivante :

- Avec AutoCAD : *pas_elementaire = largeur_minimale_du_plus_petit_solide / 10;*
- Avec CATIA :*pas_elementaire = taille_du_dessin / 50* car nous n'avons pas été capables de calculer la largeur des différents solides du dessin.

L'utilisateur peut modifier manuellement ce pas s'il le juge nécessaire.

Collision?

Tout au long du projet, nous avons utilisé le plus possible les fonctionnalités de CATIA et d'AutoCAD déjà existantes. Pour savoir s'il y a collision, nous avons simplement utilisé les fonctions d'interférences proposées par ces deux logiciels. Nous n'avons jamais cherché à créer de nouveaux outils quand nous pouvions nous satisfaire des outils existants.

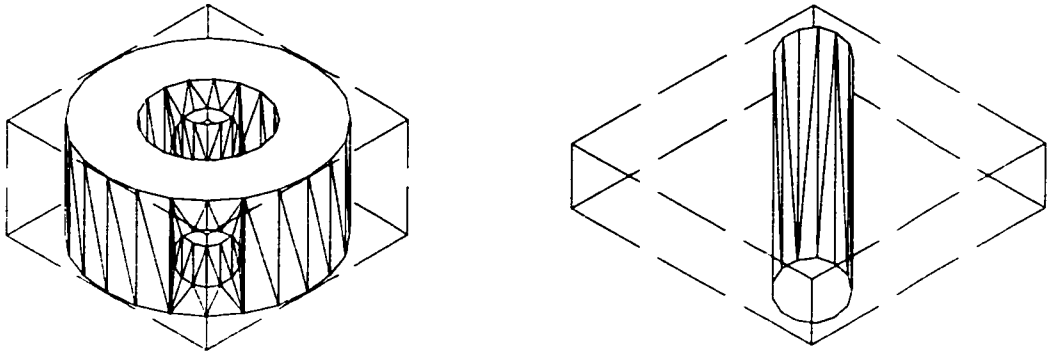
S'il y a collision, il est inutile de continuer le test. Nous sommes sûrs que la direction choisie n'est pas une direction possible pour désassembler $comp_j$ de $comp_i$.

Continuer?

Ce test indique si le programme doit continuer les déplacements. Si les deux solides sont suffisamment éloignés l'un de l'autre alors la direction essayée est une bonne direction de désassemblage.

Il est nécessaire de définir le terme « suffisamment éloigné » :

- Avec CATIA, l'utilisateur doit indiquer la dimension maximale du produit assemblé. « Suffisamment éloigné » signifie que $comp_j$ a parcouru au moins cette distance.
- Avec AutoCAD, il est possible d'envelopper chaque solide dans une boîte avec une fonction enveloppe (*GetBoundingBox*) comme indiqué sur la figure ci-dessous. Cette fonction permet de connaître les points extrêmes du solide.



Enveloppe de *Dessus* Enveloppe de la *Tige*

Figure 4.4 : Enveloppe de deux solides de l'assemblage de la figure 3.7

Dès que le point le plus à gauche de $comp_j$ est passé à droite de $comp_i$ alors nous sommes sûrs qu'il est inutile de poursuivre le test. De même dans les autres directions : le déplacement de $comp_j$ s'arrête dès qu'il passe « hors du champ » de $comp_i$.

Replacement de $comp_i$

Pour pouvoir continuer les tests avec les autres directions et les autres paires de solides, le programme doit repositionner $comp_j$ dans sa position initiale.

Performances de la méthode

Temps de calculs

Cette méthode nécessite un grand nombre de tests de collision. Il faut en faire plusieurs dizaines pour essayer entièrement chaque direction. Ces tests sont des opérations assez longues à l'échelle des autres opérations (de 1 à 1.5 centièmes de seconde par test pour AutoCAD et de 1 à 4 pour CATIA), donc ils sont responsables de quasiment tout le temps de calcul passé dans cette première partie.

Fiabilité des résultats

Prendre un pas trop large peut mener à ne pas voir certaines collisions. Ceci se produit surtout lorsque l'assemblage contient des pièces très fines comme des plaques. On peut voir ce phénomène sur la figure ci-dessous :

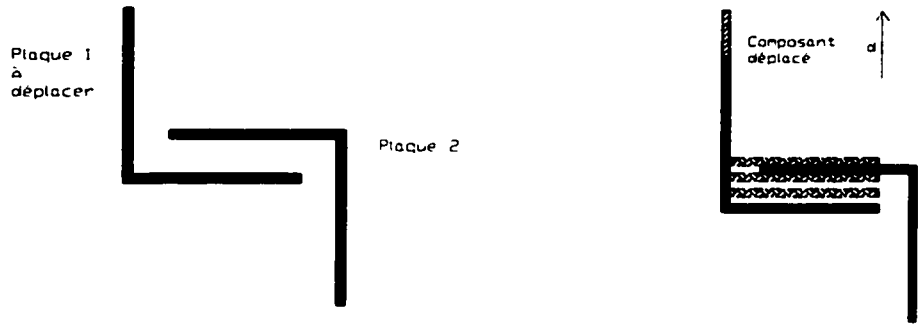


Figure 4.5 : Problème possible en cas de présence de plaques

Dans ce cas précis le programme ne verra pas la collision : le pas de déplacement est égal au dixième de la largeur minimale des deux solides, et cette valeur est supérieure à leur épaisseur.

Lorsqu'il y a des composants très fins, il faut toujours veiller à ce que *pas_elementaire* soit strictement inférieur à la somme des deux plus petites épaisseurs présentes dans la structure. Il est important d'être conscient du problème s'il se présente pour pouvoir y remédier en choisissant un pas élémentaire plus petit.

Améliorations possibles

Avec CATIA, nous pourrions essayer de connaître la largeur des plaques de façon automatique lorsqu'il y en a. Nous pourrions ainsi modifier le pas de déplacement selon le type de solides déplacé.

Par contre avec AutoCAD, les possibilités d'amélioration sont plus limitées car ce logiciel offre moins de fonctionnalités (dans sa version actuelle et pour la modélisation solide).

4-2.2 Calcul de T($comp_i$. $comp_j$) pour toutes les directions

Pour terminer le calcul de $T(comp_i, comp_j)$, le programme vérifie toutes les directions pour obtenir les autres valeurs de translation de la paire.

Si il n'existe pas de direction connue qui permet de désassembler les deux composants ($T_d=0$ pour toutes les directions), alors Polyassemblage est capable de rechercher par lui-même d'autres directions de désassemblage, à condition qu'elles respectent les conditions évoquées dans l'introduction (rectilignes, sans rotation ni déformations des solides...).

4-3 Recherche d'une nouvelle direction d'assemblage

4-3.1 Algorithme général

La recherche d'une nouvelle direction se fait par étapes. Le programme essaye d'abord les directions X, Y et Z.

Si les composants ne peuvent pas se désassembler selon l'un de ces 3 axes, il cherche ensuite des directions faciles à connaître et très probablement bonnes : les directions principales des deux composants à assembler (c'est à dire ses axes de symétrie).

Si ces directions ne conviennent toujours pas, le programme en cherche d'autres « en aveugle » : il essaie le plus de directions possibles dans le plan (recherche en « coordonnées cylindriques »), puis dans l'espace (« coordonnées sphériques ») jusqu'à en trouver une bonne.

S'il n'en trouve définitivement pas, le programme s'arrête car il sait qu'il ne sera pas capable d'assembler le produit par la suite.

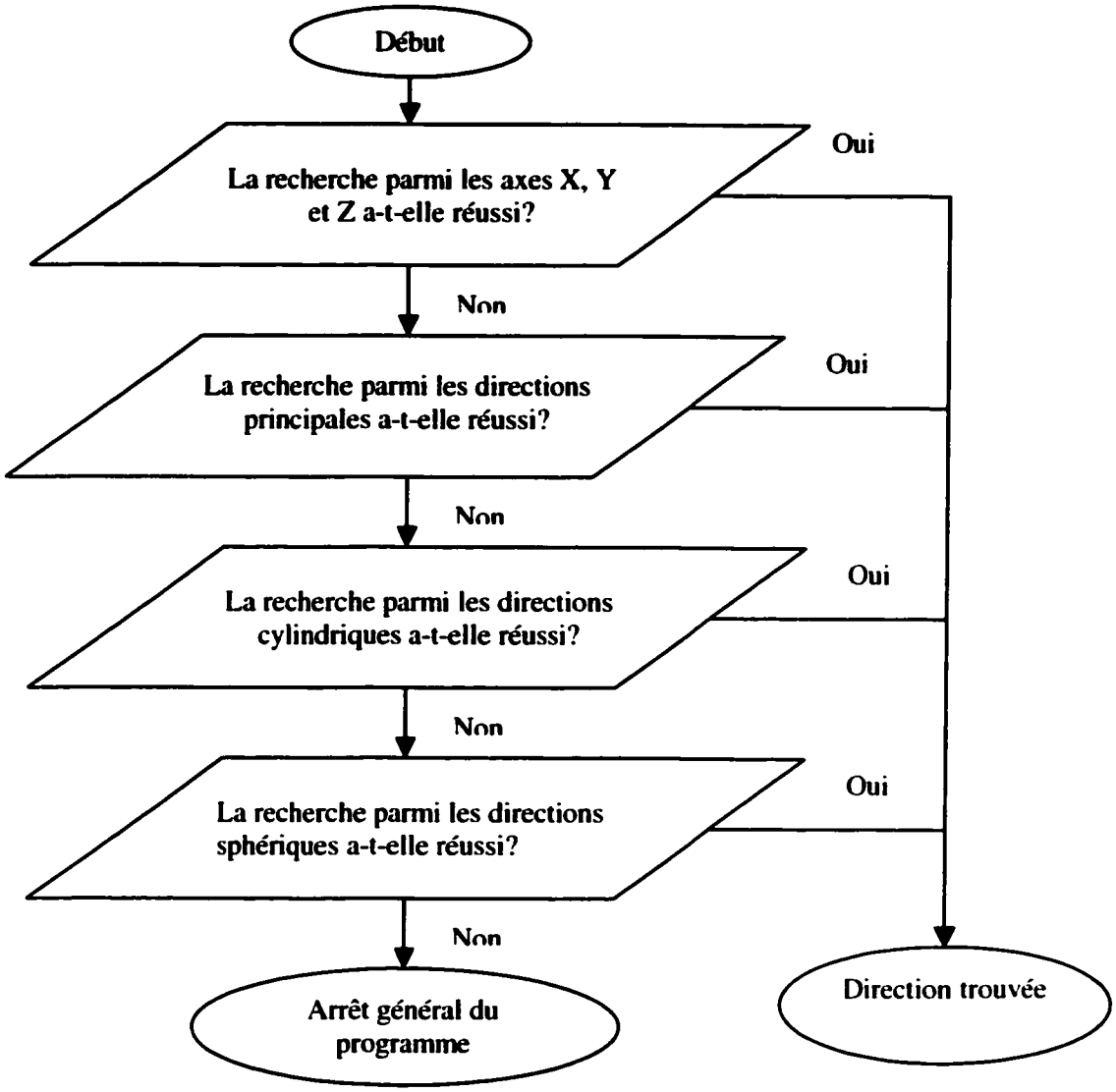


Figure 4.6 : Algorithme pour trouver une nouvelle direction de désassemblage

Lorsqu'une nouvelle direction d'assemblage est trouvée, le programme la rajoute à la liste des directions d'assemblage initiales.

4-3.2 Recherche parmi les axes X, Y et Z

Le programme déplace *comp*$_j$ dans les 6 directions X+, X-, Y+, Y-, Z+ et Z-, et regarde si l'une de ces directions est une bonne direction de désassemblage avec l'algorithme décrit précédemment.

4-3.3 Recherche parmi les directions principales des solides

Les directions principales d'un composant sont ses axes de symétrie (AutoCAD), ou les axes dans lesquels le composant a été dessiné pour la première fois (CATIA).

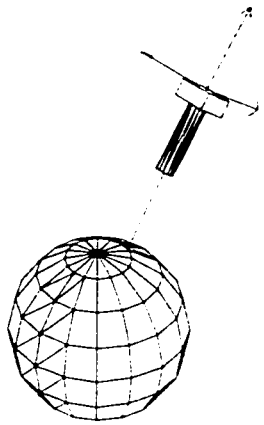


Figure 4.7 : Insertion d'un composant selon l'un de ses axes principaux

Ces directions principales sont très faciles à connaître dans CATIA et AutoCAD. Cette opportunité nous est extrêmement utile car dans la pratique, les composants sont le plus souvent insérés selon une de ces directions.

4-3.4 Recherche dans le plan

Cette option permet d'essayer toutes les directions du plan pour savoir si elles sont des bonnes directions de désassemblage. Le programme peut fonctionner dans les plans XY, YZ ou XZ, mais par défaut il recherche des directions dans le plan XY. L'utilisateur doit indiquer avant de lancer le programme s'il souhaite que la recherche se fasse plutôt dans un des deux autres plans.

Le programme essaie chaque direction en partant de l'axe X avec un pas angulaire de 5 degrés. Là aussi, l'utilisateur peut modifier ce pas s'il le souhaite. Le programme s'arrête automatiquement lorsqu'il a trouvé une bonne direction.

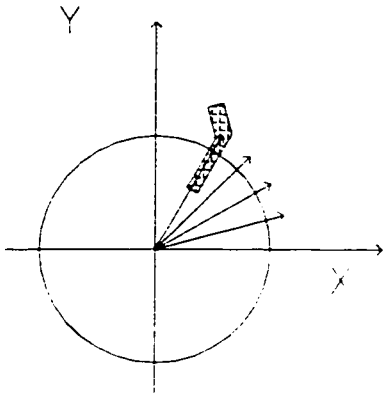


Figure 4.8 : Différentes directions essayées pendant la recherche en cylindriques

L'algorithme est le suivant :

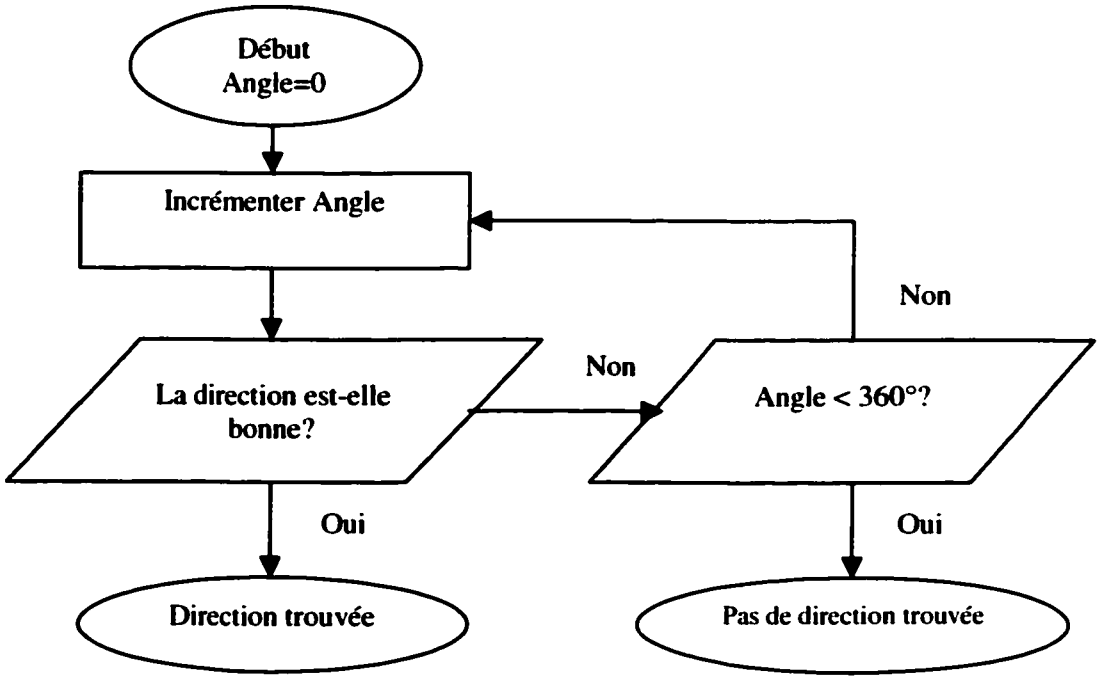
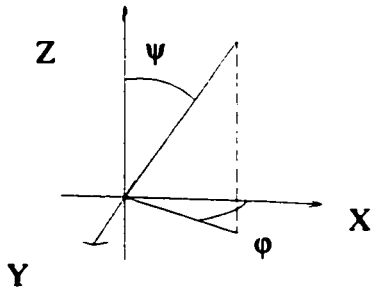


Figure 4.9 : Algorithme pour trouver une direction de désassemblage en cylindriques

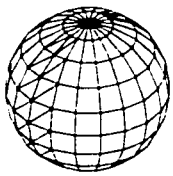
Le principal problème est qu'on n'est pas sûr de trouver la direction d'insertion, même si elle existe et qu'elle est bien dans le bon plan. Le pas angulaire est réglé à 5 degrés par défaut et est modifiable par l'utilisateur. Mais même en prenant un pas très petit (qui rallongera les temps des calculs), le programme ne pourra jamais essayer tous les angles possibles.

4-3.5 Recherche dans l'espace

Si aucune direction n'a été trouvée, dans le plan la recherche se poursuit en essayant toutes les directions de l'espace. Pour chaque valeur de ψ (de 0 à 180 degrés), φ tourne de 0 à 360 degrés. Les pas angulaires de φ et ψ ont la même valeur par défaut (5 degrés), et sont modifiables. Par exemple sur la figure 4.10 b, le pas angulaire a été fixé à 18 degrés pour éclaircir la figure.



a) Les directions essayées sont définies par (φ, ψ)



b) Les points d'intersection des lignes sont les directions essayées par Polyassemblage avec dans ce cas un pas angulaire de 18 degrés pour φ et ψ

Figure 4.10 : Directions essayées en coordonnées sphériques

L'algorithme est le même que précédemment, sauf qu'on incrémente les deux paramètre φ et ψ pour parcourir toute la sphère. Mais nous avons le même problème que précédemment :

même en prenant un pas très petit le programme ne pourra jamais essayer tous les angles possibles.

Nous ne proposons pas de solution pour ce problème dans ce projet. Mais avec CATIA il est possible de connaître l'importance de la collision pendant un test. Parmi les directions essayées il serait peut-être possible de rechercher le couple (φ, ψ) qui minimise cette valeur. Nous pourrions ensuite nous rapprocher par approximations successives de la bonne direction d'assemblage. Mais cette proposition serait très coûteuse en temps de calculs.

4-3.6 Mise à jour des données après avoir trouvé une nouvelle direction d'assemblage

Rajout de deux directions à la liste des directions

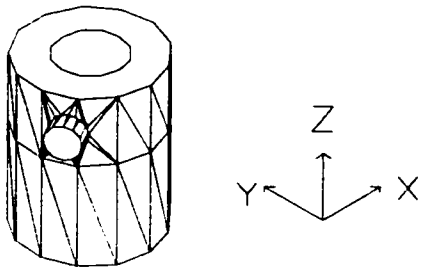
Lorsqu'une nouvelle direction d'assemblage a été trouvée, le programme l'ajoute à la liste des directions connues. Il ajoute aussi la direction opposée, car nous savons déjà qu'elle sera la bonne direction de désassemblage pour la paire symétrique *(comp$_j$, comp$_i$)*.

Calcul des matrices de translation pour ces deux directions

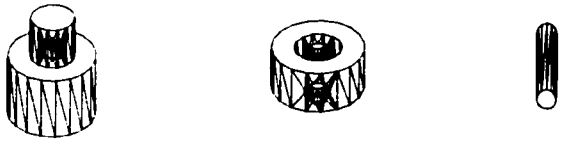
Le programme doit ensuite calculer les matrices de translation pour toutes les paires de solides avec ces deux nouvelles directions.

4-3.7 Exemple

Pour illustrer cet algorithme, nous allons prendre pour exemple la figure ci-dessous et suivre le programme pas à pas.



a) Produit assemblé



Base Dessus Tige

b) Liste des composants

Figure 4.11 : Exemple repris de la figure 3.7 sans les bouchons

Initialement le programme ne connaît aucune direction d'assemblage. Les 3 paires de solides sont *(Base, Dessus)*, *(Base, Tige)* et *(Dessus, Tige)*. Nous allons voir ci-dessous quelles sont leurs valeurs de contact et de translation.

Première paire (Base, Dessus)

Le programme commence par rechercher une direction pour désassembler la paire parmi les axes X, Y et Z. X et Y ne sont pas des bonnes directions pour désassembler *Base* et *Dessus*. Par contre la direction Z+ est une bonne direction de désassemblage (aucune collision n'est détectée lors du déplacement de *Dessus* selon la direction Z+). La liste des directions devient donc :

Direction n°1 : (0, 0, 1)

Direction n°2 : (0, 0, -1)

Dessus peut se désassembler de *Base* selon la direction 1 (Z+) mais pas selon la direction 2 (Z-). Nous obtenons donc pour *T(Base, Dessus)* :

T(Base, Dessus) = (1, 0)

Seconde paire (Base, Tige)

Ni la direction 1, ni la direction 2 ne permet de désassembler *Tige* de *Dessus*.

T(Base, Tige) = (0, 0)

Le programme doit donc chercher une nouvelle direction d'assemblage pour désassembler les deux composants. X et Y ne conviennent toujours pas. Par contre la direction principale de la tige (définie approximativement par le vecteur (0.866, 0.5, 0), comme vu plus haut) est une bonne direction d'assemblage. Le programme est très facilement capable de la trouver avec l'algorithme de la figure 4.6. Il la rajoute donc, avec la direction opposée, à la liste des directions :

Direction n°1 : (0, 0, 1)

Direction n°2 : (0, 0, -1)

Direction n°3 : (0.866, 0.5, 0)

Direction n°4 : (-0.866, -0.5, 0)

Calcul final de *T(Base, Tige)* : les directions 3 et 4 permettent toutes les deux de désassembler *Tige* de *Base*. Leur valeur de translation vaut donc 1 pour les deux directions.

T(Base, Tige) = (0, 0, 1, 1)

Calcul de *T(Base, Dessus)* avec les deux nouvelles directions (3 et 4) :

T(Base, Dessus) = (1, 0, 0, 0)

Troisième paire (Dessus, Tige)

On obtient sans problème :

T(Dessus, Tige) = (0, 0, 1, 1)

Solution finale

Les matrices de translation sont :

T(Base, Dessus) = (1, 0, 0, 0)

T(Base, Tige) = (0, 0, 1, 1)

T(Dessus, Tige) = (0, 0, 1, 1)

Et par symétrie :

T(Dessus, Base) = (0, 1, 0, 0)

T(Tige, Base) = (0, 0, 1, 1)

T(Tige, Dessus) = (0, 0, 1, 1)

La liste finale des directions d'assemblage est :

Direction n°1 : (0, 0, 1)

Direction n°2 : (0, 0, -1)

Direction n°3 : (0.866, 0.5, 0)

Direction n°4 : (-0.866, -0.5, 0)

4-4 Calcul des matrices de contact

Ces matrices indiquent pour chaque paire de composants *(comp$_i$, comp$_j$)* s'ils se touchent ou non. Le calcul se fait différemment, selon que le programme travaille avec AutoCAD ou CATIA.

4-4.1 Avec AutoCAD

Le programme déplace *comp$_j$* d'un pas élémentaire selon chacune des 6 directions X+, X-, Y+, Y-, Z+ et Z-. Ce pas élémentaire a été fixé au dixième du *pas_elementaire* présenté dans la partie précédente. Si *comp$_j$* entre en collision avec *comp$_i$* dans l'un de ces 6 cas, alors c'est qu'il y a contact.

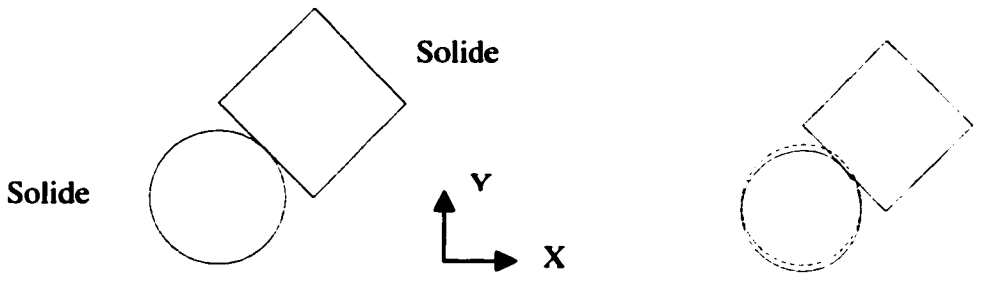


Figure 4.12 : Test de contact avec AutoCAD

Dans l'exemple ci-dessus, le contact sera repéré lorsque le solide 1 sera déplacé selon la direction Y+ ou X+.

Dans tous les cas, il est suffisant de ne déplacer les solides que selon que ces six directions.

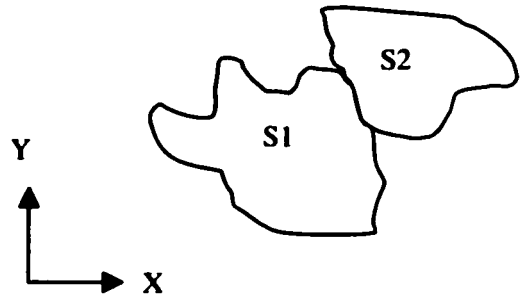


Figure 4.13 : Contact entre deux solides orienté dans une direction quelconque

Sur cette figure, les solides S1 et S2 sont en contact selon une surface orientée dans une direction quelconque. On voit qu'il y aura quand même une collision au cours des déplacements selon les axes X et Y.

La principale limite de cette approche est qu'il peut y avoir un espace libre très petit entre deux composants qui ne se touchent pas. Le programme verra un contact si cet espace est inférieur au dixième de *pas_elementaire*. Mais ces cas sont très rares.

4-4.2 Avec CATIA

La fonction qui calcule les interférences entre deux solides présentée dans le chapitre 2 indique aussi s'il y a simple contact entre deux solides. Nous avons juste à faire ce test pour chaque paire de composants.

4-5 Conclusion

Ce chapitre nous a permis de comprendre comment à partir de la description géométrique de l'assemblage, il est possible de retrouver la valeur des matrices de contact et de translation pour toutes les paires de solides de l'assemblage. Le programme est aussi capable de chercher et de trouver des nouvelles directions d'assemblage pour assembler le produit.

Ce chapitre a aussi mis l'accent sur les limites et les imprécisions des résultats, phénomène inévitable lorsqu'on travaille directement sur des objets physiques (même s'ils sont virtuels).

Nous avons donc maintenant tous les éléments en main pour générer toutes les séquences d'assemblage possibles. Le programme calcule d'abord les matrices de contact et de translation pour toutes les paires de solides, en cherchant éventuellement des nouvelles directions. Ensuite il génère toutes les séquences d'assemblage du produit avec l'algorithme présenté dans le chapitre 3.

Il existe souvent un très grand nombre de séquences possibles et l'utilisateur a besoin de faire un choix rapide. Nous allons donc voir dans le chapitre suivant comment réduire le nombre de séquences d'assemblage trouvées, les évaluer et les classer.

CHAPITRE 5 – ÉVALUATION DES SÉQUENCES D'ASSEMBLAGE

5-1 Introduction

L'exemple très simple de la figure ci-dessous possède 96 séquences d'assemblage, dont 8 sont linéaires. Il est indispensable de pouvoir réduire ce nombre et de classer les séquences restantes pour rendre le logiciel utilisable dans la pratique.

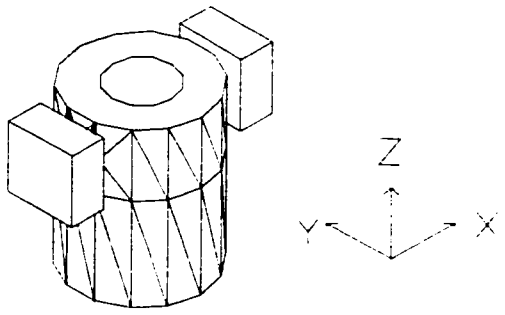


Figure 5.1 : Assemblage pour illustrer les critères d'évaluation programmés

Les critères programmés dans le cadre de ce projet sont basés uniquement sur les données disponibles dans la géométrie du dessin. Aucun ne nécessite la saisie de données complémentaires par l'utilisateur : les contraintes de production ou d'outillages sont donc absentes, de même que l'évaluation des séquences d'après leur coût. Ces critères sont des critères classiques d'évaluation des séquences d'assemblage.

Ils se regroupent en deux types : d'une part les critères d'évaluation attribuent des notes à chaque séquence et permettent d'effectuer un premier tri pour ces séquences. D'autre part les filtres permettent de « supprimer » toutes les séquences qui ne respectent pas le critère programmé.

Les séquences ne sont jamais vraiment supprimées : elles restent en mémoire mais ne sont plus affichées. Ainsi, si l'utilisateur change d'avis et décide de désactiver un filtre qu'il avait activé, il peut le faire très facilement.

5-2 Évaluer les séquences trouvées

Pour chacun des 6 critères d'évaluation qui ont été programmés, Polyassemblage attribue une note à chaque séquence, de 0 à 10.

5-2.1 Taille du premier élément

La note attribuée à chaque séquence pour ce critère est d'autant meilleure que le composant de départ est plus volumineux. Le but de ce critère est de favoriser les séquences qui débutent par les éléments les plus gros. Ces séquences sont généralement meilleures, car la manipulation de ces éléments est peu pratique : ils sont lourds, plus difficiles à manipuler et plus encombrants.

La formule pour calculer la note est :

$$Note_{Débuter\_par\_le\_plus\_gros} = 10 \times \frac{Volume\_premier\_element}{Volume\_plus\_gros}$$

5-2.2 Nombre de directions identiques

Ici, la note est d'autant meilleure qu'il y a peu de directions d'assemblage différentes. Pour chaque séquence, la formule pour calculer la note est :

$$Note_{Nombre\_directions\_identiques} = 10 \times \frac{Nombre\_\min\_directions\_assemblage}{Nombre\_directions\_assemblage}$$

Le but de ce critère est de réduire au minimum le nombre de réorientations du produit lors de l'assemblage.

Mais attention ! Les meilleures notes pour ce critère seront souvent attribuées aux séquences non linéaires, si elles sont générées :

Meilleures séquence non linéaire :

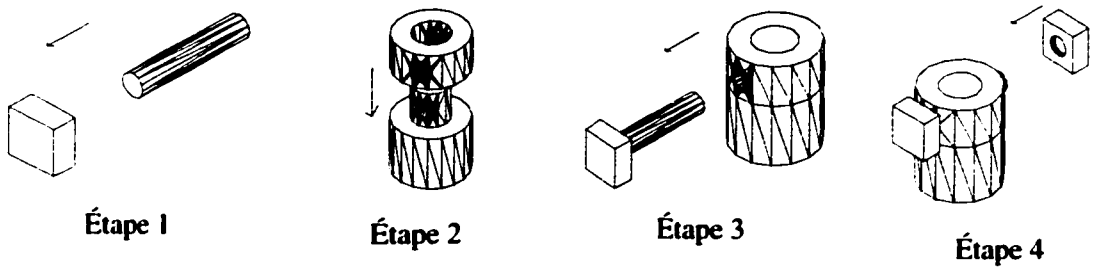


Figure 5.2 : Séquence non linéaire

Deux directions d'insertion sont suffisantes : c'est le minimum possible pour cet assemblage et la note attribuée à cette séquence sera 10.

Meilleure séquence linéaire :

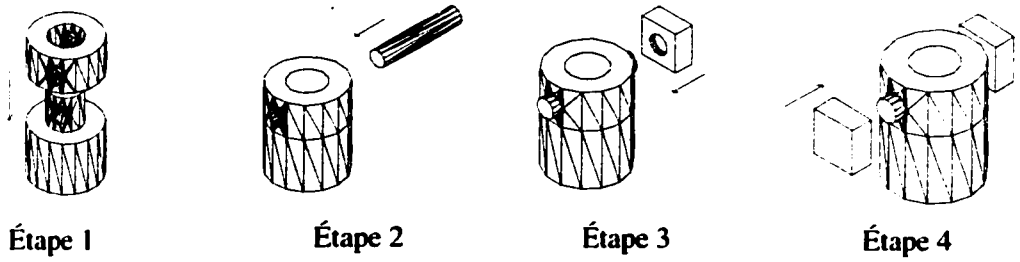


Figure 5.3 : Séquence linéaire

Pour les séquences linéaires au minimum trois directions d'assemblage sont nécessaires. Leur note sera au mieux :

$$Note_{Nombre\_directions\_identiques} = 10 \times \frac{2}{3} = 6.67 \approx 7$$

Il faut donc utiliser ce critère pour comparer des séquences qui ont le même degré de linéarité (la même note pour le critère suivant).

5-2.3 Nombre de sous-assemblages

Si l'utilisateur a choisi de générer toutes les séquences (linéaires et non linéaires) ce critère évaluera leur linéarité et donnera les meilleures notes aux séquences les plus linéaires.

Il est possible de calculer le nombre de sous-assemblages qui interviennent dans chaque séquence. Le programme peut donc calculer la note avec la formule :

$$Note_{Nombre\_sous\_assemblages} = 10 \times \frac{Nombre\_\min\_sous\_assemblages}{Nombre\_sous\_assemblages}$$

Il est souvent utile dans l'industrie de passer par des sous-assemblages pour simplifier le processus de production. Mais lorsque c'est le cas, le concepteur les fait apparaître clairement dans la structure générale du produit. Dans les autres cas, la gamme d'assemblage la plus linéaire est aussi la plus simple, car elle évite les préassemblages (qui peuvent en outre se révéler instables tant que l'assemblage n'est pas terminé). C'est pourquoi Polyassemblage n'a pas à essayer de créer de nouveaux sous-assemblages, sauf bien sûr si c'est indispensable pour la génération des séquences.

Par exemple ci dessous se trouve la structure d'un alternateur qui contient un sous-assemblage nommé *Ensemble frontal.* Polyassemblage va tenir compte automatiquement de cette structure, mais ne va pas créer de nouveaux sous-assemblages lors de l'analyse du produit.

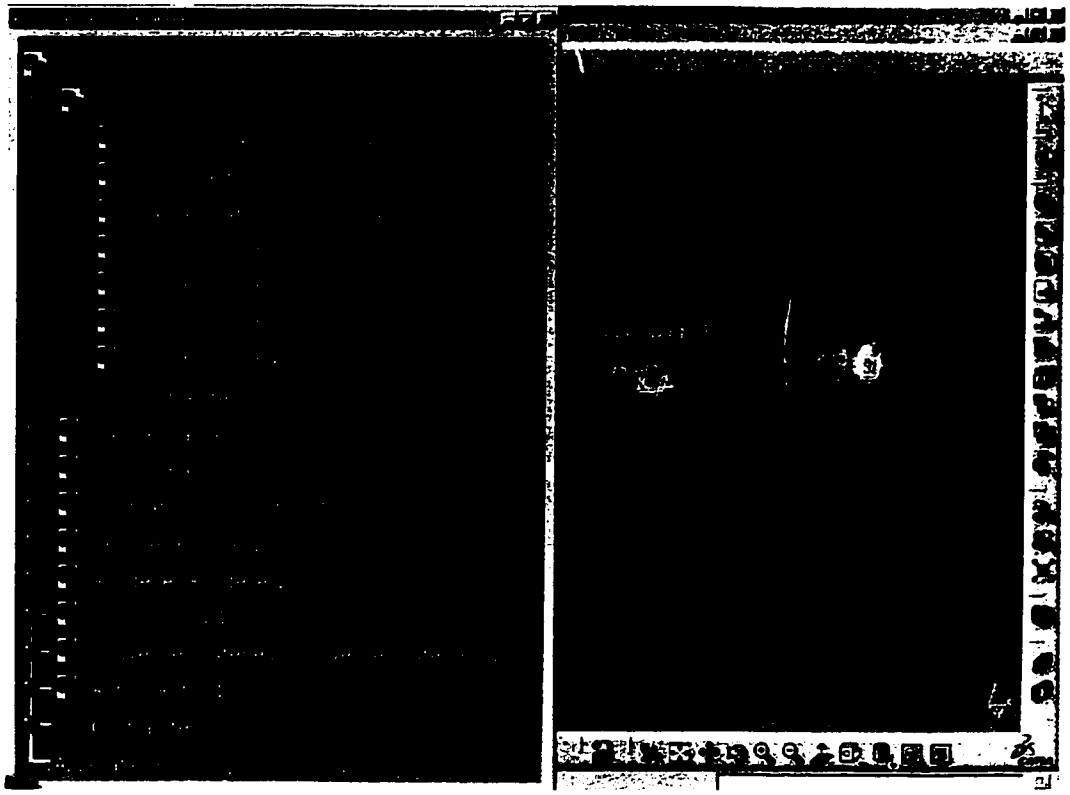


Figure 5.4 : Produit constitué d'un sous-assemblage

5-2.4 Stabilité des assemblages partiels

Les composants qui apportent la cohésion du produit (vis, tiges, etc.) ne sont pas forcément assemblés directement après les composants qu'ils sont sensés tenir. Dans ce cas ces composants ne sont pas bien fixés et les états associés sont dits instables. Le but de ce critère est de favoriser les séquences dont les assemblages partiels sont les plus stables.

Pour calculer la stabilité d'un assemblage partiel le programme compte le nombre de degrés de liberté de chaque composant (excepté le premier qui sert de base) par rapport aux autres. Le premier composant qui sert de base n'intervient pas car il est supposé être fixé sur l'établi.

En fait le programme calcule plutôt l'instabilité des états et non leur stabilité, comme l'indique l'algorithme ci-dessous :

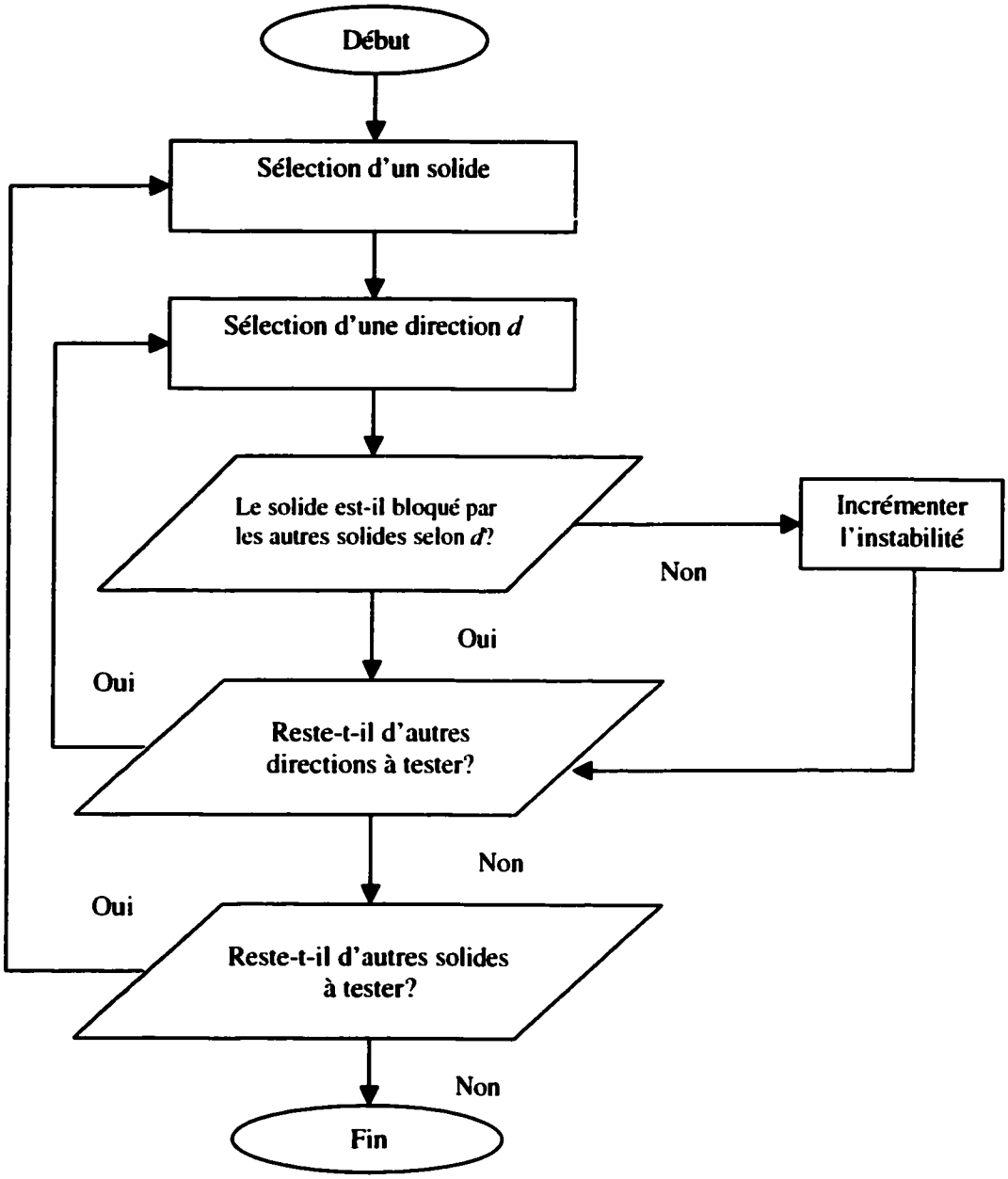


Figure 5.5 : Algorithme pour calculer le degré d'instabilité d'un état

Ensuite le programme calcule le degré d'instabilité de chaque séquence en additionnant les degrés d'instabilité de chaque état qui la constitue.

Enfin, la note finale attribuée à chaque séquence est :

$$Note_{Stabilité} = 10 \times \frac{Degré\_\min\_d'instabilité}{Degré\_d'instabilité}$$

Toutes les séquences linéaires de notre exemple habituel ont la même note pour ce critère. Nous allons donc l'illustrer avec l'exemple ci-dessous :

Séquence 1 :

Figure 5.6 : Séquence stable

Pour calculer l'instabilité de la séquence, le programme calcule l'instabilité de son seul état intermédiaire, c'est-à-dire de l'état *(Cube_du_bas; Tige)*. *Cube_du_bas* est supposé être fixé sur l'établi. *Tige* peut se déplacer selon Z+ et Z- mais elle est bloquée dans les autres directions. Son degré d'instabilité est donc 2. C'est le degré d'instabilité total de la séquence.

Séquence 2 :

Figure 5.7 : Séquence moins stable

Avec cette séquence le seul état intermédiaire est l'état *(Cube_du_bas, Cube_du_haut)*. De même que précédemment, *Cube_du_bas* est supposé être fixé sur l'établi. Rien n'empêche *Cube_du_haut* de se déplacer selon les 4 directions horizontales et 1 direction verticale. Le degré d'instabilité de l'état, et donc de la séquence, est donc 5. La note attribuée à cette séquence sera donc :

$$Note_{Stabilité} = 10 \times \frac{2}{5} = 4$$

Mais attention ! Ici aussi les meilleures notes pour ce critère seront forcément attribuées aux séquences non linéaires. Pour chaque sous-assemblage, l'un des composants est supposé être lié à l'établi, donc les sous-assemblage sont forcément plus stables. Ce critère est par contre utile pour comparer les séquences qui ont le même degré de linéarité.

5-2.5 Regroupement des directions d'assemblage identiques

Ce critère favorise les séquences qui regroupent les différentes directions d'assemblage entre elles. La formule pour calculer cette note est :

$$Note_{Regroupement\_directions\_identiques} = 10 \times \frac{Nombre\_min\_directions\_assemblage}{Nombre\_de\_changements\_de\_direction}$$

L'intérêt de ce critère est aussi de minimiser les réorientations du produit. L'exemple ci-dessous compare deux séquences presque équivalentes, mais qui ont deux notes différentes :

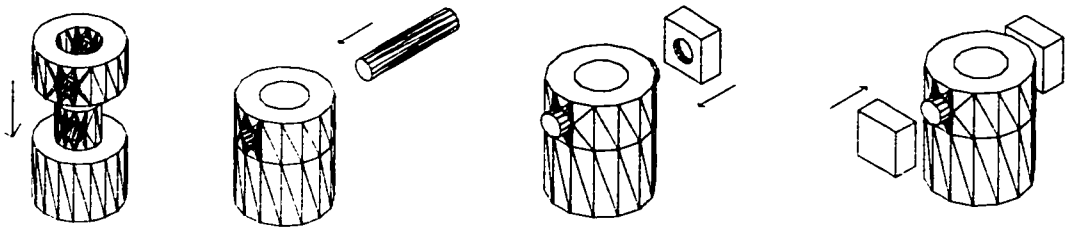


Figure 5.8 : Séquence linéaire avec 3 changements de direction

La note de cette séquence sera :

$$Note_{\text{Re groupement\_direction\_identiques}} = 10 \times \frac{2}{3} = 6.67 \approx 7$$

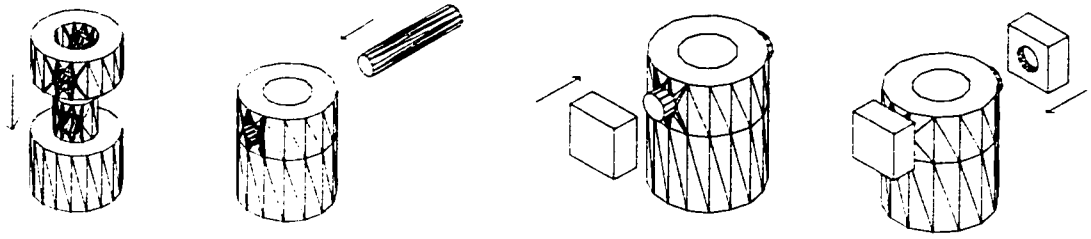


Figure 5.9 : Séquence linéaire avec 4 changements de direction

La note de cette séquence sera :

$$Note_{\text{Re groupement\_direction\_identiques}} = 10 \times \frac{2}{4} = 5$$

5-2.6 Note générale

Après avoir calculé les notes pour tous les critères mentionnés ci-dessus, le programme en calcule la moyenne. Les coefficients de pondération α, β, δ, γ et λ sont égaux à 1 par défaut et l'utilisateur peut les modifier.

$$Note_{générale} = \frac{\alpha Note_1 + \beta Note_2 + \delta Note_3 + \gamma Note_4 + \lambda Note_5}{\alpha + \beta + \delta + \gamma + \lambda}$$

Finalement les séquences sont classées selon cette note finale. Les meilleures se trouvent en haut de la liste.

5-3 Critères pour réduire le nombre de séquences

Le but des critères présentés dans cette partie est de réduire le nombre de séquences affichées : le programme met de côté toutes les séquences qui ne respectent pas chacun des critères activés.

5-3.1 Interdire l'insertion de composants depuis le bas

Cette fonction interdit les séquences qui contiennent des composants insérés depuis le bas (direction Z-). Ce critère ne peut être activé que si l'assemblage de composants depuis le bas n'a pas été interdit lors de la génération des séquences. Il est très facile à programmer : dès qu'au cours d'une séquence un composant est inséré selon la direction Z-, la séquence est automatiquement mise de côté.

Interdit Autorisé

Figure 5.10 : Exemple de séquence interdite et de séquence autorisée

Ce critère ne doit pas toujours être actif, surtout lorsque certains composants sont volumineux ou difficiles à retourner (par exemple dans la fabrication automobile ou aéronautique). Dans ces cas il arrive souvent que des composants doivent être insérés depuis le bas.

5-3.2 Débuter par l'élément le plus volumineux

Lorsque ce critère est actif, seules les séquences qui débutent par l'élément le plus volumineux sont affichées. Ce sont les séquences dont la note pour le critère « Taille du premier élément » est 10.

5-3.3 Minimum de directions d'assemblage

Lorsque ce critère est actif, seules les séquences dont la note est 10 pour le critère « Nombre de directions identiques » sont affichées.

5-3.4 Minimum de sous-assemblages

Lorsque ce critère est actif, seules les séquences dont la note est 10 pour le critère « Nombre de sous-assemblage » sont affichées. Ce sont les séquences linéaires s'il y en a, sinon ce sont celles qui ont le minimum de sous-assemblages.

5-3.5 Interdire une séquence spécifique

L'utilisateur peut aussi interdire certaines séquences parmi les séquences affichées. Il le fait « à la main », éventuellement après les avoir visualisées. Il le fait en tenant compte de contraintes qui n'ont pas encore été programmées dans Polyassemblage (contraintes d'outillage, de production...), ou pour n'importe quelle autre raison.

Il pourra bien sûr revenir en arrière par la suite en vidant la liste des séquences interdites.

5-3.6 Interdire un état spécifique

Polyassemblage propose aussi la représentation des séquences trouvées avec le *And/Or Graph* (voir chapitre 6). Comme écrit dans le chapitre 1, ce graphe permet à l'utilisateur de voir tous les états intermédiaires qui mènent à l'assemblage final.

L'utilisateur peut visualiser certains de ces états dans AutoCAD ou CATIA et les interdire. Polyassemblage met de côté automatiquement toutes les séquences qui les contiennent.

Comme précédemment, il sera possible de revenir en arrière en vidant la liste des états interdits.

5-4 Conseils d'utilisation

Nous avons présenté ici de nombreux critères d'évaluation. Nous conseillons de les appliquer un par un, pour aller progressivement vers la meilleure séquence.

Les critères d'évaluation ne sont pas indépendants entre eux. Les cas où une séquence aura 10/10 partout sont rares (en particulier lorsqu'on considère les séquences non linéaires, aucun séquence n'aura 10 à la linéarité, 10 à la stabilité et 10 au minimum de directions différentes). Il est possible de jouer avec les coefficients de pondération pour identifier la meilleure séquence en fonction de chaque cas particulier.

Enfin il est important de visualiser les séquences et les états pour bien les comprendre et les comparer (cet outil est présenté dans le chapitre 6).

5-5 Conclusion

Nous avons présenté dans ce chapitre tous les critères d'évaluation qui ont été programmés dans Polyassemblage. Il sera possible de programmer d'autres critères pour perfectionner le logiciel. Ces critères pourront être basés sur des informations extraites de la géométrie uniquement, ou prendre en compte d'autres informations.

Nous allons maintenant présenter d'autres outils destinés à simplifier la tâche de l'utilisateur, pour lui permettre d'identifier plus facilement la meilleure séquence d'assemblage.

CHAPITRE 6 – AFFICHAGE DES SÉQUENCES ET AUTRES OUTILS PRATIQUES

6-1 Introduction

Dans ce chapitre nous allons montrer comment sont affichés les résultats trouvés par Polyassemblage. Cet affichage est sous deux formes :

- La « fenêtre des séquences » affiche les séquences trouvées;
- La « fenêtre des états » affiche le *And/Or Graph.*

Ensuite nous montrerons les autres outils programmés dans Polyassemblage qui servent à aider l'utilisateur du programme :

- Visualisation des états intermédiaires et des séquences trouvées;
- Réglage des paramètres de l'étude;
- Autres outils : sauvegarde, impression des résultats, etc.

6-2 Affichage des séquences

6-2.1 Affichage des séquences d'assemblage

Les séquences d'assemblage sont présentées dans la « fenêtre des séquences ». Seules les séquences qui respectent les filtres activés sont montrées. Comme indiqué dans le chapitre 5, elles sont classées selon leur note générale (les meilleures sont en haut).

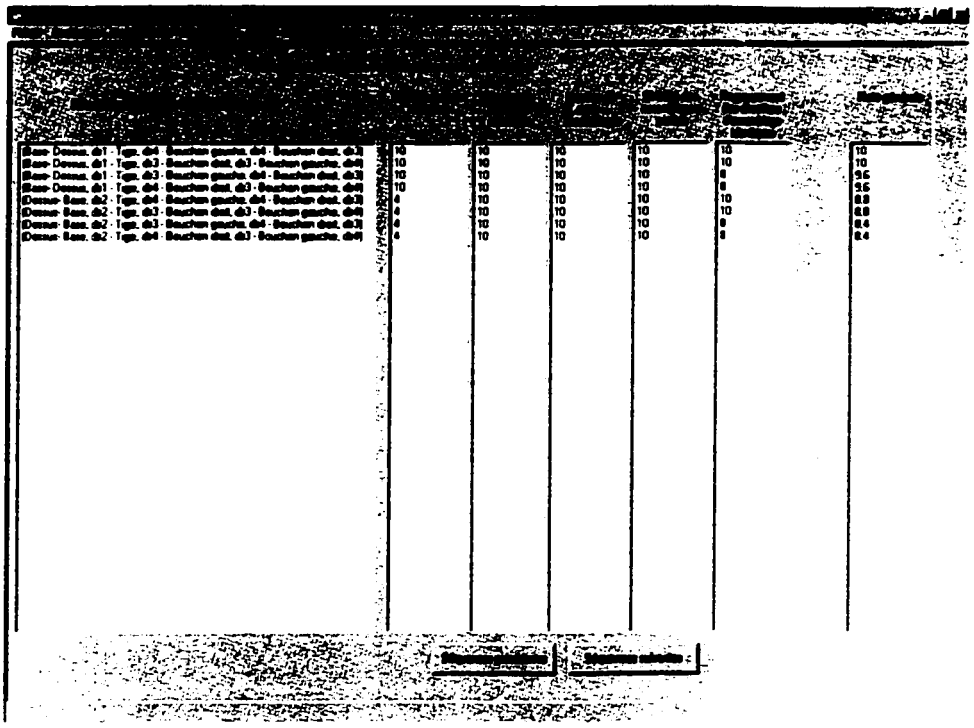


Figure 6.1 : Fenêtre des séquences

Cette fenêtre est montrée plus claire dans l'annexe 2. Dans la colonne de gauche de la fenêtre se trouve la description détaillée de chaque séquence. Dans les 5 colonnes suivantes sont affichées les notes pour les 5 critères d'évaluation programmés dans Polyassemblage, et à droite se trouve la moyenne de ces notes (7° colonne).

Description détaillée des séquences

Pour une séquence linéaire

L'affichage est de ce type :

5_(Base, Dessus – dir1, Tige – dir3, Bouchon_droit – dir3, Bouchon_gauche – dir4)

Le chiffre du début (5) représente le numéro de la séquence : chaque séquence est identifiée de manière unique à un nombre. Ensuite, les noms représentent les noms des solides à insérer et les numéros suivis de « dir » représentent le numéro de la direction d'insertion. Le

premier solide n'a pas de direction d'insertion car c'est le solide qui sert de base. La formule ci-dessus représente la séquence illustrée dans la séquence 6.2 :

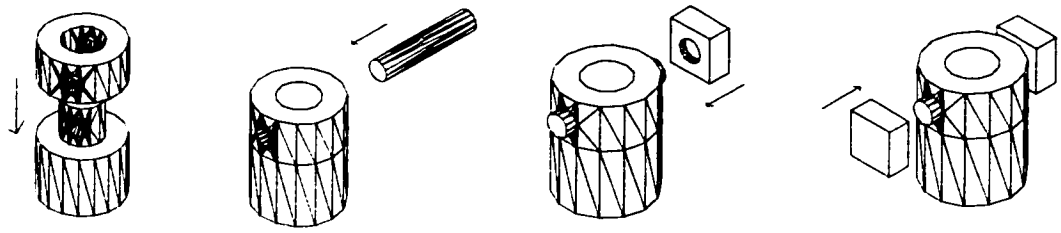


Figure 6.2 : Séquence d'assemblage décrite par une formule linéaire

Pour une séquence non linéaire :

L'affichage est du même type sauf que le nom d'un ou plusieurs composants peut être remplacés par un ou plusieurs sous-assemblage :

6_(Bouchon_gauche, Tige – dir3, (Base, Dessus – dir1) – dir3, Bouchon_droit – dir3)

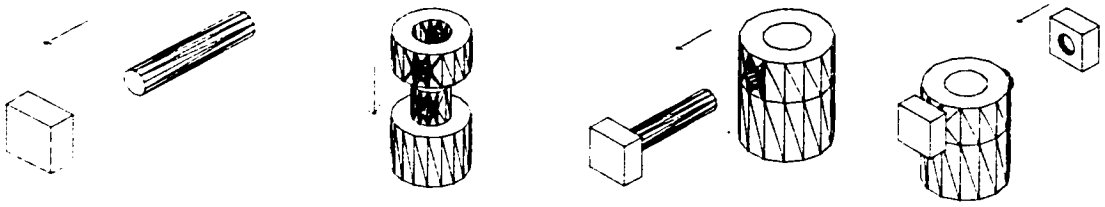


Figure 6.3 : Séquence d'assemblage décrite par une formule non linéaire

6-2.2 Affichage du And/Or Graph

Il est possible de visualiser l'ensemble des séquences avec le *And/Or Graph* décrit dans le chapitre 1. Celui-ci est visible dans la fenêtre « Fenêtre des états » accessible en cliquant sur la barre de menus « Fenêtre ».

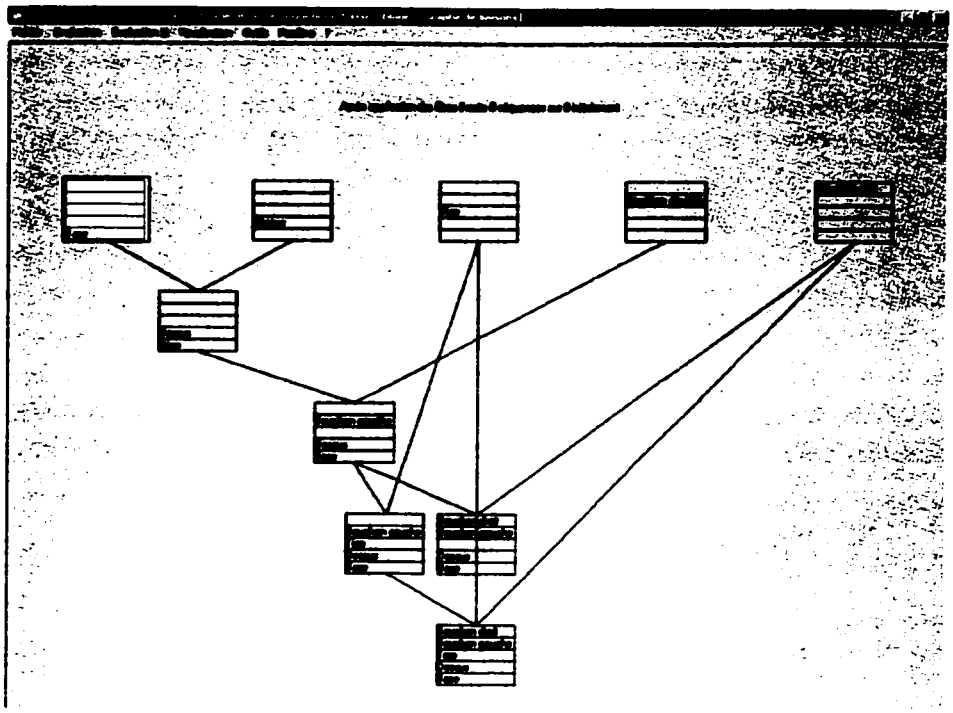


Figure 6.4 : Fenêtre des états

Il y a aussi un exemplaire plus clair de cet affichage dans l'annexe 2.

6-3 Outils de visualisation

6-3.1 Visualisation des séquences

La séquence courante est par défaut la séquence de tête sur la figure 6.1. Il est possible de sélectionner n'importe quelle autre séquence en cliquant dessus. Pour visualiser la séquence courante dans CATIA ou AutoCAD, l'utilisateur doit appeler la fonction « Visualiser la séquence courante » dans le menu « Visualiser », et aller dans CATIA ou AutoCAD pour voir les solides se déplacer comme indiqué dans la séquence.

6-3.2 Visualisation des états

L'état courant est l'état surligné en blanc sur la figure 6.4. Il est possible de sélectionner un autre état en cliquant dessus. Pour visualiser l'état courant dans CATIA ou AutoCAD, l'utilisateur doit appeler la fonction « Visualiser l'état courant » et aller dans CATIA ou AutoCAD pour voir l'état.

6-4 Paramètres de l'étude

Nous avons vu tout au long des chapitres précédents qu'il était possible de modifier un certain nombre de paramètres. Nous allons récapituler dans cette partie la liste de ceux que l'utilisateur peut modifier, et dans quels cas il est conseillé de le faire.

6-4.1 Paramètres concernant l'extraction de données

La feuille qui récapitule tous ces paramètres se trouve dans l'annexe 3. Elle est n'accessible à l'utilisateur qu'avant de commencer l'étude.

Choix des directions à tester

Pour accélérer la recherche de nouvelles directions de désassemblage, l'utilisateur peut choisir de ne pas rechercher certains types de directions (directions principales, cylindriques ou sphériques). Par exemple, s'il est sûr qu'il n'y aura pas de directions d'assemblage en coordonnées sphériques, alors il peut annuler cette option. Par défaut tous les types de recherche sont installés. Pour la recherche de directions en cylindriques, l'utilisateur doit aussi indiquer l'axe principal : par défaut c'est l'axe Z (c'est-à-dire que les directions testées sont toutes contenues dans le plan XY).

Pas angulaire

Le pas angulaire a été défini dans le chapitre 4. Il est réglé par défaut à 5 degrés. Il doit être modifié si des composants sont insérés avec des angles qui ne sont pas multiples de 5 degrés.

Pas des déplacements

La valeur des pas de déplacements a aussi été définie dans le chapitre 4. L'utilisateur peut :

- Soit le laisser à sa valeur par défaut;
- Soit lui donner une valeur fixe, donnée dans la même unité que le reste du dessin,
- Soit la régler à un certain rapport de :
  - La largeur minimale des solides du dessin avec AutoCAD
  - La dimension du dessin avec CATIA

S'il n'y a pas de solide spécialement fin (plaque...) dans l'assemblage, il est possible d'augmenter le pas élémentaire, et les calculs seront beaucoup plus rapides. Par contre, il doit être diminué en cas de présence de solides très fins.

6-4.2 Paramètres concernant la génération des séquences

La feuille qui s'affiche pour modifier ces paramètres est aussi dans l'annexe 3. Elle n'est aussi accessible qu'au tout début de l'étude.

Séquence linéaires ou non

Par défaut le programme ne génèrera que des séquences linéaires. S'il n'en existe pas (nous en avons vu un exemple dans le chapitre 3) alors l'utilisateur devra générer toutes les séquences possibles.

Débuter par le solide le plus volumineux

Par défaut cette contrainte n'est pas activée. Si l'utilisateur décide de l'activer, seules les séquences débutant par l'élément le plus volumineux seront générées.

Interdire l'assemblage de composants depuis Z-

Par défaut cette contrainte n'est pas non plus activée. Si l'utilisateur décide de l'activer, seules les séquences dont aucun solide n'est assemblé depuis la direction Z- seront générées.

6-4.3 Paramètres concernant l'évaluation des séquences

La fenêtre affichée pour modifier ces données se trouve aussi dans l'annexe 3.

Coefficients de pondération des notes

Ces coefficients correspondent aux paramètres α, β, γ, δ et λ présentés dans le chapitre 5. Ils permettent d'augmenter ou de diminuer l'impact de certaines notes dans le classement général des séquences.

6-5 Outils supplémentaires

D'autres outils ont été programmés pour simplifier l'utilisation du logiciel. Rappelons que le but de Polyassemblage est de permettre de trouver la meilleure séquence d'assemblage. Tous les moyens doivent être mis en œuvre pour rendre cette opération la plus rapide et la plus facile possible pour l'utilisateur.

6-5.1 Renommer les solides dans AutoCAD

AutoCAD ne permet pas d'attribuer des noms aux solides, à moins de passer par des blocs, des groupes ou des calques mais ces possibilités ne sont pas standardisées.

Polyassemblage permet donc à l'utilisateur de renommer chaque solide. Ainsi il pourra mieux comprendre la description des séquences et des états. Sinon par défaut les solides sont nommés « *Solide 1* », « *Solide 2* », ..., « *Solide n* ».

Dans CATIA les solides sont déjà nommés : Polyassemblage est capable de récupérer leurs noms et de travailler avec.

6-5.2 Sauvegarde des résultats

Pour ne pas perdre le travail effectué, le programme est capable de sauvegarder les résultats. L'extraction des données depuis AutoCAD ou CATIA est une opération longue. Il est donc très utile d'enregistrer ces données après la première utilisation.

L'enregistrement se fait dans un fichier binaire, donc plus compact qu'un fichier texte. Il enregistre l'ensemble des informations nécessaires pour reproduire le document de manière identique (mêmes séquences, mêmes critères actifs, même classement) :

Matrices de contact et de translation

Le programme enregistre les matrices de contact et de translation plutôt que la liste des séquences générées. Le désavantage est qu'il faut recalculer ces séquences à chaque fois qu'on ouvre le document (l'opération peut prendre un peu de temps). Mais les données occupent moins de place dans le fichier de sauvegarde.

Nom des solides

Le programme enregistre aussi le nom de tous les solides de l'assemblage.

Liste des directions d'assemblage

Chaque direction d'assemblage est enregistrée comme expliqué dans le chapitre 4, sous forme de trois données de type *double* qui définissent le vecteur directeur de la direction.

Liste des filtres actifs, liste des séquences et des états interdits

Le programme écrit dans le fichier de sauvegarde les numéros des filtres actifs, ainsi que les numéros des séquences et des états interdits par l'utilisateur manuellement.

Volume des solides

Ces donnée permettent de recalculer les deux critères d'évaluation qui s'y rapportent : « Débuter par l'élément le plus gros volumineux » et la note relative à la taille du premier élément.

Paramètres pour l'évaluation des séquences

Le programme enregistre les différents coefficients de pondération α, β, γ, δ et λ.

Dimension maximale

Le programme enregistre la dimension maximale du dessin pour éviter de la redemander à l'utilisateur s'il interagit avec CATIA.

6-5.3 Ouverture d'une étude existante

Il est possible d'ouvrir n'importe quelle étude sauvegardée. Le programme sera capable de générer à nouveau toutes les séquences actives et de reproduire le document comme l'utilisateur l'avait laissé au moment de l'enregistrement.

Par contre pour pouvoir visualiser les séquences et les états, l'utilisateur doit penser à ouvrir à nouveau le dessin d'origine dans le logiciel de CAO.

6-5.4 Impression

Il est possible d'imprimer le *And/Or Graph* ainsi que la liste des séquences si l'utilisateur le souhaite (voir l'annexe 2).

6-6 Conclusion

Nous avons vu dans ce chapitre des outils qui permettent d'aider l'utilisateur de Polyassemblage à identifier les meilleures séquences :

- La visualisation des résultats est la plus claire et la plus complète possible avec le double affichage des séquences et des états;
- Les paramètres réglables permettent d'accélérer la recherche ou de rendre les résultats plus fiables si leurs valeurs par défaut ne sont pas satisfaisantes;
- Les outils de visualisation et d'autres outils propres au programme ont été aussi présentés.

Pour finir nous allons tester le programme avec plusieurs exemples et analyser les résultats.

CHAPITRE 7 - RÉSULTATS

7-1 Introduction

Nous allons présenter dans ce chapitre différents exemples testés avec Polyassemblage :

- Terminer l'étude de l'exemple simple vu dans les derniers chapitres;
- Un alternateur modélisé dans CATIA;
- Un système de freins de vélo modélisé dans AutoCAD.

Tous les calculs ont été faits sur un Pentium III à 700MHz avec 256 Mo de RAM et munis de Windows 2000 Professional.

7-2 Comparaisons CATIA – AutoCAD sur un exemple simple

Dans cette première partie nous allons détailler les résultats obtenus avec l'assemblage présenté tout au long du projet :

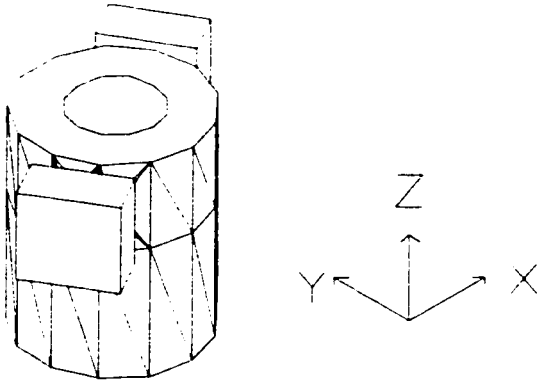


Figure 7.1 : Assemblage testé par Polyassemblage

7-2.1 Acquisition des données

Les pas des déplacements ont été laissés à leurs valeurs par défaut : un dixième de la largeur du solide le plus fin avec AutoCAD et un cinquantième de la taille maximale du dessin avec CATIA.

Avec AutoCAD

Le solide le plus fin est la tige avec une hauteur de 10mm. Les pas des déplacements sont donc de 1mm. Les calculs ont duré environ 22s. Ce temps se répartit comme suit :

- Chargement des données initiales : 0.5s. Les opérations faites pendant cette étape sont :
  - Identification des éléments solides du dessin;
  - Calculs de leur volume;
  - Calculs de leurs dimensions limites;
  - Test d'interférence initial (pour vérifier qu'il n'y a pas d'interférence dans le dessin de départ).
- Calcul des matrices de contact et de translation : 22s. Ceci comprend :
  - 449 tests d'interférence dont 28 se sont révélés positifs;
  - 499 déplacements.

Avec CATIA

La dimension maximale du dessin est de 60mm. Les pas des déplacements sont donc 1.2mm. Le temps total de cette partie a été environ de 278s (4.5min). Ce temps se répartit comme suit :

- Chargement des données initiales : 2s. Les opérations faites pendant cette étape sont :
  - Création des paires de composants;
  - Création des objets de type *collision* pour chaque paire;
  - Calcul du volume des solides;
  - Lecture du nom des solides;
  - Test d'interférence initial et calcul de la matrice de contact.
- Calcul des matrices de translation : 275s.

- 1512 tests d'interférence dont 26 se sont révélés positifs;
- 1464 déplacements.

Les temps de calcul avec CATIA sont beaucoup plus longs pour deux raisons :

- Plus de tests sont nécessaires car il est impossible d'envelopper les solides dans des espaces restreints comme avec AutoCAD (les *Bounding Box*);
- Les déplacements de solides, tests de collision, etc. sont toujours plus lents avec CATIA.

Résultats

Les résultats ont déjà été décrits dans les chapitres précédents.

7-2.2 Génération des séquences

Dans cette partie et les parties suivantes les résultats sont identiques entre CATIA et AutoCAD car Polyassemblage n'interagit plus avec les logiciels de CAO. Nous n'avons généré que les séquences linéaires : le programme en a trouvé 8. La génération des séquences a été quasiment instantanée.

Les séquences trouvées sont montrées ci-dessous, dans l'ordre proposé par Polyassemblage :

Séquence 1 :

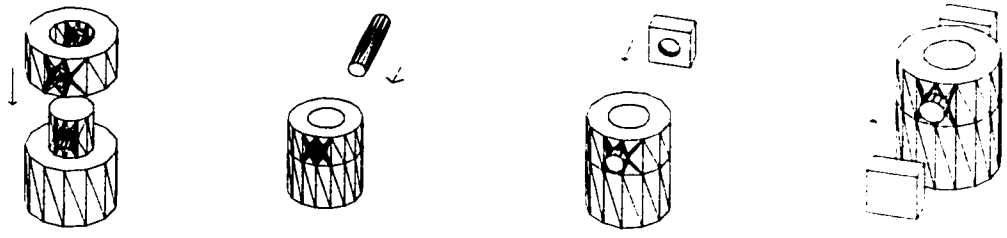


Figure 7.2 : Séquence classée numéro 1

Séquence 2 :

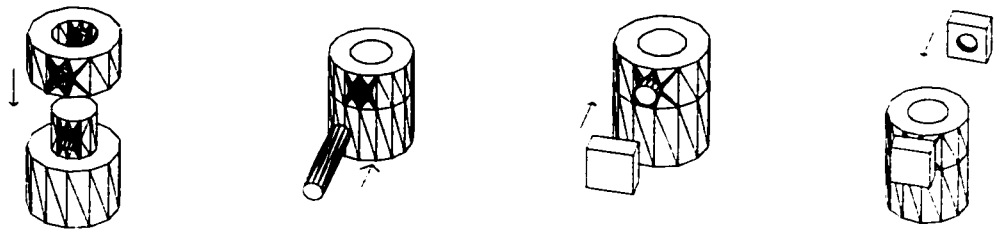


Figure 7.3 : Séquence classée numéro 1 ex-aequo

Séquence 3 :

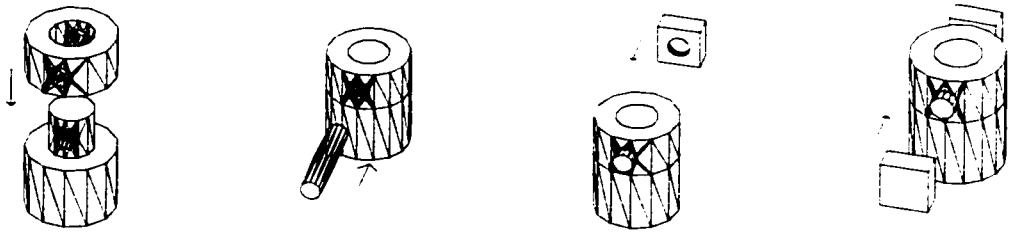


Figure 7.4 : Séquence classée numéro 3

Séquence 4 :

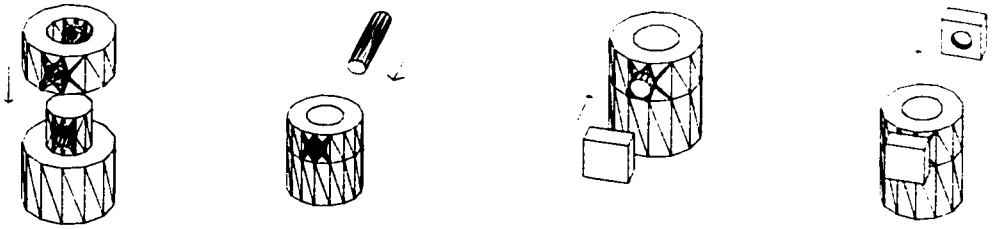


Figure 7.5 : Séquence classée numéro 3 ex-aequo

Séquence 5 :

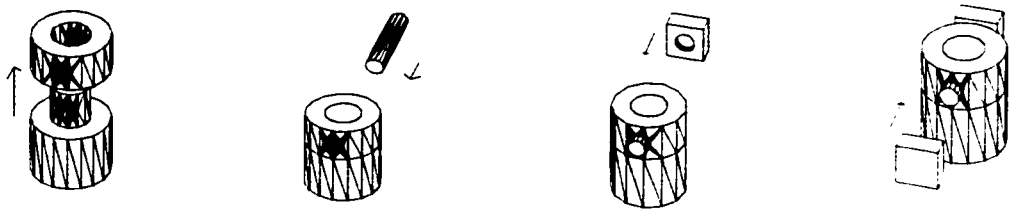


Figure 7.6 : Séquence classée numéro 5

Séquence 6 :

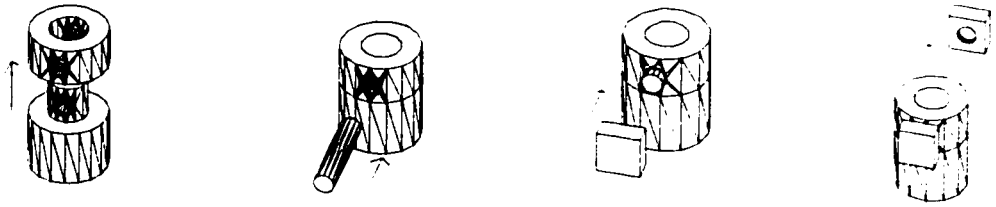


Figure 7.7 : Séquence classée numéro 5 ex-aequo

Séquence 7 :

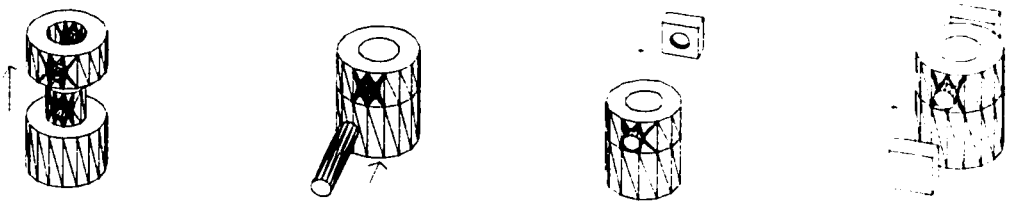


Figure 7.8 : Séquence classée numéro 7

Séquence 8 :

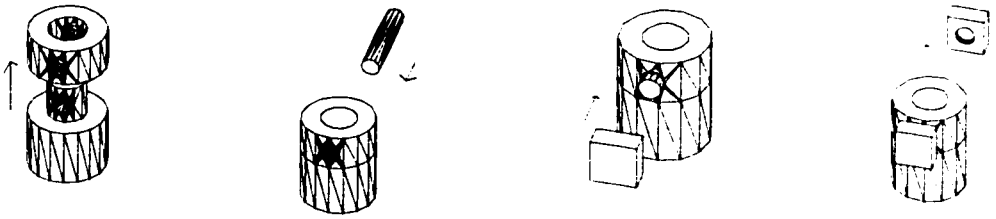


Figure 7.9 : Séquence classée numéro 7 ex-aequo

Enfin, le *AND/Or Graph* de l'assemblage est le suivant :

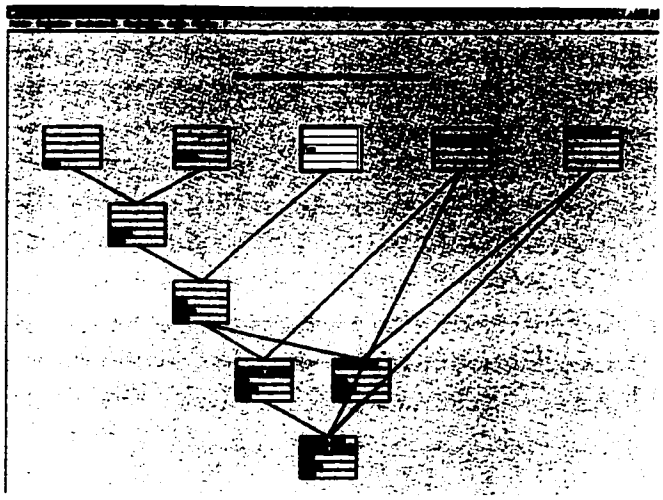


Figure 7.10 : *And/Or Graph* pour les séquences linéaires de l'assemblage

7-2.3 Évaluation des séquences

La valeur des notes pour les 8 séquences est indiquée dans le tableau ci-dessous :

Tableau 7.1 : Notes attribuées aux séquences trouvées

Séquence	Taille du 1er élément	Nbre de directions identiques	Nbre de sous-assemblages	Stabilité	Regroupement des directions identiques	Moyenne
1	10	10	10	10	10	10
2	10	10	10	10	10	10
3	10	10	10	10	8	9.6
4	10	10	10	10	8	9.6
5	4	10	10	10	10	8.8
6	4	10	10	10	10	8.8
7	4	10	10	10	8	8.4
8	4	10	10	10	8	8.4

Toutes les séquences ont le même nombre de directions d'insertion, la même stabilité et bien sûr le même degré de linéarité.

Seuls deux critères nous permettent de différentier ces séquences :

- Volume du premier élément : *Base* a un volume de 42462mm^3 et celui de *Dessus* n'est que de 17252mm^3. Les 4 séquences qui commencent avec *Base* sont donc meilleures pour ce critère.
- Le facteur « regroupement des directions identiques » permet d'identifier deux séquences parmi les 4. Ces deux dernières séquences sont d'ailleurs identiques car l'assemblage est symétrique.

7-3 Étude de l'alternateur

7-3.1 Description de l'alternateur

L'alternateur est constitué de 9 composants principaux et 4 vis. L'axe de l'alternateur est l'axe X (par construction).

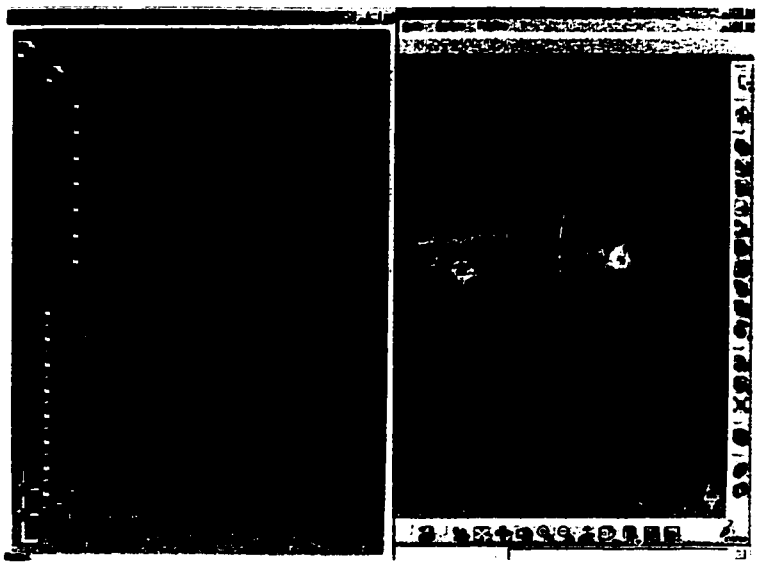
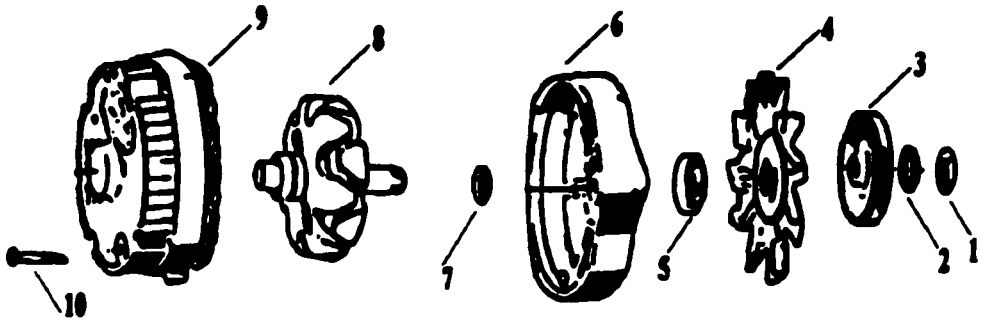


Figure 7.11 : Alternateur



1)	Écrou	6)	Ensemble frontal
2)	Rondelle	7)	Bague 1
3)	Poulie	8)	Rotor
4)	Ventilateur	9)	Couvercle postérieur
5)	Bague 2	10)	Vis (4)

Figure 7.12 : Première décomposition de l'alternateur, extrait du mémoire de Aguilar (1996)

L'ensemble frontal est lui même constitué d'autres composants mais Polyassemblage les considère chacun comme un unique élément, comme expliqué dans le chapitre 5.

Enfin 4 vis servent à assembler le couvercle postérieur et l'ensemble frontal. Elles ne sont pas modélisées pour Polyassemblage, car il est évident qu'elles seront ajoutées directement après le composant qu'elles sont censées tenir. Il est inutile donc d'alourdir les calculs pour elles.

7-3.2 Acquisition des données

Le pas des déplacements élémentaires a dû être réduit à 3mm. Sa valeur originale était de 168mm/50 = 3.36mm et cette dimension était supérieure à l'épaisseur de la plaque du ventilateur (3mm). Le programme aurait pu ne pas détecter certaines collisions.

Comme toujours avec CATIA les calculs ont été relativement longs : 9min et 7s.

Le programme a pu désassembler tous les composants en ne considérant que les directions X+ et X-. Pour chaque paire de composants les matrices de translation ne contiennent donc que deux valeurs : la possibilité de désassemblage selon X+ et selon X-.

7-3.3 Génération des séquences

Seules les séquences linéaires ont été générées. Le programme en a trouvé 256. Le temps nécessaire pour générer toutes ces séquences est négligeable.

Le *And/Or Graph* de l'alternateur pour ces séquences est le suivant :

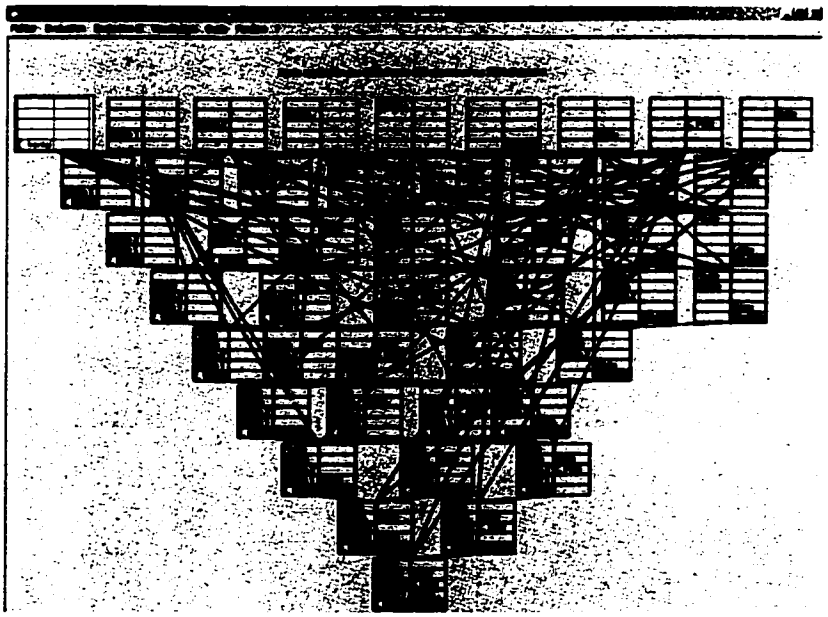


Figure 7.13 : *And/Or Graph* pour toutes les séquences linéaires de l'alternateur

7-3.4 Évaluation des séquences

Pour nous la première étape est de réduire le nombre de séquences trouvées :

- Il est inutile d'interdire l'assemblage de composants depuis le bas : tous sont déjà insérés selon l'axe X.

- Nous sommes obligés de terminer l'assemblage par le couvercle postérieur, car sinon il serait impossible de bloquer le rotor pendant le vissage de l'écrou : il reste 128 séquences.
- Nous ne souhaitons pas assembler la poulie, le ventilateur etc. si le rotor et l'ensemble frontal ne sont pas déjà assemblés. Nous ne gardons que les états qui contiennent ces deux composants lorsque c'est possible : il reste 4 séquences.
- Parmi les 4 séquences deux commencent avec la bague 1. Ce composant est trop petit pour débuter l'assemblage, nous supprimons donc ces séquences.

Voici le *And/Or Graph* de l'alternateur lorsqu'il ne reste que ces 2 séquences :

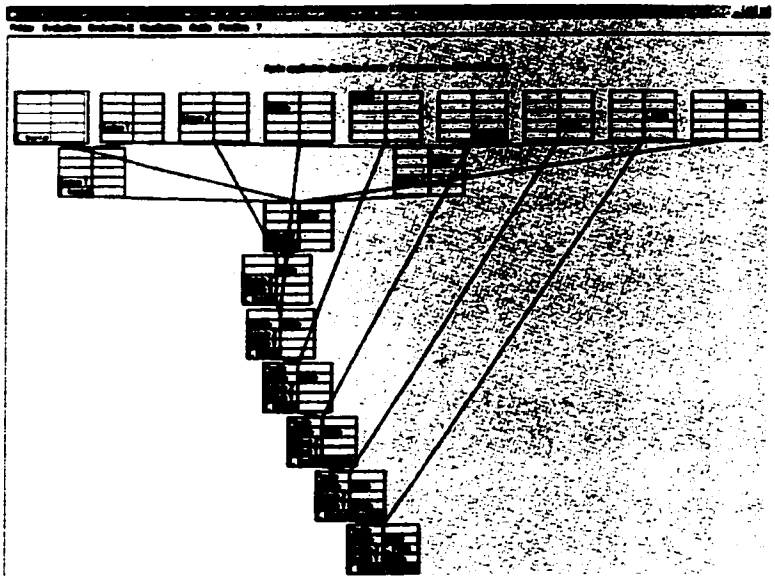


Figure 7.14 : *And/Or Graph* après l'activation des critères d'évaluation pour l'alternateur

Maintenant il ne nous reste plus qu'à choisir entre les 2 séquences restantes.

Séquence 1 :

Rotor – Bague 1 – Ensemble frontal – Ventilateur – Poulie – Rondelle – Écrou – Couvercle postérieur

Séquence 2 :

Ensemble frontal – Bague 1 – Rotor – Ventilateur – Poulie – Rondelle – Écrou – Couvercle postérieur

Les notes pour ces deux séquences sont :

Tableau 7.2 : Notes attribuées aux deux séquences finales de l'alternateur

Séquence	Taille du 1er élément	Nbre de directions identiques	Nbre de sous-assemblages	Stabilité	Regroupement des directions identiques	Moyenne
1	10	5	10	9	5	7.8
2	5	5	10	10	3	6.6

Ces deux séquences se ressemblent beaucoup. Après les avoir visualisées, notre choix se porterait sur la première séquence, choix qui est aussi celui de Polyassemblage.

7-4 Étude du système de freins

Notre troisième exemple est un système de freins de vélo. Cet assemblage a été modélisé avec AutoCAD 2000.

7-4.1 Description et fonctionnement du système de freins

Description

Le système de freins est montré sur la figure ci-dessous.

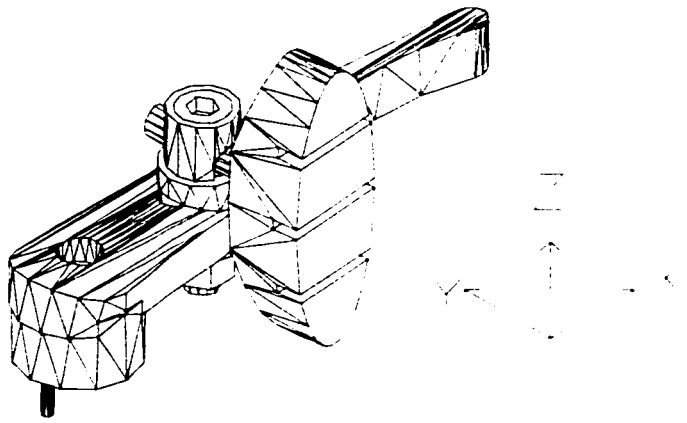
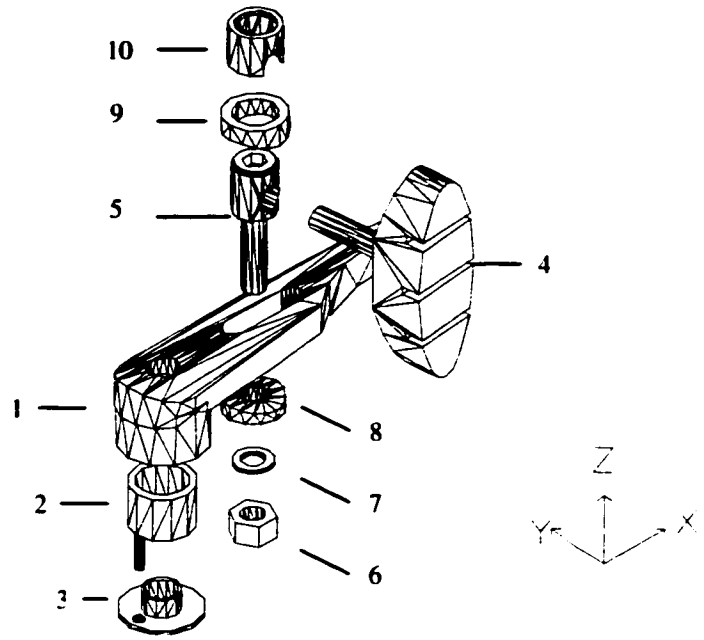


Figure 7.15 : Système de freins

L'assemblage est constitué de 10 composants.



1)	Support principal	6)	Écrou
2)	Ressort	7)	Rondelle
3)	Cache du ressort	8)	Bague inférieure
4)	Plaquette	9)	Bague supérieure
5)	Tige	10)	Cache de la tige

Figure 7.16 : Système de frein décomposé

L'axe d'insertion de la plaquette est dans le plan XY et forme avec l'axe Y un angle de 5 degrés. Tous les autres composants s'insèrent selon la direction Z. L'axe X est l'axe principal du support principal (le composant 1).

Fonctionnement général

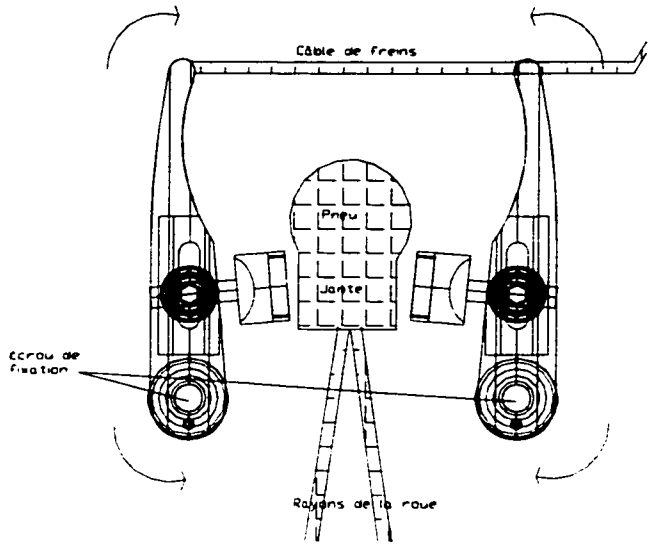


Figure 7.17 : Vue de profil de la roue avec le système de freins fonctionnel

Lorsque le câble de freins est tiré, le haut des deux systèmes de frein se déplace vers le centre (flèches du haut). Comme la partie inférieure du système est fixée, les deux plaquettes de freins entrent en contact avec la jante de la roue, ce qui la freine. Lorsque le câble est relâché, le ressort (composant 2) du système replace le système dans sa position initiale (flèches du bas).

7-4.2 Acquisition des données

Le pas des déplacements a été laissé à sa valeur par défauts : un dixième de l'épaisseur de la rondelle, soit 0.1mm. Il aurait pu sans problème être changé à 1mm, mais même avec un pas de 0.1mm les calculs ne sont pas trop longs (62s, soit 1min).

7-4.3 Génération des séquences

La génération des séquences d'assemblage a été plus longue. Nous avons dû nous restreindre aux assemblages linéaires commençant par l'élément le plus volumineux (l'élément *support*). Finalement les calculs ont duré 4min et 12s.

Le *And/Or Graph* pour les 9792 séquences trouvées est :

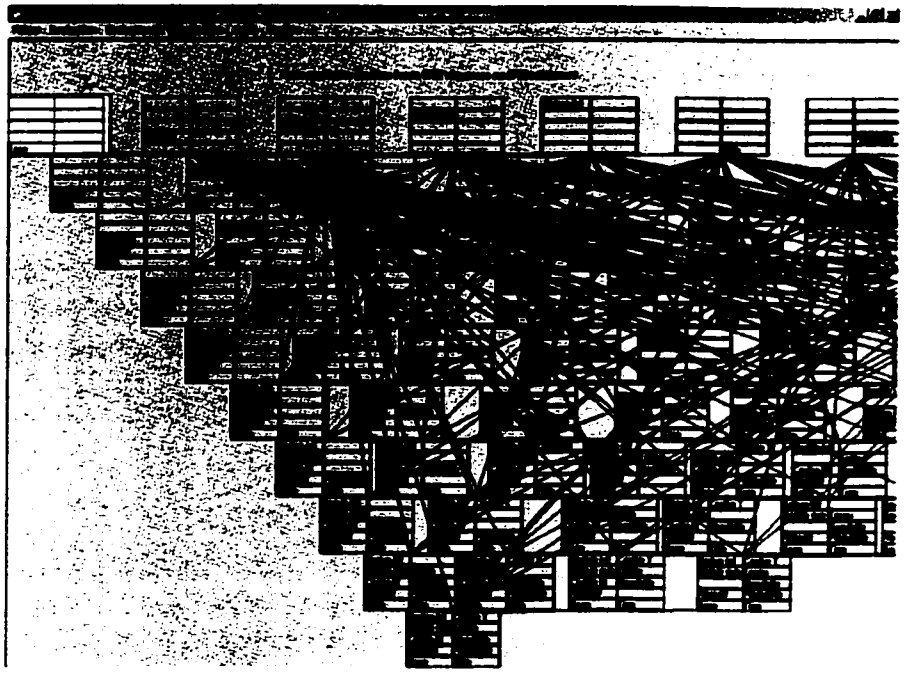


Figure 7.18 : And/Or Graph pour toutes les séquences linéaires du système de freins

Les états sont trop nombreux. Ils ne rentrent pas tous sur l'écran. Notre but est donc maintenant de réduire le nombre de séquences pour arriver à éliminer celles qui ne sont pas satisfaisantes.

7-4.4 Évaluation des séquences

L'attribution des notes a pris 1min et 40s.

Les critères « débuter par l'élément le plus volumineux » et « minimum de sous-assemblages » ne sont pas utilisables car ils ont déjà été activés pour la génération des séquences. Nous ne pouvons pas non plus interdire l'insertion de composants depuis le bas car plusieurs composants ne peuvent être insérés que dans cette direction.

Activer le critère « minimum de directions d'assemblage » permet d'enlever toutes les séquences pour lesquelles des composants sont ajoutés depuis Y+ (la direction opposée de la plaquette) : il reste 7056 séquences.

Seulement 4 séquences ont 10/10 pour les deux derniers critères « stabilité des sous-assemblages » et « regroupement des directions d'insertion ». Le *And/Or Graph* pour ces 4 séquences est :

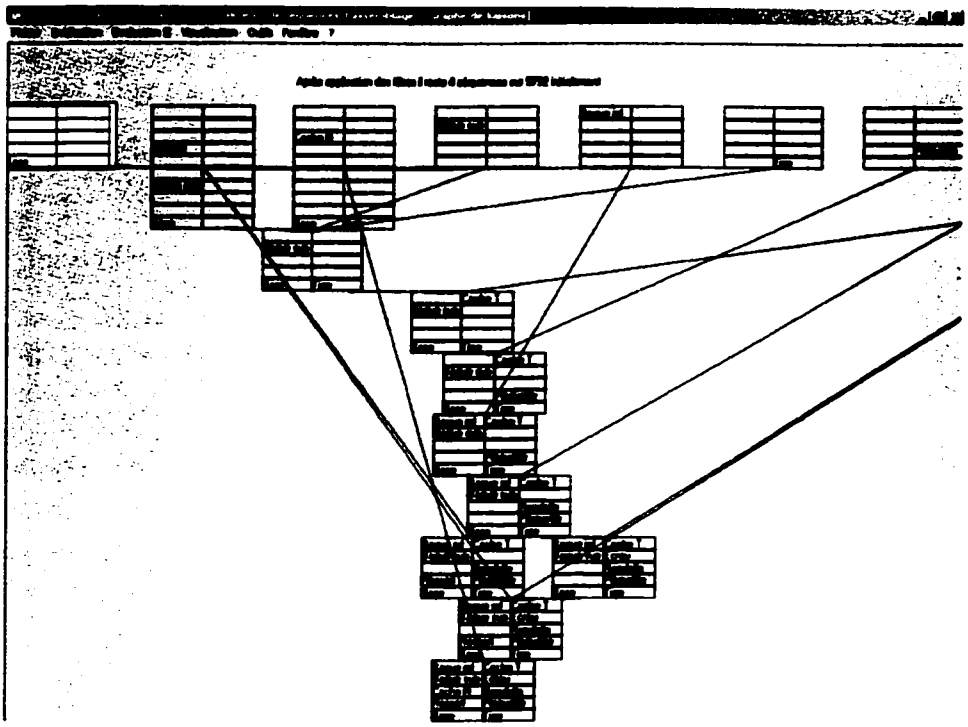


Figure 7.19 : *And/Or Graph* du système de frein pour 4 séquences finales

Sur cette figure, on ne voit pas le graphe en entier car il ne rentre pas à l'écran. C'est le choix que j'ai effectué en développant Polyassemblage. Je préfère que ce soit ainsi plutôt que le *And/Or Graph* soit illisible.

Ces 4 séquences sont 4 variantes de la séquence suivante :

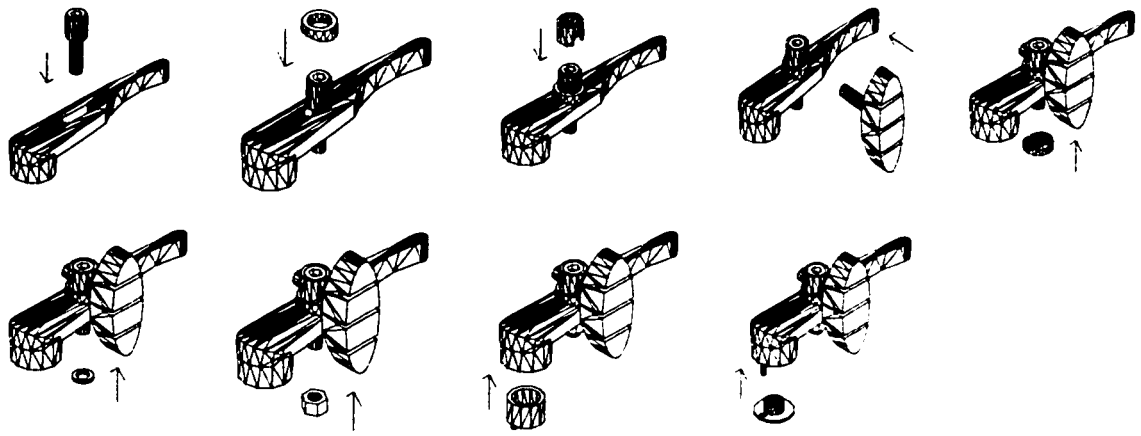


Figure 7.20 : Séquence d'assemblage proposée pour le système de freins

Variante 1 :

Il est possible d'insérer la bague supérieure avant ou après la tige (intervertir les opérations 1 et 2). Nous choisissons ce second cas car la bague supérieure peut glisser facilement sur le support si elle n'est pas tenue par la tige.

Variante 2 :

Il est possible de regrouper l'assemblage de la rondelle et de l'écrou d'une part et du ressort et de son cache d'autre part ou bien d'alterner les opérations 7 et 8. Nous préférons la première solution. La séquence choisie finalement sera la séquence de la figure 7.20.

Autres possibilités :

Des séquences qui n'ont pas 10 à tous les critères peuvent aussi se révéler très bonnes. Par exemple la séquence classée 23 termine l'assemblage par les deux caches, ce qui est souvent le cas de tels produits dans la pratique.

7-5 Conclusion

Les exemples montrés dans ce chapitre illustrent le fonctionnement de Polyassemblage. Ils en montrent aussi les limites. Les temps de calculs deviennent longs dès qu'il existe plusieurs

milliers de séquences. Pourtant des assemblages comme le système de frein restent des petits assemblages : à peine une dizaine de composant.

Dans des versions futures de Polyassemblage il sera intéressant de réduire encore le nombre de séquences générées. On pourrait par exemple :

- Tenir compte des connecteurs (comme indiqué dans le chapitre 1) pour ne jamais perdre du temps avec les vis et les composants de ce type.
- Introduire d'autres contraintes dès la génération des séquences d'assemblage pour éviter l'explosion du nombre de séquences.

CONCLUSION

Nous avons présenté dans ce rapport le logiciel Polyassemblage. Ce programme est capable de générer et d'évaluer toutes les séquences d'assemblage d'un produit, et ce à partir uniquement de sa modélisation solide dans AutoCAD ou CATIA.

Après avoir identifié les principaux courants de recherche dans ce domaine, nous avons expliqué les bases du programme :

D'abord nous avons utilisé les interfaces d'automatisation proposées dans les dernières versions d'AutoCAD et CATIA sous Windows pour obtenir toutes les données nécessaires à la génération des séquences d'assemblage. Le programme a été testé avec CATIA V5R8 et AutoCAD 2000.

Ensuite les séquences d'assemblage ont été générées avec un algorithme développé initialement en 1995, et qui a été modifié et amélioré pour ce projet.

Polyassemblage est aussi capable d'évaluer l'ensemble des séquences trouvées avec des critères basés sur les informations contenues dans le dessin.

Enfin les autres outils programmés dans Polyassemblage rendent le programme très convivial, et permettent à l'utilisateur de trouver facilement une bonne séquence, la bonne séquence.

De plus son champ d'application est très étendu. Il peut être utilisé dans le cadre de l'ingénierie simultanée par les concepteurs de nouveaux produits, par le bureau des méthodes ou par les responsables de la production. Il est très simple d'utilisation et donne des résultats en quelques minutes seulement. Il représente un outil efficace pour réduire les coûts d'assemblage présents et futurs.

Enfin Polyassemblage fonctionne avec les logiciels de CAO les plus connus. La principale limite est que la version de CATIA la plus répandue aujourd'hui dans l'industrie est encore la version 4. Il faudra attendre quelques années avant que Polyassemblage puisse être appliqué à la majorité des entreprises.

Par contre Polyassemblage mérite encore d'être développé. Dans plusieurs types d'assemblage assez fréquents, il ne sera pas capable de trouver des séquences d'assemblage. Les conditions pour que le programme fonctionne sont :

- Les composants doivent être indéformables : les ressorts, les clips, les courroies sont hors du champ d'utilisation de Polyassemblage;
- Ils ne doivent pas être tournés pour être assemblés ou désassemblés : avec cette version de Polyassemblage, les pas de vis ne doivent pas être modélisés sur le produit.
- Les seuls chemins de désassemblage ou d'assemblage pris en compte sont rectilignes et unidirectionnels;
- La possibilité d'assembler ou de désassembler un composant doit apparaître dans la géométrie de l'assemblage. Aucune autre contrainte n'est prise en compte.

Dans d'autres cas l'utilisateur de Polyassemblage doit être mis en garde sur d'éventuels problèmes de fiabilité, en particulier s'il y a présence de solides très fins (plaques, etc.) ou de fils.

Ces limites nous amènent à proposer des améliorations qui permettront de disposer d'un logiciel vraiment efficace. Pour la partie « recherche de chemins de désassemblage » nous pourrions essayer de :

- Rechercher des chemins non linéaires;
- Rechercher des manières de désassembler les composants en les tournants (pour les pas de vis);
- Désassembler des composants déformables.

Pour la partie « génération des séquences d'assemblage », la durée de l'opération augmente vite lorsqu'on arrive à plusieurs milliers de séquences d'assemblage. Nous pourrions :

- Éviter d'atteindre ce nombre en appliquant d'autres contraintes d'assemblage dès la génération des séquences (en ce moment la plupart des contraintes ne s'applique qu'après);
- Considérer les connecteurs (vis, rivets...) comme des caractéristiques de l'assemblage, plutôt que comme des composants à part entière (comme expliqué dans le chapitre 1).

C'est la partie « évaluation des séquences trouvée » qui laisse le plus de place aux améliorations. Nous pourrions par exemple :

- Programmer des critères d'évaluation basés sur d'autres renseignements que juste ceux contenus dans le dessin. L'utilisateur devrait alors rentrer ces informations manuellement;
- Intégrer Polyassemblage à des logiciels de calcul des coûts d'assemblage comme celui développé par Boothroyd (1994);
- Augmenter le nombre d'informations extraites du modèle de CATIA. Ce logiciel contient énormément d'informations supplémentaires sur la géométrie du produit qui pourraient être réutilisées pour évaluer les séquences avec d'autres critères.

Dans l'annexe 4, la structure des données utilisées par le programme est expliquée. Ces renseignements seront utiles pour modifier et améliorer le programme tel qu'il existe actuellement.

Néanmoins ses avantages et ses capacités actuelles sont des bases solides pour des développements futurs et une implantation assez rapide en situation industrielle.

RÉFÉRENCES

AGUILAR, J. (1996). Méthode Interactive de Détermination de Gammes en Respectant les Contraintes de Géométrie et de Procédure, *Mémoire de Maîtrise, École Polytechnique de Montréal.*

BOOTHROYD, G (1994). Product Design for Manufacture and Assembly, New York : Marcel Dekker, Inc., New-York.

BOURJAULT, A. (1984). Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires, *Thèse d'État, Université de Franche Comté, Besançon, France.*

CHOI, C. K., ZHA, X. F., NG, T. L. et LAU, W. S. (1998). On the Automatic Generation of Product Assembly Sequence, *International Journal of Production Research*, v 36, n 3, p 617-633.

DE FAZIO, T. L. et WHITNEY, D. E. (1987). Simplified Generation of all Mechanical Assembly Sequences, *IEEE Journal of Robotics and Automation*, p640-658.

DINI, G., FAILLI, F., LAZZERINI, B. et MARCELLONI, F. (1999). Generation of optimized assembly sequences using genetic algorithms, *CIRP Annals - Manufacturing Technology*, v 48, n 1, p 17-20.

ENG T. H., LING Z. K., OLSON W. et McLEAN C. (1999). Feature-based assembly modeling and sequence generation, *Computers and Industrial Engineering*, v 36, n 1, p 17-33.

FOUDA, P., DANLOY, J., L'EGLISE, T., DE LIT, P., REKIEK, B. et DELCHAMBRE, A. (2001). A Heuristic to Generate a Precedence Graph Between Components for a Product

Family, *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, May 28-29 2001, Fukuoka, p43-48.

FUJIMOTO, H. et SEBAALY, M. F. (2000). A New Sequence Evolution Approach to Assembly Planning, *Transactions of the ASME*, v 122.

GHOSH, K. et GOTTIPOLU, R. (1995). An Integrated Approach to the Generation of Assembly Sequences, *International Journal of Computer Applications in Technology*, V8 n3/4, p125-138.

GÜNGÖR, A. et GUPTA, S., (2001). Disassembly Sequence Plan Generation Using a Branch-and-Bound Algorithm, *International Journal of Production Research*, v 39, n 3, p481-509.

GUPTA, S. et KRISHNAN, V., (1998). Product Family-Based Assembly Sequence Design Methodology, *IIE Transactions (Institute of Industrial Engineers)*, v 30, n 10, p 933-945

HOMEM DE MELLO, L. S. et SANDERSON, A. C. (1990). AND/OR Graph Representation of Assembly Plans, *IEEE Transactions on Robotics and Automation*, v6, n2, p188-199.

JONES, R. E., WILSON R. H. et CALTON T. L. (1998). On constraints in assembly planning, *IEEE Transactions on Robotics and Automation*, v 14, n 6, p 849-863.

LANHAM, J. et DIALAMI, F. (2001). The Assembly State vector: A New Approach to the Generation of Assembly Sequences, *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, May 28-29 2001, Fukuoka, p 37-42.

LATINNE, P., DE LIT, P., REKIEK, B. et DELCHAMBRE, A. (2001). Assembly Planning with an Ordering Genetic Algorithm, *International Journal of Production Research*, v 39, n 16, p3623-3640.

LATOMBE, J. C. et WILSON R. (1995). Assembly Sequencing with Toleranced Parts, *Symposium on Solid Modeling and Applications - Proceedings*, May 17-19 1995, Salt Lake City, UT, USA, p 83-94.

MARTINEZ, M. T., FAVREL J. et GHODOUS, P. (2000). Product Family Manufacturing Plan Generation and Classification, *Concurrent Engineering Research and Applications*, v8, n1, p 12-23

MASCLE, C. et BALASOIU, B. A. (2001). Disassembly-Assembly Sequencing Using Feature-Based Life-Cycle Model, *International Symposium on Assembly and Task Planning Soft Research Park*, May 28-29 2001, Fukuoka, p31-36.

OSETROV, V. G. (1998). Calculus of Statements in Machine Assembly Process Planning, *Russian Engineering Research, v 18, n 3, p 44-50.*

O'SHEA, B., KAEBERNICK, H. et GREWAL, S. S. (2000). Using a Cluster Graph Representation of Products for Application in the Disassembly Planning Process, *Concurrent Engineering : Research and Applications*, v8, n3, p 158-170.

ROMNEY, B., GODARD, C., GOLDWASSER, M. et RAMKUMAR, G. (1995). An Efficient System for Geometric Assembly Sequence Generation and Evaluation, http://citeseer.nj.nec.com/romney95efficient.html.

SARVANANTHAN, V. C. et PANDEY, P. C. (1999). An Automated Assembly Sequence Planning System for Mechanical Parts, *Electronic Journal of the School of Advanced Technologies, Asian Institute of Technology*, 1, http://www.sat.ait.ac.th/ej-sat/articles/1.1/vcs-full.html.

SENIN, N., GROPPETTI, R. et WALLACE, D. (2000). Concurrent Assembly Planning with Genetic Algorithm, *Robotics and Computer Integrated Manufacturing*, v16, n1, p 65-72.

SMITH, S. et SMITH, G (2001). Automatic Stable Assembly Sequence Generation and Evaluation, *Journal of Manufacturing System*, v 20 n 4.

TSENG, H. E. et LI, R. K. (1999). Novel Means of Generating Assembly Sequences Using the Connector Concept, *Journal of Intelligent Manufacturing*, v10, n5, p 423-435.

VAN HOLLAND, W. et BRONSVOORT, W (1996). Assembly Features and Sequence Planning, http://www.cg.its.tudelft.nl/wwwcc/papers/vanHolland.ifip96.pdf.

WOLTER, J. et CHAKRABARTY, S. (1997). Structure-Oriented Approach to Assembly Sequence Planning, *IEEE Transactions on Robotics and Automation*, v 13, n 1, p 14-29

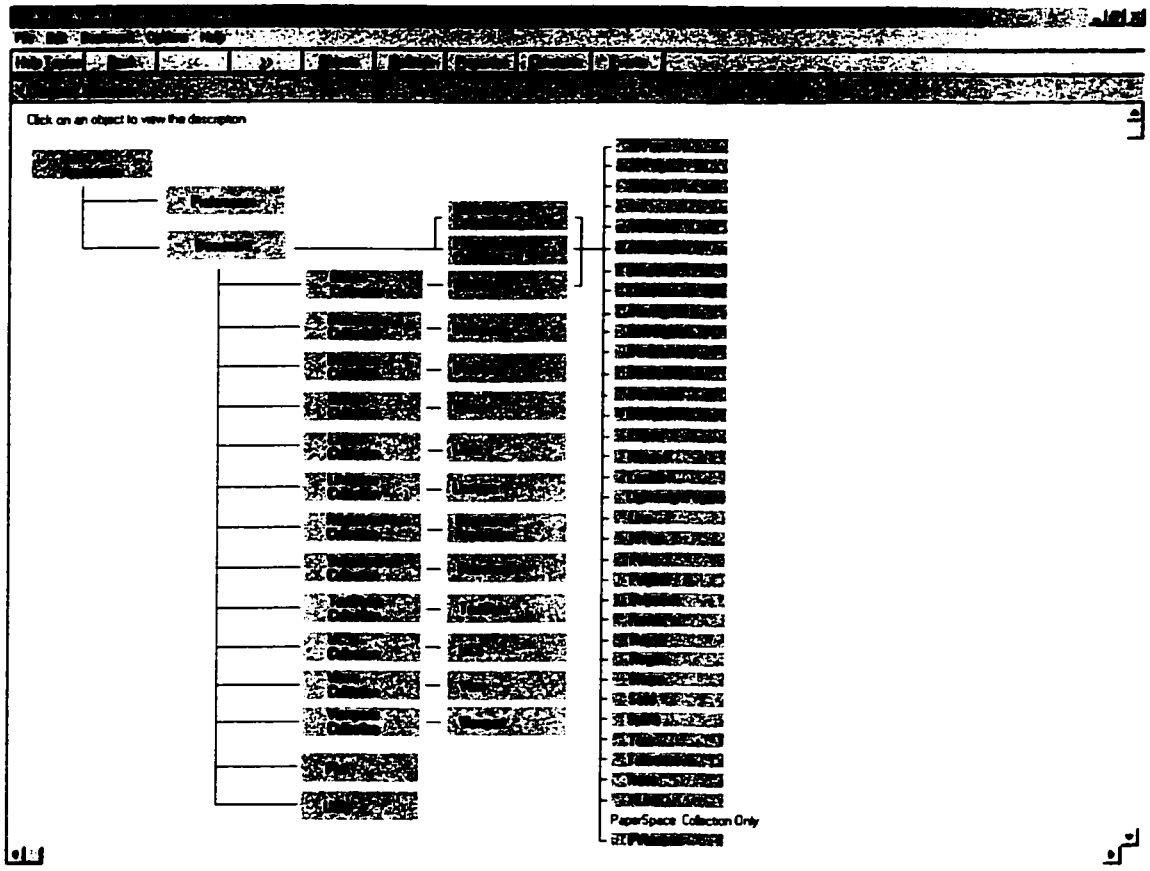
YEE, S. T. et VENTURA, J. (1999). A Petri Net Model to Determine Optimal Assembly Sequences with Assembly Operation Constraints, *Journal of Manufacturing Systems*, v 18 n 3.

ZHA, X. F., LIM, S. Y. E. et FOK, S. C. (1998). Integrated Intelligent Design and Assembly Planning: A Survey, *International Journal of Advanced Manufacturing Technology*, v 14, n 9, p 664-685.

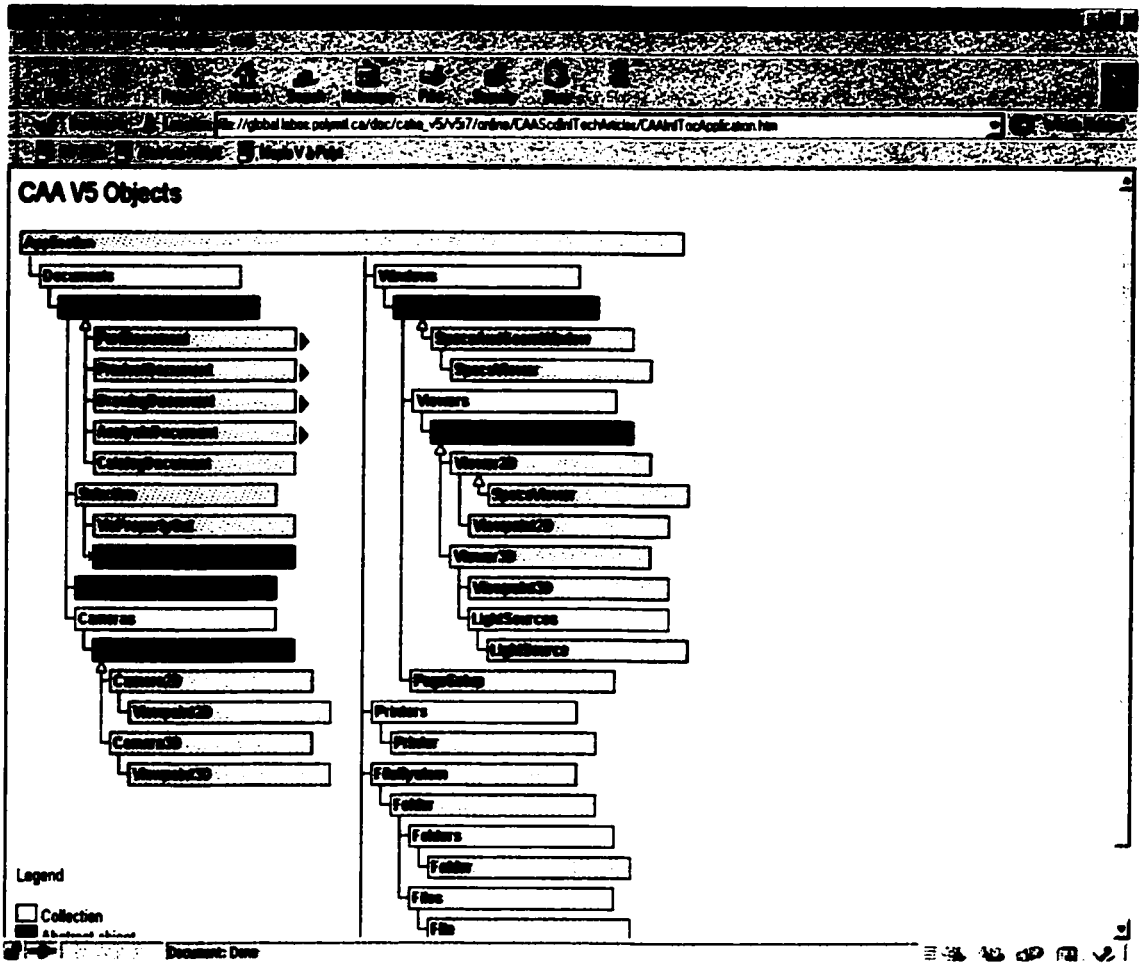
ZHAO, J. et MASOOD, S. (1999). Intelligent Computer-Aided Assembly Process Planning System, *International Journal of Advanced Manufacturing Technology*, v 15, n 5, p 332-337.

ANNEXE 1 : HIÉRARCHIE DES OBJETS

AutoCAD

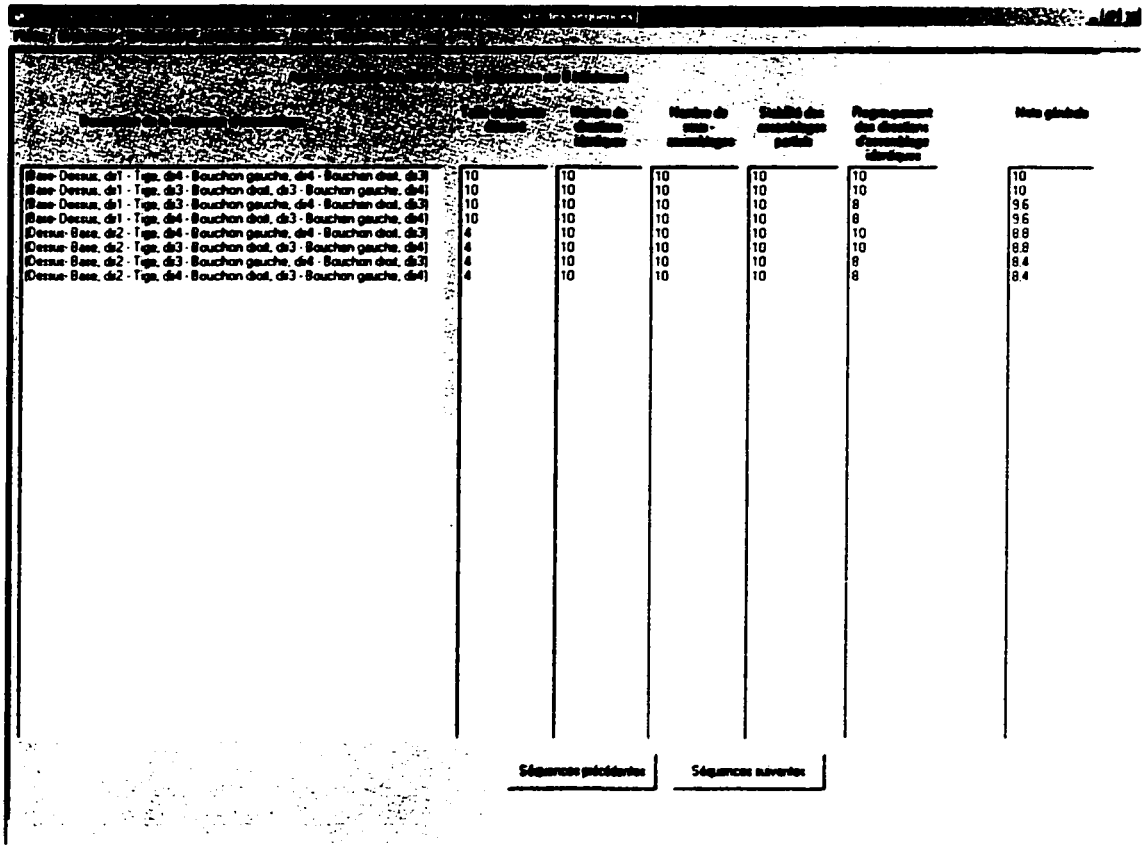


CATIA

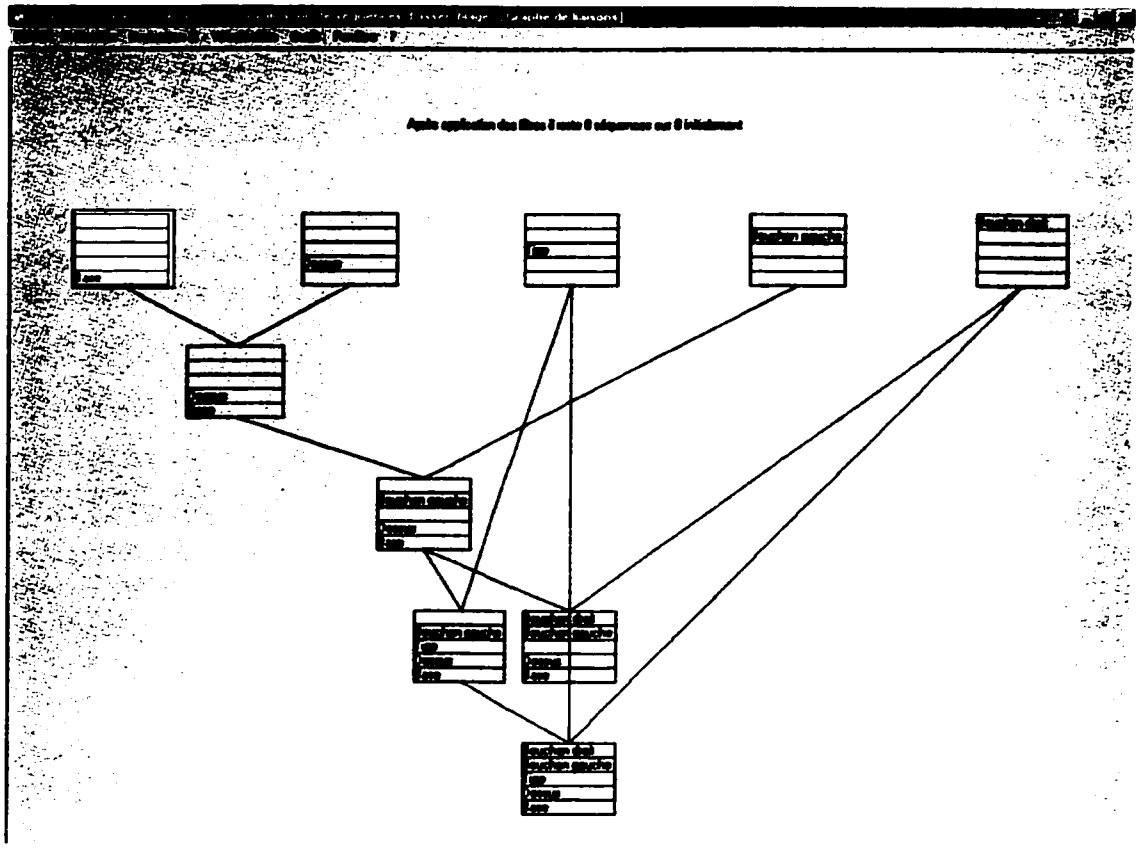

CAA V5 Objects
Application
Documents
Windows
Viewers
Cameras
LightSources
Printers
Printer
FileSystem
Folder
Folders
Files
File
Legend
Collection
Document: Done

ANNEXE 2: AFFICHAGE DES RÉSULTATS

Fenêtre des séquences

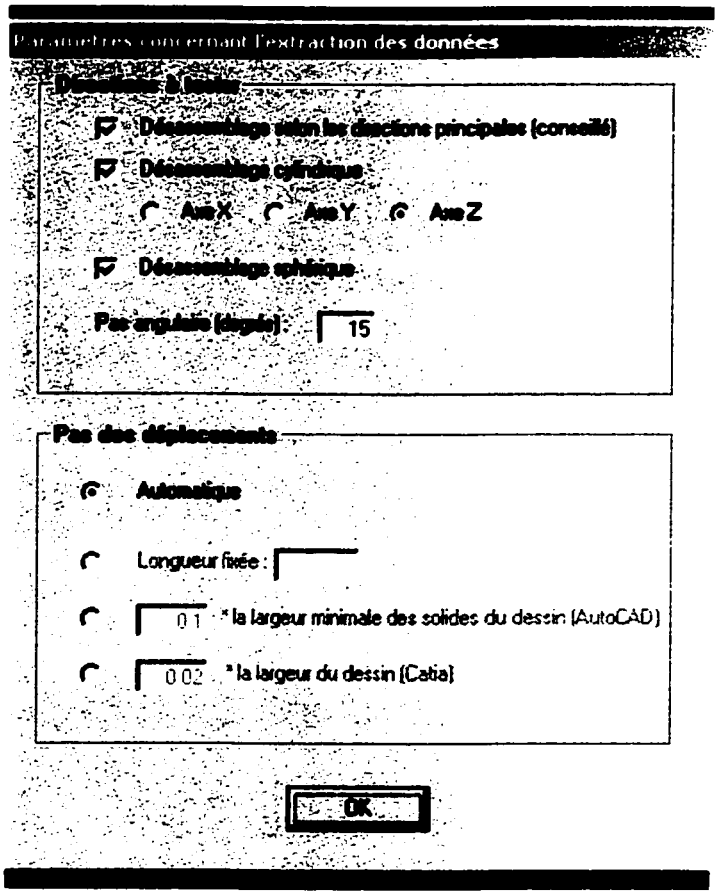


Fenêtre des états

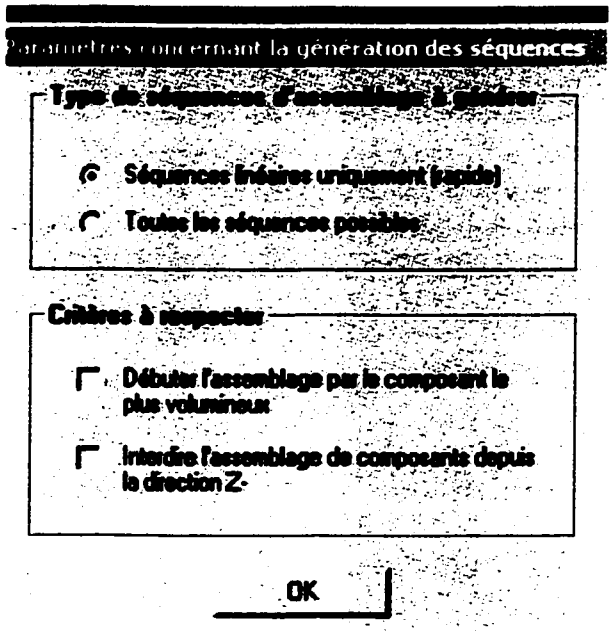


ANNEXE 3 : FENÊTRES POUR MODIFIER LES PARAMÈTRES

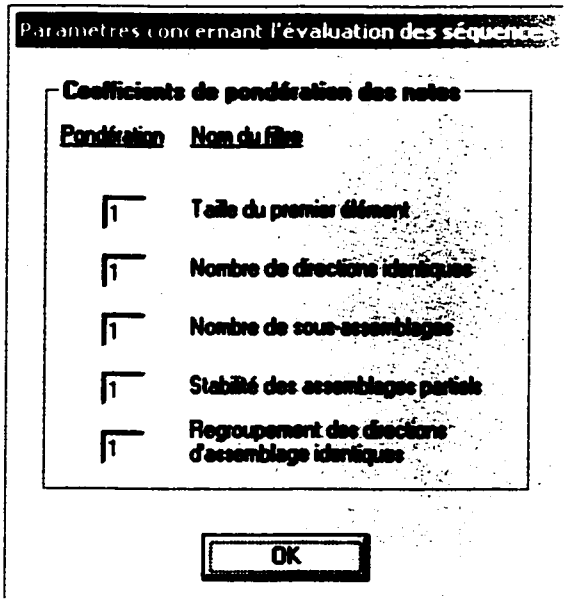
Paramètres pour l'extraction de données



Paramètres pour la génération des séquences



Paramètres pour l'évaluation des séquences



ANNEXE 4 : STRUCTURE DES DONNÉES DU PROGRAMME

Nous allons décrire dans cette annexe comment sont modélisées les données à l'intérieur du programme. Seulement les données principales sont décrites et à travers elles on peut comprendre comment est organisé le programme.

Ces données concernent les trois étapes principales du logiciel : la création des matrices de contact et de translation, la génération des séquences d'assemblage et leur évaluation.

Données relatives à la géométrie de l'assemblage

Éléments du dessin

Les solides du dessin sont accessibles par la collection nommée *dessin_assemblage*. Le programme utilise cette collection pour les déplacer, obtenir leurs propriétés et les modifier. Nous avons indiqué dans le chapitre 2 comment l'obtenir.

Avec AutoCAD

Les solides sont numérotés de *1* à *n_solides* (*n_solides* est le nombre de solides). Le dessin peut contenir aussi des éléments non solides. La collection *solide_acad* permet de pointer les éléments solides uniquement : *dessin_assemblage(solide_acad(1))* représente le solide le plus ancien de la base de données et *dessin_assemblage(solide_acad(n_solides))* représente le plus récent.

Avec CATIA

Les solides sont numérotés de *1* à *n_solides* : *dessin_assemblage.Item(1)* représente le solide qui se trouve en haut de l'arbre de la structure du produit et *dessin_assemblage.Item(n_solides)* représente celui qui est en bas.

Caractéristique des éléments

Enveloppe des solides (seulement pour AutoCAD)

Déclaration : *Public dim_ext As New Collection*

Rôle :

Chaque élément de la collection *dim_ext* contient les points inférieurs et supérieurs de la boîte qui enveloppe le solide, comme indiqué dans le chapitre 4.

Exemple :

Dim_ext(1) représente les dimensions extérieures du solide *dessin_assemblage(solide_acad(1))*. *Dim_ext(n_solides)* représente les dimensions extérieures du solide *dessin_assemblage(solide_acad(n_solides))*.

Volume des solides

Déclaration : *Public volume_solide As New Collection*

Rôle :

Chaque élément de la collection *volume_solide* contient le volume du solide associé.

Exemple :

volume_solide(i) est le volume de *dessin_assemblage(solide_acad(i))* pour AutoCAD et *dessin_assemblage.Item(i)* pour CATIA.

Nom des solides

Déclaration : *Public nom_solide(1 to n_solides) As String*

Rôle :

Les noms des solides sont enregistrés dans ce tableau.

Exemple :

nom_solide(i) est le nom de *dessin_assemblage(solide_acad(i))* pour AutoCAD et *dessin_assemblage.Item(i)* pour CATIA.

Matrices de contact et de translation

Matrice de contact

Déclaration: *Public M_contact(1 to n_solides, 1 to n_solides) as Boolean*

Rôle :

Cette matrice contient les valeurs de contact de chaque paire de solides, comme définies dans le chapitre 3.

Matrice de translation

Déclaration :

Public M_translation(1 To n_solide, 1 To n_solide, 1 To n_directions) as Boolean

Rôle :

Cette matrice contient les valeurs de translation de chaque paire de solides et de chaque direction, comme définies dans le chapitre 3.

Liste des directions d'assemblage possibles

Déclaration : *Public Liste_Directions As New Collection*

Chaque élément de la collection contient un tableau de 3 paramètres définissant le vecteur directeur de la direction.

Données nécessaires pour la génération des séquences

Liste des états

Déclaration : *Public etat_solide() As New Collection*

Rôle :

Chaque élément de cette collection représente un état qui résulte de l'assemblage de deux autres états. Comme indiqué dans le chapitre 3, tous ces états ne permettront pas forcément de terminer la séquence d'assemblage.

Description :

etat_solide(i) représente l'état n°i. Cet état est représenté par une collection qui contient les informations suivantes :

[*nombre_solides*; *liste_ordonnée_des_solides*; *liste_des_assemblages_qui_mènent_à_cet_état*]

nombre_solides : c'est le nombre de solides que l'état contient.

liste_ordonnée_des_solides : c'est la liste des numéros des solides que l'état contient.

liste_des_assemblages_qui_mènent_à_cet_état : c'est la liste des manières d'assembler deux autres états pour former l'état n°i.

Exemple :

[2, 1, 3, (3, 8, 7), (4, 8, 7), (4, 9, 12)] représente l'état constitué des 2 solides n°1 et n°3. Pour former cet état on a 3 possibilités :

1) Assembler les états n°8 et n°7 selon la direction n°3;

2) Assembler les états n°8 et n°7 selon la direction n°4;

3) Assembler les états n°9 et n°12 selon la direction n°4.

Taille des états

Déclaration : *Private n_etats() As Integer*

Rôle :

Ce tableau permet d'accéder directement aux états d'une taille donnée dans *etat_solide.*

Exemple :

L'ensemble des états contenant i solides est l'ensemble des éléments de *etat_solide* dont les indices sont compris entre *n_etats(i-1)+1* et *n_etats(i).*

Caractéristique des états

Déclaration : *Public etat_caract(1 to n_etats(n_solides), 0 to 2) As Variant*

Rôle :

Ce tableau indique 3 caractéristiques des états :

- *etat_caract(i, 0)* indique si l'état i permet de terminer l'assemblage. Si ce n'est pas le cas l'état pourrait être est supprimé. Sinon :
- *etat_caract (i, 1)* indique si il existe une séquence qui passe par l'état i et qui est active (c'est à dire si elle respecte tous les filtres activés).
- *etat_caract (i, 2)* indique l'abscisse de l'état i sur le *And/Or Graph.*

Liste des séquences

Déclaration : *Public liste_sequence As New Collection*

Rôle :

Le tableau *etat_solide* contient toutes les informations nécessaires pour retrouver toutes les séquences d'assemblage. Mais le programme stocke aussi ces séquences de manière explicite pour simplifier la suite du programme dans *liste_sequence.*

Description :

Chaque séquence est enregistrée sous la forme d'un tableau de 3 lignes et *n_solides* colonnes :

- Chaque colonne représente l'insertion d'un nouveau composant dans la séquence;
- La première ligne représente le numéro du solide inséré;
- La seconde représente la direction d'insertion;

- La troisième indique le nombre de solides insérés à la fois (pour les séquences non linéaires).

Le tableau ci-dessous représente la séquence linéaire de la figure 6.2 :

	Solide de base	1° solide inséré	2° solide inséré	3° solide inséré	4° solide inséré
Solide n°	1	2	3	4	5
Direction n°		1	3	3	4
Nombre de solides insérés		1	1	1	1

Nous avons les correspondances :

- *Base* = solide n°1;
- *Dessus* = solide n°2;
- *Tige* = solide n°3;
- *Bouchon_gauche* = solide n°4;
- *Bouchon_droit* = solide n°5.

Le tableau ci-dessous représente la séquence non linéaire de la figure 5.2 :

	Solide de base	1° solide inséré	2° groupe de solides inséré	3° solide inséré	4° solide inséré
Solide n°	5	3	1	2	4
Direction n°		3	3	1	3
Nombre de solides insérés		1	2	1	1

Liste des états contenus dans chaque séquence

Déclaration : *Public etat_dans_sequence As New Collection*

Rôle :

Cette collection permet de retrouver rapidement tous les états qui font partie d'une séquence donnée. Chaque élément de la collection est un tableau qui contient la liste des états et leurs états pères (cette dernière information sera nécessaire plus tard pour construire le *And/Or Graph*).

Pour une séquence donnée on dit qu'un état est l'état père de deux autres états si il est égal à l'assemblage des deux. Pour chaque état (excepté celui qui correspond au produit final) et pour chaque séquence il y a un et un seul état père.

Description :

etat_dans_sequence(sequence) = ((etat$_1$, etat_pere$_1$), (etat$_2$, etat_pere$_2$), ... , (etat$_n$, etat_pere$_n$))

Liste des séquences associées à chaque état

Déclaration : *Public sequence_dans_etat() As New Collection*

Rôle :

Ce tableau permet de retrouver rapidement toutes les séquences qui passent par un état donné (c'est le tableau symétrique du tableau précédent). Chaque élément du tableau correspond à un état et contient la collection des séquences qui le contiennent.

Description :

Sequence_dans_etat(etat) = ((seq$_1$, etat_pere$_1$), (seq$_2$, etat_pere$_2$).... (seq$_n$, etat_pere$_n$))

Données nécessaires pour l'évaluation des séquences

Liste des filtres qui ne sont pas associés à une note

Déclaration : *Public filtre_dur(1 To 1) As Boolean*

Rôle :

Ce tableau contient la liste des filtres pour lesquels aucune note n'est attribuée. Dans notre projet il n'y en a qu'un : « interdire l'assemblage depuis le bas ».

Liste des filtres et des critères associés à une note

Déclaration : *Public filtre_note(1 To 5) As Boolean*

Rôle :

Ce tableau contient la liste des critères pour évaluer les séquences. Comme vu dans le chapitre 5, ils sont utilisés de deux manières :

- Pour attribuer des notes aux séquences;
- Pour supprimer celles qui n'ont pas 10 si le critère correspondant est actif.

Liste des états interdits et des séquences interdites

Déclaration :

Public liste_seq_interdit As New Collection

Public liste_etat_interdit As New Collection

Rôle :

Ces deux collections contiennent respectivement la liste des séquences et des états qui ont été interdits manuellement par l'utilisateur.

Notes attribuées aux séquences

Déclaration : *Public notes_sequences(1 to $n_{sequences}$, 1 to $2+n_{filtres}$) As Variant*

Rôle :

Ce tableau contient les notes de chaque séquence pour chacun des filtres. Pour le seul filtre qui n'admet pas de note le tableau indique si la séquence respecte ou non le critère.

Enfin la dernière colonne indique si la séquence respecte l'ensemble des critères activés et donc si elle peut être affichée.

Séquence	Filtre sans note	Note critère 1	...	Note critère n	Note moyenne	Séquence à afficher?
...						
i	True/False	$Note_1$	...	$Note_n$	Moyenne	True/False
...						

Liens entre les états

Déclaration : *Public liens() As New Collection*

Rôle :

Ce tableau indique pour chaque état actif quels sont les états pères qui sont aussi actifs. Il permet de savoir quels traits dessiner entre quels états dans le *And/Or Graph*.