THÈSE

Présentée à

l'UNIVERSITÉ DE METZ

Pour l'obtention du grade de :

DOCTEUR DE l'UNIVERSITÉ DE METZ

Spécialité : INFORMATIQUE

Sandrine LEINEN

UNE NOUVELLE APPROCHE POUR LA MODÉLISATION ET LA GESTION DES CONTRAINTES EN CAO

Soutenue à Metz, le 19 décembre 1997

Composition du jury :

Directeur de thèse :	Yvon	GARDAN	(Professeur à l'Université de Metz)
Rapporteurs :	Umberto	CUGINI	(Professeur à l'Université de Parme)
	Denis	VANDORPE	(Professeur à l'Université C. Bernard de Lyon)
Examinateurs :	Jean-Pierre	JUNG	(Professeur à l'Université de Metz)
	Roland	MARANZANA	(Professeur à l'École de Technologie Supérieure de Montréal)
	René	SOENEN	(Professeur à l'Université de Valenciennes)

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

THÈSE

Présentée à

l'UNIVERSITÉ DE METZ

Pour l'obtention du grade de :

DOCTEUR DE l'UNIVERSITÉ DE METZ

Spécialité : INFORMATIQUE

Sandrine LEINEN

UNE NOUVELLE APPROCHE POUR LA MODÉLISATION ET LA GESTION DES CONTRAINTES EN CAO

Soutenue à Metz, le 19 décembre 1997

Composition du jury :

Directeur de thèse :	Yvon	GARDAN	(Professeur à l'Université de Metz)
Rapporteurs :	Umberto	CUGINI	(Professeur à l'Université de Parme)
	Denis	VANDORPE	(Professeur à l'Université C. Bernard de Lyon)
Examinateurs :	Jean-Pierre	JUNG	(Professeur à l'Université de Metz)
	Roland	MARANZANA	(Professeur à l'École de Technologie Supérieure de Montréal)
	René	SOENEN	(Professeur à l'Université de Valenciennes)

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

À mes parents,

À Frédéric,

« J'hésite à citer, car citer c'est tronquer. »

J. Green

« Attendre d'en savoir assez pour agir en toute lumière,
c'est se condamner à l'inaction. »

J. Rostand

REMERCIEMENTS

Mes premiers remerciements vont à mon directeur de thèse, Monsieur le Professeur Yvon GARDAN, pour m'avoir accueillie au sein de son laboratoire et m'avoir fait découvrir le monde de la recherche et plus particulièrement le domaine de la CAO. Sa clairvoyance, ses connaissances et sa confiance m'ont été précieuses.

Je remercie également Monsieur le Professeur Jean-Pierre JUNG pour son encadrement, ses critiques pertinentes et ses relectures accrocheuses de ce document qui ont largement contribué à l'amélioration de sa qualité.

Je tiens à remercier également messieurs U. CUGINI, professeur à l'Université de Parme et D. VANDORPE, professeur à l'Université Claude Bernard de Lyon, pour l'intérêt dont ils ont fait preuve à l'égard de mon travail en acceptant d'être rapporteurs de cette thèse.

Pour leur participation en tant que membres du jury et pour avoir ainsi accepté de juger mon travail, j'adresse mes remerciements à messieurs R. MARANZANA, professeur de l'Université du Québec et R. SOENEN, professeur à l'Université de Valenciennes.

À Estelle, Isabelle, Benoît et Frédéric, j'exprime ma reconnaissance pour leur soutien, leur disponibilité et leur amitié.

À tous les membres du LRIM, merci pour leur disponibilité et l'atmosphère conviviale qu'ils savent faire régner.

Enfin, mes derniers remerciements, mais pas les moindre, vont à mes parents qui m'ont toujours soutenu et m'ont fait bénéficier des meilleures conditions pour la réussite de mes études.

À tous ceux que je viens de nommer et à tous ceux que j'ai oubliés, merci.

Sandrine.

TABLE DES MATIÈRES

INTRODUCTION

Les connaissances et le savoir-faire des logiciels de CAO (Conception Assistée par Ordinateur) ainsi que les techniques d'IA (Intelligence Artificielle) sont associés de plus en plus souvent à la résolution de problèmes dans des domaines nombreux et variés. Initialement, les systèmes de CAO se contentaient souvent d'être des calculateurs performants ou encore des outils de dessin 2D. À présent, tout logiciel de CAO doit fournir à l'utilisateur une réelle assistance, être capable de réaliser des raisonnements et même, dans le meilleur des cas, fournir des solutions valides et auxquelles l'utilisateur n'avait pas forcément pensé.

Dans le premier chapitre, nous présentons les principales caractéristiques requises par les systèmes de CAO actuels ainsi que les méthodes de mise en œuvre possibles. Parmi ces différentes propriétés, nous nous intéressons plus particulièrement à la gestion des différents types de contraintes. Dans les systèmes de conception sous contraintes, on considère généralement trois types de contraintes correspondant à des niveaux d'abstraction différents. Le niveau le plus concret est celui permettant d'exprimer les contraintes géométriques qui sont des caractéristiques dimensionnelles sur les éléments. Le niveau intermédiaire correspond aux contraintes topologiques qui définissent les relations liant les éléments les uns par rapport aux autres. Le niveau le plus abstrait permet à l'utilisateur d'exprimer des contraintes fonctionnelles plus proches de sa démarche de conception : on exprime ainsi l'objectif à atteindre et non la manière d'y parvenir.

La présentation de problèmes contraints dans des domaines variés, montre que leur résolution peut se faire par des logiciels de CAO. On évoquera en particulier les problèmes de conception de pièces mécaniques, d'ordonnancement des tâches ou encore d'aménagement spatial.

Tout d'abord, au deuxième chapitre, nous nous intéressons à la gestion des contraintes géométriques et topologiques lors de la conception de pièces mécaniques et plus particulièrement à l'étude de la comparaison de modèles de CAO par normalisation de graphes. L'objectif du système que nous souhaitons réaliser, est d'assister efficacement l'utilisateur en lui fournissant des informations complémentaires sur la pièce en construction, de manière à rendre la conception et la fabrication de nouvelles pièces plus simples et plus performantes. Pendant la conception, le système se propose d'indiquer à l'utilisateur des contraintes géométriques ou topologiques qui existent implicitement sur la représentation de sa construction ou de comparer la pièce à un

ensemble de pièces existantes. Cette comparaison peut ainsi permettre à la conception la réutilisation en totalité ou en partie de pièces existantes. Lors de processus ultérieurs, comme la fabrication, si la nouvelle pièce est similaire à une pièce déjà conçue et fabriquée, on pourra réutiliser des informations de fabrication (gammes d'usinage, calibrages, gammes opératoires, ...).

Pour réaliser de telles comparaisons, nous définirons différents types d'égalités. A priori, l'égalité de deux pièces est obtenue si ces pièces sont constituées des mêmes éléments et des mêmes contraintes. Or, deux pièces similaires peuvent avoir des modèles de représentation différents compte tenu de l'historique de construction qui peut varier mais également des expressions multiples possibles d'une même combinaison de contraintes.

Afin de représenter précisément les objets à comparer, nous avons défini un modèle de contraintes qui utilise comme modèle de représentation une structure de graphe. Outre sa grande expressivité et son extensibilité, cette représentation permet de mettre en œuvre une conception variationnelle qui, comme nous le présentons au premier chapitre, a de nombreuses qualités. Pour une pièce donnée, il existe autant de graphes différents que de façons différentes d'exprimer la combinaison de contraintes impliquées, une représentation unique de toute pièce est donc indispensable à une recherche performante d'égalité entre pièces.

Nous avons considéré, dans un premier temps, le graphe complet contenant toutes les contraintes possibles comme une représentation unique de la pièce. Cependant, même s'il modélise de manière unique une situation donnée, sa taille le rend difficilement utilisable. Nous proposons de définir un graphe normalisé représentant toute construction de manière unique et avec une structure de taille minimale.

La représentation de chaque objet sous forme de graphe normalisé est utilisée pour réaliser deux types de comparaison de graphes. La première méthode présentée est une méthode algorithmique qui à l'aide de filtres, cherche à apparier tout nœud d'un des graphes avec un nœud de l'autre graphe. La seconde méthode réalise la comparaison de graphes en utilisant le formalisme CSP (Constraint Satisfaction Problem) qui permet de modéliser assez naturellement ce problème tout en fournissant des algorithmes robustes de résolution. Dans ce type de problème, on donne un ensemble de variables qui ont chacune un domaine de valeurs et un ensemble de contraintes. Le problème consiste à trouver une affectation de valeurs aux variables, telle que les contraintes soient satisfaites. Nous réalisons finalement dans ce chapitre, une

comparaison des performances de la méthode algorithmique et de la méthode basée sur le formalisme CSP.

Au troisième chapitre, nous abordons l'expression, la modélisation et la résolution des contraintes fonctionnelles. Nous étudions ce type de contraintes dans le cadre d'une application CAO sur l'aménagement spatial d'objets contraints. Dans un premier temps, nous présentons donc différentes méthodes utilisées pour la résolution de problèmes de placement. Nous considérons également plusieurs modélisations des données nécessaires au problème d'aménagement spatial d'objets contraints c'est-à-dire, l'espace de placement, les objets et les contraintes. Pour chaque modélisation, nous précisons si elle nous semble plus ou moins appropriée à notre problème.

Un problème particulier d'aménagement spatial est considéré : la conception de cuisines intégrées. Les méthodes de modélisation mises en œuvre sont ensuite présentées ainsi que deux méthodes de résolution. La première méthode est une méthode algorithmique qui dispose de plusieurs optimisations techniques. Quant à la seconde méthode, elle est basée sur le concept de CSP déjà mis en œuvre au chapitre précédent.

Finalement, nous montrons que le problème traité est semblable à celui de la conception d'usine avec toutefois un impact économique différent et une importance du temps dans ce processus décisionnel différent.

Les annexes que nous présentons à la fin de ce mémoire rappellent certains concepts fondamentaux et décrivent les différentes implémentations qui nous ont permis d'illustrer nos travaux.

CHAPITRE 1.

ÉTAT DE L'ART

1. INTRODUCTION

Dans ce mémoire, nous nous intéressons à la gestion des contraintes par les systèmes de CAO, or l'utilisation de nos jours, de logiciels de CAO dans des domaines très divers et par des utilisateurs souvent non informaticiens oblige ces systèmes à être de plus en plus conviviaux.

Dans une première partie, nous présentons certains services requis de plus en plus systématiquement par les systèmes de CAO actuels, ainsi que des méthodes de mises en œuvre possibles de ces différentes fonctionnalités. En effet, nous pouvons remarquer que la facilité de manipulation des différentes contraintes dépend du type de conception utilisée. Trois problèmes fortement liés sont abordés : la réutilisation de connaissances, la classification d'ensembles de solutions et l'inclusion de bibliothèques. Effectivement, la possibilité pour un utilisateur d'un système de CAO de réutiliser tout ou partie de pièces précédemment construites, ou encore des solutions ou ébauches de solutions pour certains problèmes, c'est-à-dire des connaissances acquises auparavant, est un point fondamental. Pour offrir cette fonctionnalité aux utilisateurs, les systèmes de CAO manipulent des bibliothèques, ils les construisent, les consultent, les manipulent, ... Une optimisation de l'utilisation des bibliothèques peut passer par une classification de l'ensemble des solutions ou données, de manière à réduire le nombre d'éléments voisins. La prise en compte de ces différentes notions par les systèmes de CAO améliore leur convivialité et leur efficacité quelque soit le type de contraintes manipulées.

Dans notre étude de la gestion des contraintes par les systèmes de CAO, nous nous intéressons ensuite aux contraintes fonctionnelles qui constituent un type de contraintes encore assez peu pris en compte par les systèmes. Elles interviennent dans de nombreux problèmes et permettent, par exemple, d'exprimer des notions plus abstraites que les contraintes géométriques ou topologiques. Nous étudions la gestion des contraintes fonctionnelles dans un domaine d'application particulier qui est celui de l'aménagement spatial d'objets contraints.

Nous présentons notamment des méthodes particulièrement intéressantes de modélisation et de traitement de ces contraintes fonctionnelles. Les méthodes d'aménagement spatial, quand elles sont suffisamment générales peuvent permettre d'aborder des domaines d'applications multiples. Les objets à placer dépendent du type d'applications considérées.

Les méthodes visant à résoudre les problèmes d'occupation d'espace peuvent être dédiées dans certains cas à l'aménagement de salles informatiques, de cuisines, d'ateliers, de sous-marins, ... Les objets à placer sont alors des ordinateurs, des imprimantes, des meubles, des appareils électroménagers, des machines, ...

Quand le problème à résoudre est le placement d'éléments en vue de leur connexion, les éléments considérés sont généralement des modules de circuits intégrés, des rectangles interconnectés nécessitant une répartition sans chevauchement sur une surface bornée ou encore des éléments mécaniques, électroniques et pneumatiques à implanter dans des structures métalliques.

Ces méthodes peuvent également être utilisées pour le partitionnement, de façon optimale, d'un espace donné. Les éléments peuvent alors être, par exemple, des pièces dans le cas de réalisation de plans de maison ou encore, des parcelles lors de l'organisation de forêts en vue de la réalisation de coupes.

Finalement, bien que les problèmes rencontrés dans les problèmes d'ordonnancement de tâches et dans les problèmes d'aménagement spatial soient a priori de nature différente, l'organisation de la production dans un atelier comporte également des problèmes spatiaux. Nous étudierons donc ces deux aspects avec pour objectif de réutiliser ou d'adapter éventuellement certaines méthodes utilisées en ordonnancement de tâches pour notre problème.

Comme nous venons de le voir brièvement, les domaines d'études relatif à l'aménagement spatial sont divers et variés. Les contraintes à traiter sont toutes aussi diverses. C'est pourquoi une partie de ce chapitre est consacrée à la présentation des différents types de contraintes utilisés dans plusieurs systèmes étudiés.

2. LES SYSTÈMES DE CAO INTELLIGENTS

Dans le cadre de l'étude de la gestion des contraintes, la convivialité du système est déterminée notamment en fonction du type de conception choisie pour le développement du système ; le type de conception dépendant lui-même de la finalité du système. Ainsi, dans une première partie, nous présentons les différentes méthodes de conception existantes.

L'intérêt d'un logiciel de CAO peut être lié aux services qu'il propose. En effet, les systèmes actuels s'attachent à proposer de plus en plus d'options permettant de faciliter le travail de l'utilisateur quel qu'il soit. La réutilisation de connaissances, la classification de solutions, l'inclusion de bibliothèques ou la gestion de contraintes fonctionnelles font partie des fonctionnalités proposées presque systématiquement par les systèmes de CAO. Elles seront présentées également dans cette partie.

2.1. Conception paramétrique ou variationnelle

Pour la conception de nouveaux produits, les systèmes de CAO traditionnels exigent souvent de la part de l'utilisateur une spécification précise de la localisation et des dimensions de chaque élément de base.

Un ingénieur a besoin d'outils de conception automatisés qui facilitent la créativité, encouragent l'évaluation d'approches de conception alternatives et permettent leur optimisation pour arriver à la meilleure solution possible. C'est pourquoi, les méthodes dites de conception conduite par les dimensions se chargent de mettre à jour la géométrie et de maintenir les contraintes géométriques et d'ingénierie représentant les intentions de l'utilisateur lorsqu'un paramètre est modifié.

Il existe essentiellement deux types de systèmes : les systèmes de conception paramétrique et les systèmes de conception variationnelle. Dans [CHU 90], les auteurs fournissent une définition de chacune de ces conceptions (il n'existe pas de définition universelle).

Une conception variationnelle est définie comme une méthodologie de conception qui utilise la théorie des graphes fondamentaux et des techniques robustes de recherche de solutions numériques pour fournir un système conduit par les contraintes et applicable à une combinaison couplée de contraintes géométriques et d'équations d'ingénierie. Une conception paramétrique est une méthodologie de conception qui utilise une recherche de solutions par cas particuliers pour produire un système conduit par les dimensions (sous-ensemble des méthodes conduites par les contraintes) et applicable principalement à des contraintes géométriques non couplées et des équations simples. On pourra trouver un exemple de système à géométrie paramétrique basé sur les dimensions dans [ROL 91] même si déjà ce système tend à ajouter des propriétés supplémentaires non spécifiques aux conceptions paramétriques.

Paramétrique	Variationnel
Caractéristiques générales	
- restriction à la conception : ensemble prédéfini de contraintes géométriques donc pas d'extension possible des fonctionnalités ; - résolution locale ; - comportement intuitif : les enchaînements sont les mêmes qu'à la construction ⇒ facilité d'utilisation et de compréhension.	- combinaison de contraintes ; - couplage de contraintes (géométrique + ingénierie) ; - évolutivité : de nombreux phénomènes sont traduisibles en équations ; - complexité : pas de chronologie respectée donc impossible d'anticiper les changements.
Domaines d'application	
- géométrie non couplée.	- manipulation d'entités géométriques ; - manipulation de contraintes.
Sous-dimensionnement	
- bloqué par manque d'information.	- capable d'obtenir dans certains cas une solution valide.
Équations d'ingénierie	
- peu traitées (seulement de façon isolée).	- traitement identique aux contraintes géométriques.
Résolution de conflits	
- détermination d'un ensemble possible de contraintes.	- détermination d'un ensemble minimal de contraintes.
Bilan	
- adapté aux modifications mineures ; - adapté à la création d'objets paramétrés.	- haut niveau de conception ; - adapté à la création de nouvelles alternatives.

Figure 1.1. *Tableau récapitulatif : comparaison des deux systèmes de conception*

La figure 1.1 propose un tableau récapitulatif des caractéristiques de chacun des deux types de conception variationnelle et paramétrique. On peut constater que le choix du type de conception pour un système, est fonction des caractéristiques requises pour le logiciel en question.

Nous avons été amené à choisir un type de conception mais ce n'était pas l'objet fondamental de notre étude ; nous ne réaliserons donc pas, dans ce document, une étude détaillée des systèmes variationnels et paramétriques existants, cette étude ayant été réalisée précédemment dans [LEI 94]. Nous préciserons simplement notre choix, ainsi que les raisons qui nous ont amenés à choisir une méthode de conception plutôt que l'autre.

Le mode de conception variationnelle est mieux adapté au système de CAO que nous souhaitons réaliser de part sa flexibilité et son expressivité. En effet, les systèmes de conception variationnelle peuvent en général réaliser les variations nécessaires lorsqu'un paramètre est

modifié ou lorsqu'une contrainte est ajoutée de façon à obtenir de nouvelles solutions consistantes. L'utilisateur du système est ainsi encouragé à tester les différentes alternatives d'approches d'une conception et peut choisir entre différentes solutions possibles. Les systèmes paramétriques, par contre, en cas de modifications, réévaluent l'historique de construction et ne fournissent dans le meilleur des cas qu'une seule solution.

L'utilisation d'une conception variationnelle pour un système permet également de stocker et de comparer ensuite différentes représentations d'objets. Cette fonctionnalité du système incite, par exemple, à créer des bibliothèques de solutions permettant la réutilisation d'éléments, ce qui apportera une aide intéressante pendant les phases de conception et de fabrication d'objets.

Des caractéristiques supplémentaires rendent un système variationnel attractif ([ALD 88], [GOS 88], [HAN 90], [LIG 82], [SHA 95]) :

- la plupart des intentions sont modélisées par les contraintes géométriques, les équations d'ingénierie et les liens entre les paramètres, ce qui permet aux autres utilisateurs d'apporter des changements à une conception en préservant les intentions du concepteur ;
- un modèle variationnel contient une description mathématique complète d'une conception ; l'analyse de tolérances, l'analyse des mécanismes et l'optimisation de la conception peuvent ainsi être prises en compte.

Le mode de conception variationnelle correspond aux critères que nous attendons d'un système de CAO ; nous avons donc développé une conception variationnelle, intégrant un graphe pour représenter les différents éléments (les nœuds) et les relations entre ceux-ci (contraintes sur les arcs).

2.2. Réutilisation de connaissances

Du point de vue technique dans [VER 95], l'objectif est d'éviter de recommencer la recherche à zéro après chaque modification du problème. La démarche consiste à mémoriser et à réutiliser des résultats obtenus lors de recherches précédentes, qui sont, soit des solutions (permises par l'ensemble des contraintes), soit des contraintes (déduites des contraintes explicites du problème). Tout problème de satisfaction de contraintes est défini par un ensemble de contraintes et un ensemble de solutions satisfaisant ces contraintes. On peut noter que le retrait de contraintes préserve l'ensemble des solutions (toute solution s valide pour un ensemble c_1 de contraintes est également valide pour un ensemble c_2 avec $c_2 \subset c_1$) par contre l'ensemble des contraintes n'est pas conservé. Inversement, l'ajout d'une contrainte préserve l'ensemble des contraintes mais ne maintient pas l'ensemble des solutions (toute solution s vérifiant un ensemble c_1 de contraintes

ne vérifie pas forcément un ensemble c_2 si $c_2 \supset c_1$). D'où l'objectif de développer des méthodes permettant de réutiliser les solutions et/ou les contraintes, résultats de recherches précédentes, quelle que soit la nature du changement.

Le système cherche à réutiliser les résultats de constructions, de calculs ou de raisonnements coûteux. Nous allons pour cela devoir répondre à trois grandes questions qui sont :

- dans quelles situations doit-on conserver des informations de façon à les réutiliser ultérieurement ?
- sur quels critères doit-on juger une situation intéressante et nécessitant une mémorisation ?
- quelles structures de données doit-on utiliser pour stocker les informations pertinentes ?

Seuls les résultats pertinents sont stockés de façon à ne pas avoir une structure de mémorisation trop importante, qui ralentirait la recherche et annulerait tout l'intérêt de la démarche.

On distingue deux situations pour lesquelles la mémorisation des résultats des recherches ou des calculs présente un intérêt certain. En cas de contradiction, il est généralement avantageux de conserver des règles permettant de lever l'ambiguïté ou des situations délicates déjà résolues :

- lorsqu'une solution correspondant à un problème assez complexe est trouvée, on s'attache à conserver la séquence d'actions qui a permis de trouver cette solution ;
- le choix par le système de la fonction à utiliser pour un problème donné peut être facilité par la mémorisation des contextes des situations ayant précédemment suscitées l'utilisation d'une certaine fonction.

Après avoir trouvé un résultat qui méritait d'être conservé, il faut déterminer comment le mémoriser. On envisage deux méthodes ayant chacune ses avantages et ses inconvénients.

La première méthode consiste à conserver peu d'informations ainsi qu'un ensemble de règles permettant de retrouver les informations importantes. Le système conserve, par exemple, un dessin en deux dimensions d'une pièce ainsi que ses cotations et, à l'aide d'une base de règles, il retrouve l'historique de construction. Cette méthode nous paraît trop peu fiable pour un système de CAO manipulant un grand nombre d'éléments et de contraintes. En effet, la base de règles est dans ce cas difficile à écrire, le nombre de règles différentes étant conséquent et l'ordre d'application des règles important. De plus, un grand nombre de combinaisons d'instructions différentes fournissent le même résultat.

Si le système dispose d'un graphe pour stocker son modèle contraint, la seconde méthode se propose pour stocker les différentes stratégies de résolution des problèmes, d'utiliser une structure d'arbre de résolution ET/OU. Cet arbre permet de représenter dans un ordre chronologique les objets traités ainsi que les contraintes associées. Il est le résultat d'une interprétation du graphe et des chemins correspondant à la résolution d'un problème particulier.

L'avantage d'utiliser un tel arbre est que le nombre d'informations à stocker est moins important que dans un graphe et la stratégie de résolution peut être déduite de l'arbre; il n'est donc pas nécessaire de stocker les différents chemins associés aux solutions. Par contre, un travail de réécriture des informations contenues dans le graphe est nécessaire ce qui est une tâche non triviale (figure 1.2). La méthode de conservation par arbre de construction ne semble pas très avantageuse pour notre système de CAO car une structure de graphe est utilisée pour stocker et manipuler les différentes données du système. Elle impliquerait donc une double structuration et manipulation des informations.

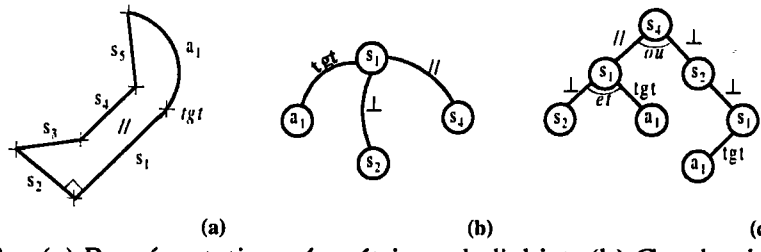


Figure 1.2. - (a) Représentation géométrique de l'objet- (b) Graphe de l'objet- (c) Arbre de construction-

2.3. Classification d'ensembles de solutions

En conception de pièces les utilisateurs travaillent souvent avec des informations sous-contraintes. Ceci implique que lorsqu'une modification est réalisée, le système trouve rarement une unique solution. Quand le nombre de solutions est assez conséquent, il peut être laborieux pour l'utilisateur de les visualiser toutes, les unes après les autres, en rejetant toutes celles qui ne conviennent pas. C'est pourquoi on envisage de regrouper les différentes solutions. L'utilisateur rejette ainsi en une seule fois tout un groupe de solutions voisines qui ne conviennent pas.

Après avoir retenu le groupe correspondant le mieux à la solution qu'il recherche, l'utilisateur peut affiner son choix en visualisant les différentes solutions du groupe choisi.

Le regroupement des chemins (représentation du parcours du graphe) correspondant aux différentes solutions est effectué à partir des solutions calculées. Il est en effet trop difficile de

regrouper les solutions a priori sur les chemins utilisés : deux chemins très proches peuvent aboutir à des solutions très différentes et inversement deux chemins complètement différents peuvent donner la même solution.

Tous les chemins différents conduisant à des solutions "voisines" sont regroupés et la complexité de cette méthode réside ainsi surtout dans la définition correcte de la notion de solutions "voisines".

La modification de la définition du voisinage en fonction du contexte permet de limiter le nombre de groupes différents. Il est important de trouver un équilibre de façon à avoir un nombre de groupes représentatifs des différentes solutions. L'inconvénient de cette méthode est que si l'on décide de conserver un unique représentant pour chaque groupe, celui-ci ne pourra représenter qu'une solution moyenne.

Il est intéressant d'identifier les différentes configurations du graphe conduisant à des solutions identiques. Les différentes situations pouvant donner des solutions identiques sont :

- des graphes ou des sous-graphes identiques ;
- des graphes différents mais des chemins identiques ;
- des graphes différents et des chemins différents.

Certaines situations sont plus facilement identifiables que d'autres, il est donc nécessaire de modifier le graphe (trouver une représentation normalisée des contraintes, définir des contraintes génériques, ...) et d'extraire de l'ensemble des connaissances, des règles de commutativité, d'associativité, ... On pourra aussi détecter les branches ou les cycles correspondant à des objets nuls de façon à rapprocher des pièces qui sont identiques mais dont la représentation interne et l'historique de construction sont complètement différents; ceci a déjà été utilisé efficacement dans les arbres CSG [TIL 87].

2.4. Inclusion de bibliothèques

Les deux parties précédentes suggèrent l'existence de bibliothèques associées à chaque logiciels de CAO. En effet, une mise en œuvre possible de la réutilisation de connaissances est la création de bibliothèques contenant des informations spécifiques au domaine d'application du système. Pour les logiciels de conception de pièces mécaniques, les bibliothèques sont constituées de modèles de pièces standard ou compliquées ; un logiciel d'architecture conserve

des exemples de plans au sol de logements de forme et de superficie différentes ; un paysagiste ou un jardinier est intéressé par le stockage de réalisations d'arrangement de terrains, ...

La classification de solutions, comme nous avons pu le constater précédemment, permet d'isoler un représentant par groupes d'éléments ; ces différents représentants correspondent typiquement au style de données à inclure dans les bibliothèques. La modélisation à l'aide de graphes peut être utilisée par les systèmes de CAO pour représenter les différentes entités conservées ainsi que les contraintes qui définissent cette pièce.

Les différents constituants d'une bibliothèque peuvent être utilisés de plusieurs façons mais la méthode suivante, nous semble efficace : lors de la création d'une pièce mécanique à l'aide d'un logiciel de CAO, le système, à certaines étapes de la construction, peut proposer à l'utilisateur des exemples de conception proche de la sienne. Le système retrouve ainsi pour l'utilisateur des pièces susceptibles de l'intéresser. Pour pouvoir proposer des éléments pertinents à l'utilisateur, le logiciel réalise des comparaisons de la situation courante avec des situations précédemment sauvegardées. Ce qui revient à comparer des graphes lorsque c'est ce type de modélisation qui est utilisé pour conserver les informations de la bibliothèque.

Or, lorsque l'on désire montrer l'égalité de deux graphes, on essaie d'apparier deux à deux tous les nœuds en respectant toutes les contraintes et caractéristiques des différents éléments. Une solution à ce problème consiste à effectuer un parcours en profondeur de l'arbre des valeurs possibles, avec retour arrière en cas d'incompatibilité. Ce problème peut être vu comme un problème de satisfaction de contraintes binaires. Nous allons donc présenter la notion de CSP ainsi que différentes méthodes de résolution et les appliquer aux cas qui nous intéressent.

2.4.1. Définition

Dans les problèmes de satisfaction de contraintes, ou CSP, un ensemble de variables est donné où chaque variable a un domaine de valeurs et un ensemble de contraintes (une contrainte lie deux variables). Le problème consiste alors à trouver une affectation de valeurs aux variables, de leur domaine respectif, telle que les contraintes soient satisfaites.

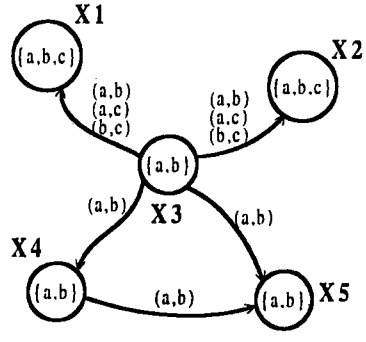
Les CSP binaires sont généralement représentés par un graphe où les nœuds représentent les variables et les arêtes, les contraintes. Souvent, chaque nœud contient l'ensemble des valeurs qu'il peut prendre et les arêtes représentant les contraintes sont accompagnées des couples de valeurs autorisées.

Variables	Domaines
x_1	{a,b,c}
x_2	{a,b,c}
x_3	{a,b}
x_4	{a,b}
x_5	{a,b}

(a) Variables et domaines

Contraintes	Variables	Relations
$c_1=(v_1,r_1)$	$v_1=(x_3,x_1)$	$r_1=\{(a,b),(a,c),(b,c)\}$
$c_2=(v_2,r_2)$	$v_2=(x_3,x_2)$	$r_2=\{(a,b),(a,c),(b,c)\}$
$c_3=(v_3,r_3)$	$v_3=(x_3,x_4)$	$r_3=\{(a,b)\}$
$c_4=(v_4,r_4)$	$v_4=(x_3,x_5)$	$r_4=\{(a,b)\}$
$c_5=(v_5,r_5)$	$v_5=(x_4,x_5)$	$r_5=\{(a,b)\}$

(b) Contraintes et relations



(c) Graphe du CSP

Figure 1.3. *Problème modélisé sous forme de CSP*

La figure 1.3 donne un exemple classique de problème représenté sous forme de CSP ; (a) représente les différentes variables du problème ainsi que leur domaine de valeurs associées ; (b) est un tableau récapitulatif des contraintes existantes, des variables qu'elles utilisent et des relations considérées ; (c) est la représentation sous forme de graphe du CSP.

2.4.2. Algorithmes de résolution

Pour résoudre un CSP, on dispose d'un certain nombre d'algorithmes de base dont font partie les procédures de recherche par "essais et erreurs" du type retour arrière. Dans [PRO 93], cinq algorithmes de base de recherche de solutions pour les CSP sont explicités et évalués. Nous allons présenter brièvement, ci-dessous, ces algorithmes :

- l'algorithme de recherche énumérative *(backtracking)* [WAL 60] : consiste à instancier les variables dans un ordre prédéfini en vérifiant si la valeur choisie satisfait les contraintes liant la variable courante aux variables déjà instanciées. Si une contrainte n'est pas satisfaite et si toutes les valeurs possibles pour la variable courante ont été testées alors le retour arrière est réalisé sur l'instanciation de la variable qui précède la variable courante afin d'essayer une autre valeur ;
- le retour arrière intelligent *(backjumping)* [GAS 79] : le premier retour arrière est réalisé directement vers la variable cause du conflit ; par contre, les autres retours arrière éventuellement nécessaires sont effectués vers la variable instanciée précédemment ;
- le retour arrière dirigé par les dépendances *(conflict-directed backjumping)* [STA 77] : lors de chaque retour arrière, un ensemble de variables de conflit est calculé dont les valeurs suffisent à expliquer un retour-arrière. On sait qu'il est alors inutile de considérer

toute instanciation partielle donnant à ces variables de conflit la même valeur que dans l'instanciation non consistante ;

- la mémorisation de contraintes aussi appelée apprentissage *(backmarking)* [GAS 77] : des contraintes induites par le CSP interdisant certaines instanciations sont mémorisées et prises en compte lors des parcours suivants ;
- la recherche de consistance en avant *(forward checking)* [HAR 80] : c'est une méthode prospective ; quand le processus de recherche fait un essai d'instanciation d'une variable, il vérifie les variables futures et supprime du domaine courant de ces variables toutes les valeurs qui sont incompatibles avec l'instanciation essayée. Le but de cet algorithme est d'échouer en détectant les inconsistances dans l'arbre de recherche aussi tôt que possible grâce au stockage des explorations d'alternatives infructueuses. Le retour arrière est chronologique. Cet algorithme fait plus de travail par nœud que les autres algorithmes présentés, mais tend à visiter le moins possible de nœuds.

Ces différents algorithmes ont été évalués sur plusieurs exemples et suivant différents critères (nombre de nœuds visités, nombre de vérifications effectuées, temps d'exécution, ...) ; ces études empiriques ([PRO 95], [HAR 79]) ont montré que l'algorithme de recherche de consistance en avant était le meilleur algorithme de l'ensemble des algorithmes énoncés ci-dessus. Cette analyse a également montré qu'il existait des algorithmes qui peuvent permettre, moyennant une implantation plus lourde, d'obtenir de meilleurs résultats.

2.5. Gestion des contraintes fonctionnelles

La prise en compte de données fonctionnelles par les logiciels de CAO permet à l'utilisateur d'exprimer des notions relativement abstraites. En effet, lorsque l'utilisateur donne une définition fonctionnelle d'un objet ("une hotte doit aspirer les odeurs, fumées, vapeurs, ... de cuisson"), le système déduit un certain nombre de contraintes qui peuvent être dimensionnelles, de positionnement, fonctionnelles, ...

Les contraintes fonctionnelles sont d'ordre général et expriment des caractéristiques requises par la solution recherchée. Lorsqu'un système permet à l'utilisateur de définir fonctionnellement un élément, ce dernier est soulagé de la définition d'un certain nombre de contraintes dimensionnelles, physiques ou mathématiques.

Il est intéressant d'exprimer fonctionnellement des données dans de nombreux domaines comme l'architecture, l'électronique ou la mécanique. En architecture, lors de la première étape de la réalisation de plan de logement qui consiste à définir un certain nombre de cloisons, les

différentes pièces ainsi créées sont de même nature. Par contre, dès lors que l'on attribue à chaque pièce une désignation (cuisine, salle de bain, séjour, chambre, ...), on définie fonctionnellement la pièce ; cette définition va déterminer un certain nombre de caractéristiques pour la pièce considérée. Si, par exemple, une pièce est désignée comme étant la cuisine, des contraintes dimensionnelles (minimum $8m^2$), topologiques (près de la salle à manger) ou fonctionnelles (possède une prise électrique haut voltage) peuvent être déduites.

De même, en électronique, lorsque l'on crée par exemple un ordinateur, plusieurs composants et caractéristiques sont requis pour sa conception ; l'ordinateur étant obligatoirement composé d'un écran, d'un clavier, d'un processeur, ... Le fait de préciser que l'une des caractéristiques de cet ordinateur est d'être portable, engendre beaucoup de contraintes de niveau plus concret et concernant entre autres les dimensions et le poids des différents composants.

Finalement, on peut considérer un exemple "gastronomique" pour montrer que les contraintes notamment fonctionnelles, sont présentes dans vraiment tous les domaines. Par exemple, pour la réalisation de menus dans les cantines scolaires ou les hôpitaux, des contraintes permettant d'équilibrer les repas sont considérées ; chaque repas requiert une certaine quantité de protéines, lipides, ... Or, la prise en compte d'informations concernant le destinataire du repas peut être considérée comme des données fonctionnelles et modifier les caractéristiques du menus. C'est notamment le cas, lorsque l'on précise que le destinataire est diabétique, végétarien, ...

2.6. Synthèse

Dans cette première partie de notre étude bibliographique, nous avons présenté certaines options dont la proposition par les systèmes de CAO est la bienvenue. Nous avons également ébauché la proposition de certaines méthodes permettant d'inclure ces options.

Pour ce qui concerne la réutilisation de connaissance, la classification de solutions ou l'utilisation de bibliothèques, nous ne détaillerons pas plus dans ce chapitre bibliographique notre étude. Par contre, un chapitre entier de ce document est consacré à la proposition d'une méthode permettant de mettre en œuvre ces fonctionnalités.

Au contraire, la gestion des contraintes fonctionnelles fait l'objet de la suite de notre état de l'art sur la gestion des contraintes et des caractéristiques des logiciels de CAO. Après la

présentation générale des contraintes fonctionnelles réalisée précédemment, nous approfondissons notre étude en immergeant ces contraintes dans des applications concrètes.

3. GESTION DES CONTRAINTES FONCTIONNELLES PAR LES SYSTÈMES DE CAO

Les contraintes fonctionnelles, intéressantes pour tout utilisateur néophyte, sont prises en compte de manière plus systématique dans les systèmes d'ordonnancement des tâches que dans les systèmes d'aménagement spatial. En effet, dans les systèmes d'aménagement spatial, les notions les plus importantes sont essentiellement de nature géométrique ou topologique même si la gestion des informations fonctionnelles est un atout majeur pour les systèmes l'incluant, cette gestion n'était pas jusqu'à présent indispensable. Par contre, dans les systèmes d'ordonnancement des tâches, les informations à traiter sont en grande partie fonctionnelle.

L'aménagement spatial consiste à disposer dans l'espace les éléments composant la solution du problème posé. Ce problème a déjà été étudié, notamment dans le cadre de l'électronique ; la difficulté consiste à répartir sans chevauchement des composants interconnectés, souvent assimilés à des rectangles, sur une carte électronique représentée par une surface bornée. Cependant, compte tenu de la forte combinatoire du problème et de la nature conflictuelle de certaines contraintes, il n'existe pas une méthode systématique donnant toujours la meilleure solution. Des méthodes heuristiques, généralement spécifiques au problème à résoudre, sont indispensables et rendent difficile la comparaison de l'efficacité des différentes méthodes.

Le problème d'aménagement spatial peut aussi être énoncé de la façon suivante [VER 92] :
Étant donné un ensemble d'objets et un ensemble de contraintes, trouver une disposition de ces objets qui satisfasse au mieux ces contraintes.

Le problème d'organisation de machines dans un atelier ou d'aménagement des meubles et de l'électroménager d'une cuisine intégrée peut être résolu en s'inspirant de la résolution d'autres problèmes. En effet, on retrouve sensiblement les mêmes difficultés dans les problèmes de réalisation de circuits intégrés ([CLÉ 87], [BRO 88], [KYU 92]) ou de réalisation de plans de maison par un architecte ([SCH 94], [CAO 90], [MED 96]).

Le placement sous contraintes de machines dans un atelier est un problème très vaste et non résolu de façon systématique par les systèmes de CAO. En effet, la difficulté du problème dépend de la complexité des contraintes mises en œuvre. Dans ce problème, comme de façon

générale, dans tous les problèmes d'organisation d'éléments dans des espaces libres, différents types de contraintes peuvent être distingués.

Nous étudierons par la suite, le problème de placement spatial. Cependant, nous nous sommes également intéressés aux problèmes d'ordonnancement de tâches, même si a priori, ces deux problèmes sont de nature différente. En effet, les problèmes d'organisation de production comportent également des problèmes spatiaux et notamment des problèmes spatiaux temporels. Ces problèmes spatiaux temporels pourront éventuellement être résolus de manière similaire dans les deux domaines.

3.1. Les problèmes spatiaux temporels

Nous abordons dans cette partie, les problèmes spatiaux temporels qui peuvent être rencontrés aussi bien dans les problèmes d'ordonnancement de tâches que dans les problèmes d'aménagement spatial.

3.1.1. En organisation de production

Les problèmes de planification et d'ordonnancement de la production consistent à organiser le travail en présence de contraintes sur le temps et les ressources. Étant donné un ensemble de tâches élémentaires et un ensemble de ressources dont la disponibilité est limitée, il faut organiser l'exécution des tâches dans le temps en respectant des contraintes variées. Les contraintes, dans ce type d'applications, sont fonctionnelles, c'est-à-dire, liées au comportement des différents intervenants.

Les contraintes de cohérence technologique ([GRA 93], [CAR 94], [HER 95]) sont liées à une utilisation et un partage des ressources ainsi qu'à une représentation cohérente, dans le temps, des ordres de fabrication. Dans [CAR 94], les contraintes de cohérence technologique concernent essentiellement les relations entre tâches. Les relations décrites sont des relations d'indépendance, de précédence ou encore d'interruption ou de préemption d'une tâche par rapport à une autre. Les différentes caractéristiques des tâches considérées sont résumées par la figure 1.4.

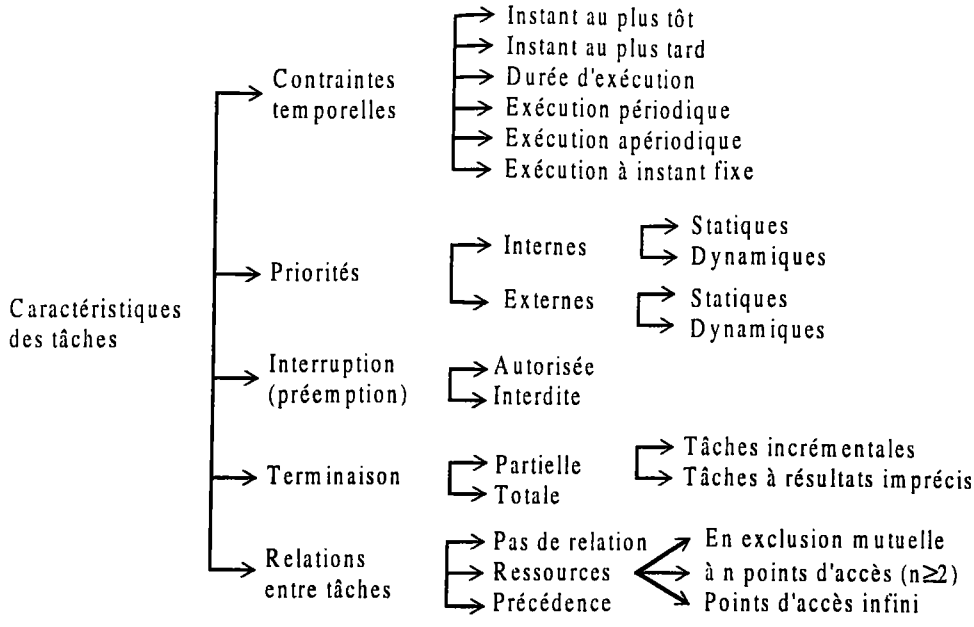


Figure 1.4. *Caractéristiques des tâches*

La qualité de la production dans une usine dépend en partie de l'organisation de l'atelier. En effet, il est toujours préférable que deux machines, utilisées consécutivement dans la chaîne de production, soient relativement proches pour réduire au maximum le temps perdu en échanges entre machines. Le positionnement d'une machine mobile ou d'outils, doit également être correctement synchronisé de manière à ce que les objets utiles soient toujours au bon moment au bon endroit.

Dans l'aménagement de certains types d'usine, il faut tenir compte non seulement de la localisation temporelle des machines mais également de la disponibilité des outils. Pour certains problèmes d'ordonnancement [HER 95], la prise en compte des contraintes d'outillage est indispensable. C'est le cas des ateliers où la capacité en nombre d'outils est limitée, des changements d'outils d'une machine vers une autre doivent alors être prévus. Lorsqu'un changement d'outils a lieu, le magasin d'outils de la machine est vidé, partiellement ou complètement, de ses outils, puis rempli des outils nécessaires au traitement des pièces suivantes prévues dans la séquence. Le temps nécessaire pour effectuer ce changement n'est d'habitude pas négligeable. Les remplacements d'outils ne pouvant se faire que lorsqu'il y a un instant de changement, l'ensemble des opérations entre deux instants de changement consécutifs peut être considéré comme une unique opération. Ainsi, étant donné un ensemble ordonné d'opérations qui doivent être effectuées sur une machine, l'objectif de l'algorithme d'ordonnancement sera de minimiser le nombre total de remplacements d'outils. Un des moyens de minimiser le temps

perdu pour les remplacements est de disposer les machines de manière à rapprocher deux machines qui doivent partager des outils.

Ce type de contraintes est appelé contraintes de disjonction dans [ERQ 95] car elles correspondent au fait que des ressources ne sont pas partageables. L'auteur évoque également à propos du partage des ressources, les contraintes cumulatives, qui interdisent l'exécution simultanée d'un nombre de tâches telles que l'intensité totale d'utilisation de la ressource dépasse l'intensité maximale de la ressource.

La synchronisation de tâches est également importante lorsque des éléments d'un produit final peuvent être fabriqués en parallèle car ils doivent se retrouver à un instant donné au même endroit pour l'assemblage. Le même type de problèmes spatiaux temporels se retrouve lorsque plusieurs produits sont en attente d'un traitement devant une même machine, une organisation optimale de la file d'attente est primordiale pour une rentabilité maximale du processus de fabrication.

3.1.2. En aménagement spatial

Si on considère maintenant l'aménagement spatial d'une usine, c'est-à-dire le placement respectif des différentes machines et non plus l'organisation de la production, on rencontre également un certain nombre de problèmes spatiaux temporels pas forcément très différents des précédents.

Lors de l'aménagement d'un atelier, le placement d'une machine à un emplacement donné peut être impératif à un instant fixé. Ceci, notamment, si cette machine est mobile et sert à l'installation ou encore si la machine suivante est en relation forte avec cette dernière (contrainte poser-sur). Ainsi, pendant son exécution, une tâche peut accepter ou non d'être interrompue au profit de la réalisation du placement d'une autre machine en relation avec la première et plus difficile à placer (plus encombrante, plus contrainte, ...).

Dans cette partie, nous venons de mettre en évidence le lien entre les problèmes d'ordonnancement et les problèmes d'aménagement spatial. Cette mise en évidence permet d'envisager de réutiliser dans un des domaines des méthodes employées dans l'autre.

3.2. Les problèmes spatiaux purs

Dans cette partie, nous allons présenter une nouvelle génération de logiciels de CAO en architecture. Ces logiciels qui se développent actuellement ne se contentent plus seulement de permettre à l'utilisateur d'exprimer des contraintes géométriques ou éventuellement topologiques entre les objets mais ils donnent la possibilité à l'utilisateur d'exprimer des besoins fonctionnels. Les logiciels d'architecture sont donc en pleine évolution, après avoir été longtemps considérés seulement comme des outils perfectionnés de dessin ou au mieux comme des modeleurs géométriques ; ils aspirent maintenant à apporter à l'architecte une réelle assistance.

La plupart des systèmes de CAO pour l'architecture distinguent maintenant trois types de contraintes correspondant plus ou moins aux trois niveaux de résolution : géométrique, topologique et fonctionnel. Ces trois niveaux peuvent avoir des dénominations différentes en fonction des systèmes considérés. Dans [KWA 97], les contraintes spatiales considérées sont soit topologiques (positionnement relatif des objets), soit de distances (peuvent être associées aux contraintes topologiques déjà définies), soit dérivées (plus à droite, moins haut, ...). Le même modèle de placement à trois niveaux est considéré dans les systèmes présentés dans [MED 96] et [CAO 90] mais le niveau géométrique est respectivement appelé niveau numérique et niveau dimensionnel. Le niveau fonctionnel est parfois appelé également niveau esthétique [CAO 90].

Les systèmes d'aménagement spatial peuvent intervenir dans différents domaines. Nous nous intéressons plus particulièrement aux problèmes d'aménagement des machines outils d'un atelier et d'aménagement des meubles et appareils électroménager d'une cuisine intégrée, mais nous avons également considéré des problèmes relatifs à des domaines différents. Nous avons étudié trois types de problèmes qui sont, l'aménagement d'un sous-marin, la réalisation de plan de maisons et la création de cartes électroniques.

3.2.1. Aménagement de compartiments propulsifs de navire

CADOO ([AND 86]) est un système d'aide intelligent pour l'aménagement spatial et probablement l'un des premiers systèmes d'aide à l'aménagement spatial utilisant des connaissances expertes et des stratégies générales de résolution de problèmes. Il a pour objectif, une satisfaction efficace des contraintes : l'espace de recherche est, pour ce faire, décomposé en différents niveaux de hiérarchie et des contraintes sont utilisées pour guider la recherche. Trois niveaux de résolution des contraintes sont distingués dans ce système : un niveau relationnel, un

niveau permettant de fixer des intervalles de valeurs pour les paramètres de conception et un niveau où sont choisies les valeurs discrètes.

L'application présentée pour illustrer la méthode proposée, consiste en un aménagement de compartiments propulsifs de navire. Le but du système est de respecter au mieux un ensemble implicite de contraintes fonctionnelles. Ces contraintes peuvent être :

- générales à tout aménagement : "Si un élément alimente un autre élément, ils doivent être proches" ;
- générales à tout aménagement de navire : "Un appareil électrique doit être éloigné d'un appareil où circule de l'eau" ;
- propres à des types de bateaux ou de propulsion : "Dans un bateau dont la coque est en forme de "U", les pompes d'eau de mer doivent être proches de l'axe".

Ceci implique pour le système :

- que les contraintes s'expriment de façon générique et dépendent de caractéristiques souvent fonctionnelles des éléments de l'aménagement ;
- que les contraintes utilisent très souvent des relations floues (près de, loin de). Elles sont alors de nature non-impérative et on peut généralement estimer leur importance relative au moyen d'un degré de sévérité.

Une particularité intéressante du système CADOO est la possibilité qu'il offre d'exprimer des méta-critères, ce qui revient à contrôler l'ordre d'application des critères. Par exemple, le critère de taille est prépondérant au début du placement, mais est devancé par la suite par le nombre de contraintes impératives.

3.2.2. Réalisation de plans au sol

L'aménagement de sol est un type de problème d'aménagement spatial. Le problème consiste à créer un placement 2D basé sur des contraintes topologiques, géométriques, fonctionnelles et esthétiques. De part la nature combinatoire exponentielle de ce problème de recherche de solutions, il est impossible d'effectuer une recherche exhaustive pour trouver toutes les solutions.

Le système Autodes

Dans [CAO 90], un modèle de planification distribuée est présenté. Après une décomposition incomplète, les interactions entre les sous-problèmes sont éliminées grâce à des opérateurs de négociation. Ce système intègre également la décomposition en trois niveaux citée précédemment ; les contraintes topologiques sont relatives au positionnement des pièces et aux

zones de trafic. Les contraintes sont représentées par un réseau d'adjacences. Ce réseau peut être utilisé pour vérifier la consistance des contraintes topologiques en utilisant un algorithme de vérification de planéité de la théorie des graphes.

Un tableau noir de données enregistre les autres contraintes dimensionnelles et fonctionnelles telles que le mode de construction, les contraintes de la région courante, les contraintes d'orientation pour la luminosité, ...

La solution au problème est recherchée en traitant les informations des plus abstraites aux plus spécifiques. Un problème est entièrement résolu à un niveau donné avant d'aborder le niveau suivant plus spécifique.

Le système Archiplan

Dans [MED 96], une définition succincte est donnée pour cette nouvelle génération de logiciels dont CADOO pourrait être un des premiers membres comme nous l'avons déjà vu. L'originalité de ces systèmes est de distinguer dans le modèle de placement trois niveaux : fonctionnel, topologique et numérique.

Le modèle à trois niveaux qui n'est pas quoiqu'en disent les auteurs de [MED 96] d'une grande originalité. En effet, il a déjà fait ces preuves (cf [AND 86] entre autres), permet de se rapprocher de la pratique du concepteur qui consiste en une méthode fonctionnelle et progressive. Ainsi, lors de la première phase de conception (à partir du cahier de charges), il y a énumération des différentes solutions topologiques. L'architecte peut alors "naviguer" dans l'espace de solutions topologiques (en faible nombre) et en choisir certaines pour une étude plus poussée. À partir de la solution topologique, la meilleure solution dimensionnelle est définie, en fonction du coût, des surfaces utiles, ...

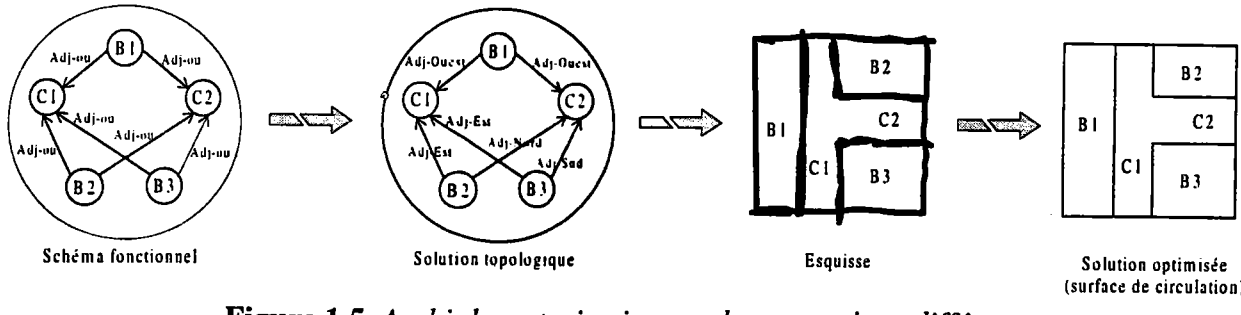


Figure 1.5. *Archiplan : trois niveaux de conceptions différents -fonctionnel, topologique, numérique-*

L'approche fonctionnelle pour un problème de placement doit pouvoir modéliser les contraintes fonctionnelles du besoin et visualiser l'espace des solutions de placements potentiels. Le concepteur doit pouvoir :

- étudier plusieurs alternatives de solutions ;
- classer les solutions ;
- affecter des priorités et des degrés d'incertitude aux contraintes ;
- tester de manière différentielle la relaxation de certaines contraintes.

Dans cette hiérarchie (figure 1.5), les solutions topologiques se situent entre les spécifications fonctionnelles et les solutions numériques. Une solution topologique est une classe d'équivalence de solutions numériques respectant les mêmes conditions d'adjacence entre tous les couples de locaux. La représentation d'une solution topologique est une esquisse sans dimension précise.

Dans ce modèle de représentation des espaces architecturaux, on distingue trois types de contraintes : les contraintes fonctionnelles, les contraintes implicites et les contraintes de réduction de l'espace de recherche. Les contraintes fonctionnelles sont des contraintes descriptives à partir du cahier des charges, d'adjacence, d'orientation et de communication. Les contraintes implicites sont des contraintes de non-recouvrement, d'inclusion et de recouvrement total de l'espace de placement. Les contraintes de réduction de l'espace de recherche sont des contraintes de symétrie, de réduction des topologies ou de propagation des orientations.

La décomposition et la modélisation des données très détaillées (peut-être trop : présentation des différentes classes mises en œuvre) nous semblent appropriées et judicieuses. Par contre, la méthode de résolution est à peine suggérée, c'est pourquoi nous ne reprendrons pas ces travaux dans la partie suivante présentant les différentes méthodes de résolutions liées aux représentations présentées dans cette partie.

Le système ABD (Automated Building Design)

Le système présenté dans [SCH 94] tente de résoudre le problème de conception automatisée de constructions en architecture. L'objectif est de développer un réel assistant d'architecte qui aide actuellement l'architecte en suggérant des alternatives de plans au sol qui respectent les contraintes imposées. Le système ABD aide l'architecte à prendre ses décisions basées sur une grande variété de plans acceptables et il peut fournir également une évaluation des différentes alternatives.

En utilisant le système ABD, l'architecte ne génère pas un plan, mais un ensemble de règles qui à leur tour, sont utilisées pour générer un ensemble de plans satisfaisants. Un éditeur de règles est mis à la disposition de l'architecte pour lui permettre d'exprimer un ensemble de règles portant sur des critères qui peuvent être fonctionnels, topologiques ou dimensionnels.

Le modèle ABD permet donc de définir des règles portant sur les critères suivants :

- fonctionnels :
  - adjacence entre deux pièces dans une direction donnée ;
  - séparation entre deux pièces dans une direction donnée ;
  - si un mur est extérieur ou intérieur ;
  - si un mur est imaginaire ou pas ;
  - l'intimité ou les attributs de bruit de la pièce.
- topologiques :
  - limite haute et basse sur la longueur et la largeur de la pièce ;
  - limite inférieure et supérieure de l'aire de la pièce ;
  - limite inférieure et supérieure de la largeur et de la longueur du lot ;
  - les types de portes.
- dimensionnels :
  - définition exacte de la longueur et de la largeur de la pièce ;
  - largeur et longueur exacte du lot ;
  - des règles conditionnelles de taille (satisfaites si au moins un des composants l'est).

3.2.3. Placement de composants sur une carte électronique

Le problème de placement le plus ancien est le problème de répartition sans chevauchement de rectangles interconnectés dans une surface bornée. Le domaine de l'électronique est le plus exploré notamment à cause de la simplicité géométrique des objets manipulés, de la discrétisation possible de l'espace utilisé et des limites de l'approche algorithmique face à la complexité croissante du problème posé.

Dans [VER 90], un processus de répartition dans la surface de placement optimise l'imbrication des blocs et permet le traitement de blocs de dimensions variées non fixées au départ. Dans [BRO 88], un concepteur de puces électroniques est comparé à un urbaniste, ils ont chacun un ensemble de modules qui doivent être branchés ou interconnectés en respectant un certain nombre de modèles prédéfinis : unité arithmétique et logique, ROM, RAM, multiplexeurs, maisons, centres commerciaux, immeubles administratifs, ...

Le premier système doit gérer une liste de connectivités tandis que le second gère des contraintes de régulations, de planifications locales et de sens commun ; la complexité du problème est la même mais l'échelle est différente. Dans cet article, le problème est présenté de la façon suivante : un nombre de modules 2D, de taille ou de forme arbitraire, doivent être arrangés aussi proche que possible sur une surface 2D et liés les uns aux autres en respectant certaines listes. Ce problème est alors décomposé en trois sous-problèmes : le placement, le routage global et le routage détaillé.

3.3. Synthèse

Dans cette partie, nous avons présenté des contraintes fonctionnelles qui sont manipulées par les systèmes d'ordonnancement de tâches mais également par les systèmes d'aménagement spatial : les contraintes spatiaux temporelles. Ces contraintes étant prise en compte plus systématiquement par les systèmes d'ordonnancement de tâches, les méthodes employées peuvent être réutilisées pour des problèmes d'aménagement spatial.

Cependant, cette partie est essentiellement consacrée à la présentation d'un certain nombre de problèmes d'aménagement spatial, des méthodes de modélisation utilisées et des contraintes manipulées puisque nous nous intéresserons à ce problème particulier au chapitre 3.

4. CONCLUSION

Dans cette partie, nous avons présenté un état de l'art sur les travaux relatifs aux systèmes de CAO et notamment à leur gestion des contraintes. Nous rappelons ici, tout d'abord, les différents thèmes abordés, puis nous dégageons les limites actuelles de la théorie et suggérons quelques pistes de réflexion.

Nous avons présenté des problèmes liés aux domaines de l'électronique, de la conception de pièces mécaniques, de l'architecture ou encore de l'organisation de production ou d'atelier et nous avons pu constater que pour tous, le nombre de solutions était souvent important et la recherche de ces solutions, sans l'aide du système, délicate.

Ces différents constats nous ont conduits à étudier des méthodes permettant d'assister efficacement l'utilisateur pendant les phases de conceptions. Ainsi, la première partie de cette étude bibliographique a présenté les fonctionnalités les plus importantes allant dans le sens d'une

assistance pertinente à l'utilisateur et proposées par les systèmes de CAO actuels. Parmi les nombreuses fonctionnalités offertes, nous en avons retenu quatre pour leur importance dans le domaine de la CAO qui nous intéresse plus particulièrement ; ce domaine étant la construction d'éléments (pièce mécanique, cuisine, atelier, ...) à partir d'objets mis en relation par un certain nombre de contraintes.

Nous avons exposé les intérêts de la réutilisation de connaissance ainsi que du regroupement des solutions en famille. Ces deux notions nous ont naturellement conduit à envisager la création et l'utilisation de bibliothèques. Par ailleurs, l'utilisation automatique des bibliothèques disponibles par le système peut permettre un gain de temps certain en plus des avantages déjà reconnus de leur utilisation. Nous proposons de faire réaliser par le système, cette comparaison d'éléments de la bibliothèque à la situation courante, en utilisant le formalisme de graphe des CSP ; c'est pourquoi une partie de ce chapitre est consacrée à la définition succincte du formalisme CSP ainsi qu'à la présentation de quelques algorithmes classiques de résolutions de ces problèmes particuliers.

La deuxième partie de ce chapitre se veut plus technique. En effet, parmi les différentes fonctionnalités souhaitables pour des logiciels de CAO intelligents et performants, la prise en compte des contraintes fonctionnelles est le point que nous avons décidé d'étudier plus avant. Dans cette partie, nous avons choisi de montrer l'expression, la modélisation et la résolution de ces contraintes au travers d'une application typiquement CAO qui est l'aménagement spatial d'objets sous contraintes. La position dans les systèmes, de ce type de contraintes par rapport aux autres a été précisée et des stratégies de résolution des problèmes d'organisation ou d'aménagement ont été exposées brièvement.

Finalement, cette étude nous a orienté vers deux grands thèmes de réflexion qui constituent, en fait, les deux prochains chapitres de ce document. Dans un premier temps, nous nous sommes interrogés sur le meilleur moyen de conserver des informations dans une bibliothèque et de les réutiliser efficacement par la suite. Dans un second temps, nos réflexions se sont portées sur le sujet très vaste de l'aménagement spatial d'objets contraints. Nous avons jugé ce sujet d'étude très intéressant car il permet la manipulation d'une panoplie de contraintes et d'objets très grande.

CHAPITRE 2.

COMPARAISON DE MODÈLES DE CAO PAR NORMALISATION DE GRAPHES

1. INTRODUCTION

La comparaison de modèles en CAO est un problème très délicat à traiter mais de grande importance. La difficulté de ce problème réside dans la multiplicité des représentations. En effet, quel que soit le domaine de conception considéré, il existe pour un même représentant plusieurs moyens de le construire, d'exprimer ces caractéristiques ou encore de conserver les différentes contraintes le liant aux autres objets. Ces différences peuvent conduire à des modèles différents pour des objets a priori identiques.

Par ailleurs, la détection de l'égalité de deux pièces, par exemple, peut faciliter leur fabrication. En effet, si une pièce a été construite en utilisant des contraintes, des caractéristiques et un historique de construction différents d'une pièce déjà définie et fabriquée mais si ces deux pièces sont similaires, on pourra réutiliser pour la nouvelle pièce, la gamme d'usinage, le calibrage et l'enchaînement des outils et machines.

La recherche d'égalité de pièces peut également être une aide à la conception pour l'utilisateur final du système. La détection de l'égalité ou un grand nombre de points communs entre la pièce en cours de construction et une pièce existante permet d'anticiper certaines actions de l'utilisateur final. De plus, si le système peut prévoir le type de manipulations et les fonctionnalités du système utiles au concepteur, le dialogue pourra être adapté à la construction en cours.

Les pièces existantes qui peuvent être comparées à la pièce en construction sont stockées dans des bibliothèques et peuvent également être reprises par l'utilisateur final pour être utilisées dans une nouvelle conception. Ce dernier aura la possibilité de les inclure à la pièce en création avec ou sans modification ou encore de ne rien ajouter mais de faire des transformations.

Nous avons développé une méthode permettant de rechercher l'égalité ou l'inclusion de deux pièces, en comparant leur représentation. Il existe de nombreux modèles de représentations des modèles de contraintes en CAO. En effet, les contraintes peuvent être exprimées à l'aide de graphes mais également par des règles (heuristiques), des prédicats logiques, des arbres ou encore des enregistrements. Dans notre application, nous utilisons une structure de graphe pour modéliser les contraintes notamment parce que les graphes ont une importante capacité d'expression mais aussi pour un certain nombre d'avantages évoqués au premier chapitre. Les objets de base ainsi que les relations entre ces objets constituant une pièce sont représentés dans

un graphe : pour comparer deux pièces, on comparera donc leur graphe. L'égalité peut tout à fait être recherchée entre les graphes de construction de différentes pièces ; cependant comme deux pièces totalement identiques ou avec des parties communes peuvent être construites différemment (historique de construction, utilisation de contraintes équivalentes, ...), l'égalité existant entre deux pièces ne sera pas forcément toujours détectée.

Ce chapitre se divise en trois parties. La première partie présente une méthode permettant d'aboutir à une représentation unique et minimale de chaque objet. Cette modélisation normalisée et restreinte de chaque objet est utilisée dans la deuxième partie pour réaliser la comparaison des différentes représentations. Finalement, la troisième partie est dédiée à la présentation de l'implémentation réalisée permettant d'illustrer les résultats énoncés précédemment.

2. MODÉLISATION NORMALISÉE MINIMALE D'OBJETS

Nous avons déjà eu l'occasion de préciser que dans notre système utilisant une conception variationnelle, les objets sont modélisés à l'aide de graphes. Notre objectif étant de définir pour chaque élément une représentation unique et minimale, un système de règles d'ajout et de remplacement de contraintes est proposé ; il permet de déduire du graphe utilisateur de l'objet un graphe normalisé restreint unique.

2.1. Modélisation des objets à l'aide de graphes

Un graphe est associé à tout modèle de pièces ; il permet de représenter les relations entre les différents éléments de base constituant l'objet. Un objet peut être composé de plusieurs parties non liées, chaque partie représente alors une composante connexe du graphe (figure 2.1).

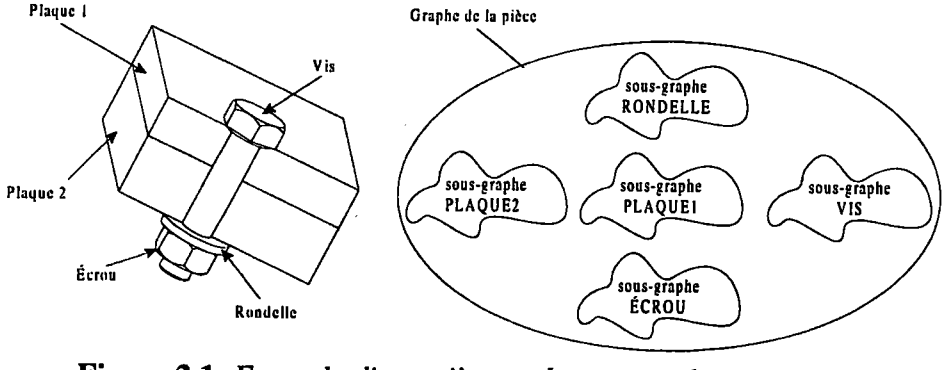
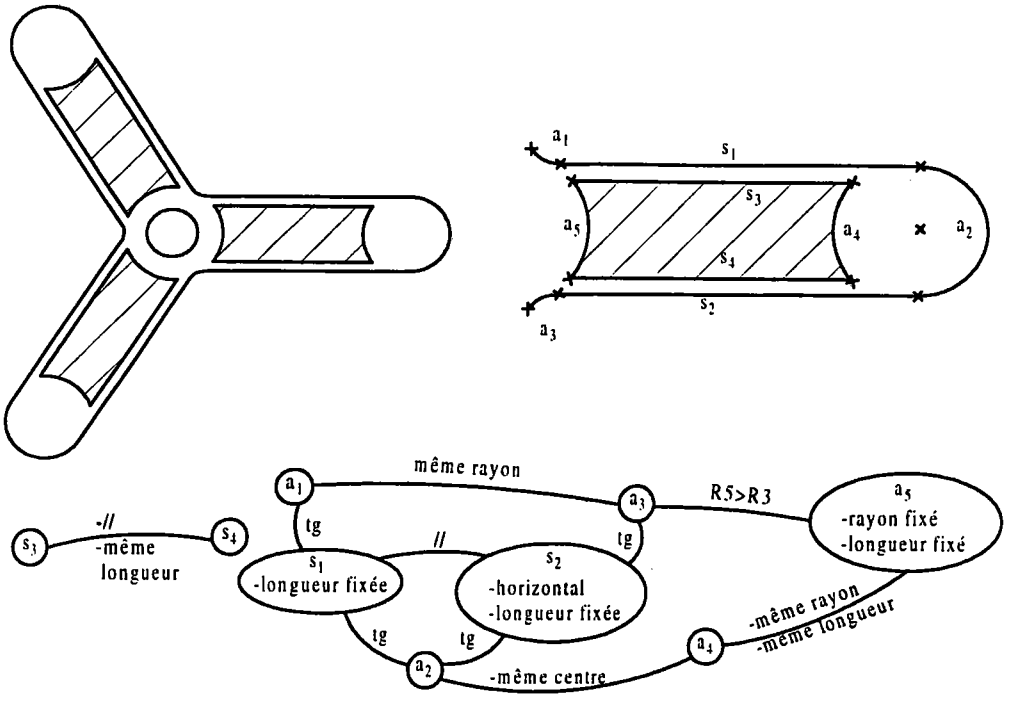


Figure 2.1. *Exemple d'une pièce et de son graphe associé*

Afin de simplifier l'étude et les démonstrations de ce chapitre, le domaine de définition de notre application est la géométrie 2D.

Notre système est un système multi-modèles. À toute pièce, deux modèles au minimum sont associés : un modèle de données et un modèle de contraintes. Le modèle de données peut être considéré comme le modèle géométrique de représentation des éléments. Dans ce modèle, un point est représenté par ses coordonnées, un segment par son point origine et son point extrémité et un arc par un centre, un rayon, un point origine ainsi qu'un point extrémité. De plus, à chaque élément géométrique de base est associé un identificateur.



Le graphe de la figure 2.2 comporte deux composantes connexes ; s_2 a deux contraintes internes ; s_3 et s_4 sont liés par deux contraintes binaires.

Figure 2.2. *Exemple détaillé d'une pièce et de sa représentation*

Le modèle de contraintes de chaque pièce est représenté par un graphe ; il permet de représenter les relations entre les différents éléments de base constituant l'objet. Le graphe est constitué de nœuds représentant des éléments de base désignés par leur identificateur et d'arêtes étiquetées représentant les relations entre les différents éléments. En tout nœud du graphe de contraintes, les contraintes sont représentées soit par des arêtes de liaison avec d'autres nœuds (contraintes binaires), soit par des caractéristiques propres au nœud lui-même (contraintes unaires). Lorsque plusieurs contraintes binaires existent entre deux mêmes objets, deux représentations peuvent être envisagées :

- la construction de plusieurs arêtes entre les deux nœuds concernés ; chaque arête représente alors une des contraintes ;
- la construction d'une seule arête mais qui possède plusieurs étiquettes.

L'exemple de la figure 2.2 propose une construction sous contraintes et sa modélisation sous forme de graphe en utilisant la deuxième méthode de représentation pour des raisons de lisibilité.

Toutefois, de manière interne, la première proposition de modélisation est employée. Cette représentation sous forme de graphe est étroitement liée au modèle géométrique de représentation des éléments. Pour éviter toute redondance entre ces deux modèles dépendants, certaines relations ne se trouvent pas dans le graphe comme, par exemple, les relations mettant en évidence un point commun entre deux éléments. Nous pouvons remarquer que considérer uniquement les deux types de contraintes unaires et binaires est suffisant car toute contrainte de degré supérieur peut être décomposée en plusieurs contraintes binaires (figure 2.3).

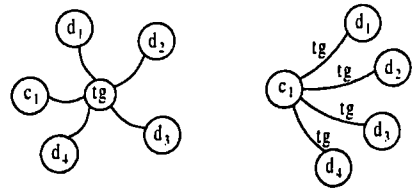


Figure 2.3. *Des relations binaires et unaires seulement*

Pour des raisons de simplification des démonstrations, notre étude est réalisée en utilisant uniquement les objets et les contraintes suivantes :

- objet : - segment
  - arc
- contraintes unaires : - horizontal
  - vertical
- contraintes binaires : - parallèle
  - perpendiculaire
  - tangent

2.2. Graphe normalisé complet et règles de production

Pour montrer l'égalité ou l'inclusion de pièces, le seul appariement de graphes n'est pas toujours suffisant, notamment lorsque les deux graphes représentent deux pièces identiques mais construites de façon différente. Différentes méthodes peuvent être envisagées pour éviter ce problème. Certains systèmes proposent une traduction de toutes les informations en relations de distances et angulaires. On pourrait envisager également d'établir un ensemble de règles tenant compte des équivalences possibles de combinaisons de contraintes. L'important étant d'avoir directement ou indirectement une représentation normalisée des différentes situations de façon à être sûr de retrouver si elle existe une égalité ou une inclusion de graphes.

La représentation la plus simple à mettre en œuvre et vérifiant ces caractéristiques est celle sous forme de graphe complet. Dans un premier temps, nous allons présenter de manière

informelle le passage du graphe utilisateur de l'objet au graphe complet. Dans un second temps, nous énoncerons les différentes règles mises en œuvre ainsi que la preuve formelle de la terminaison du processus de normalisation permettant d'obtenir le graphe complet normalisé.

2.2.1. Passage du graphe utilisateur au graphe complet

Pour toute pièce, il existe plusieurs graphes équivalents la représentant (figure 2.4). Chaque graphe modélisant la pièce peut être transformé, par un ensemble de règles, en un graphe normalisé unique, identique quel que soit le graphe de départ.

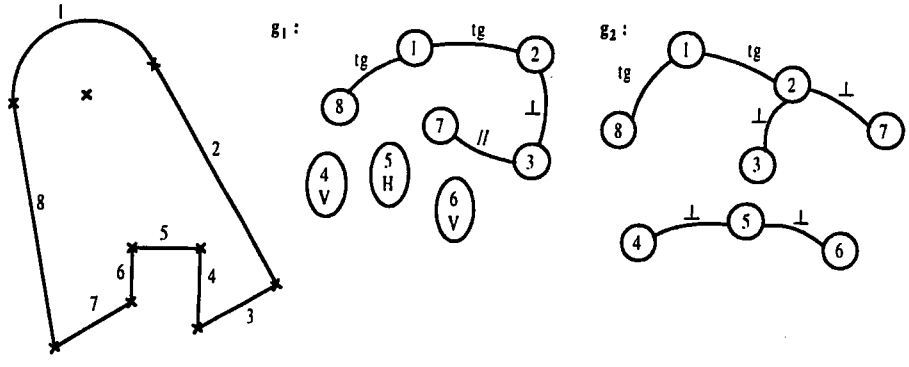


Figure 2.4. *Deux graphes différents pour la même pièce*

Pour une même pièce, on conservera donc deux graphes : le graphe utilisateur et le graphe normalisé complet. Il est important de conserver le graphe utilisateur même s'il n'est pas utilisé pour la recherche d'égalité ou d'inclusion avec d'autres pièces afin de maintenir les préférences et les choix du concepteur. Lors d'une modification et donc d'une éventuelle propagation de contraintes, il sera important de modifier les différents paramètres de la pièce en privilégiant la méthode sous-jacente donnée par l'utilisateur du système pendant la phase de conception.

Le paragraphe précédent a montré qu'il peut exister plusieurs graphes représentant la même situation. Ceci est dû au fait que les contraintes entre plusieurs objets peuvent être combinées différemment afin de fournir le même résultat géométrique. Cette remarque rejoint la notion de représentation "multi-modèles" actuellement étudiée en CAO.

Une représentation unique d'une même scène contrainte est de conserver dans le graphe toutes les contraintes : celles exprimées par l'utilisateur ainsi que celles pouvant ensuite être déduites. Les contraintes déduites sont redondantes et sont issues de l'application d'un ensemble de règles ayant comme hypothèse une combinaison de contraintes et comme résultat une nouvelle contrainte. La figure 2.5 représente le graphe complet de la pièce de la figure 2.4.

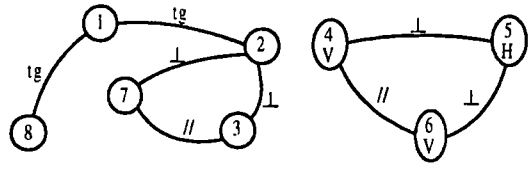


Figure 2.5. *Graphe complet*

L'ensemble des règles, dans le cas de la géométrie 2D que nous avons choisie d'étudier, consiste en une réécriture de tous les théorèmes de la géométrie euclidienne portant sur les contraintes étudiées. Cet ensemble de théorèmes étant complet, on peut admettre, dans ce cas bien précis, que la complétude de l'ensemble des règles est immédiate. Le problème est évidemment bien plus compliqué en 3D et n'est pas abordé dans ce document. En effet, cette extension qui constitue un thème de recherche à part entière, mériterait une étude approfondie qui nous éloignerai beaucoup de notre préoccupation actuelle.

2.2.2. Règles de production pour un graphe complet

En tout nœud du graphe de contraintes, les contraintes sont représentées soit par des arêtes de liaison entre deux nœuds (contraintes binaires), soit par des caractéristiques propres au nœud lui-même (contrainte unaire). L'ensemble des contraintes associées à un objet pourra être complété afin de fournir un ensemble complet.

La représentation de l'objet sera figée, de sorte que, quelle que soit la méthode de création des contraintes choisie par l'utilisateur, l'application de l'ensemble des règles de production conduit au même graphe normalisé complet. En effet, la méthode de production garantit l'ajout systématique dans le graphe de contraintes non définies à la construction par l'utilisateur. Par exemple, dans le cas de deux droites parallèles D_1 et D_2, et d'une droite D_3 perpendiculaire à D_1, le système ajoutera la contrainte binaire de perpendicularité entre D_3 et D_2 (figure 2.6).

Figure 2.6. *Vers un graphe complet*

L'ensemble des règles de production défini et appliqué aux différents nœuds du graphe des contraintes fournit de nouvelles contraintes. Le système itère le processus afin qu'aucune contrainte redondante ne soit oubliée. Si cette situation se présentait, le système serait dans

certains cas incapable d'affirmer l'égalité ou l'inclusion de pièces, ce qui aurait pour conséquence de présenter à l'utilisateur des pièces inutilement.

Dans le cadre de notre étude, le système de règles de production est directement déduit des théorèmes de géométrie euclidienne. Nous présentons ci-dessous, l'ensemble des règles de production correspondant aux objets et aux contraintes que nous avons choisis de manipuler pour réaliser notre étude. Ces contraintes, quoique propriétés géométriques simples, permettent un nombre de combinaisons très important. Les différentes règles de production sont :

R_1 D_1 Horizontal $\wedge$ D_2 Horizontal $\Rightarrow$ $D_1 // D_2$

R_2 D_1 Vertical $\wedge$ D_2 Vertical $\Rightarrow$ $D_1 // D_2$

R_3 D_1 Horizontal $\wedge$ D_2 Vertical $\Rightarrow$ $D_1 \perp D_2$

R_4 D_1 Vertical $\wedge$ D_2 Horizontal $\Rightarrow$ $D_1 \perp D_2$

R_5 D_1 Horizontal $\wedge$ $(D_1 // D_2)$ $\Rightarrow$ D_2 Horizontal

R_6 D_1 Horizontal $\wedge$ $(D_1 \perp D_2)$ $\Rightarrow$ D_2 Vertical

R_7 D_1 Vertical $\wedge$ $(D_1 // D_2)$ $\Rightarrow$ D_2 Vertical

R_8 D_1 Vertical $\wedge$ $(D_1 \perp D_2)$ $\Rightarrow$ D_2 Horizontal

R_9 $(D_1 // D_2) \wedge (D_1 \perp D_3)$ $\Rightarrow$ $D_2 \perp D_3$

R_{10} $(D_1 \perp D_2) \wedge (D_1 // D_3)$ $\Rightarrow$ $D_2 \perp D_3$

R_{11} $(D_1 \perp D_2) \wedge (D_2 \perp D_3)$ $\Rightarrow$ $D_1 // D_3$

R_{12} $(D_1 \perp D_2) \wedge (D_2 \perp D_3) \wedge (D_1 \perp D_4)$ $\Rightarrow$ $D_3 \perp D_4$

R_{13} $(D_1 // D_2) \wedge (D_1 // D_3)$ $\Rightarrow$ $D_2 // D_3$

Nous allons montrer que la stabilité du graphe est atteinte, lorsque l'application de l'ensemble des règles de production sur le graphe courant n'engendre pas la création de nouvelles relations. Le graphe considéré est alors la représentation normalisée sous forme de graphe complet de la situation modélisée.

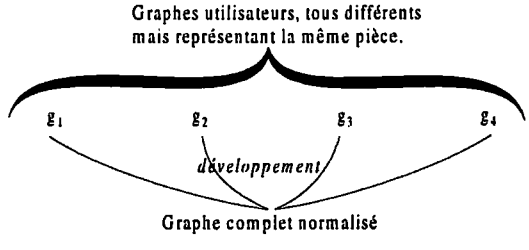


Figure 2.7. *Première étape de la normalisation*

On appelle Σ l'ensemble des relations entre les différents éléments de la scène, cet ensemble est défini de la façon suivante :

Σ = { (d_i), (d_{ij}) i,j $\in$ N / d_i est une relation unaire sur l'objet i et d_{ij} est une relation binaire entre les objets i et j }

Soit Σ_n l'ensemble des relations entre les différents éléments de la scène à l'étape n du processus de normalisation. Lors de l'application de l'ensemble des règles de production en un nœud, cet ensemble est transformé en un ensemble Σ_{n+1}. Compte tenu de la nature des règles appliquées à l'ensemble Σ_n, l'ensemble Σ_{n+1} ne peut être constitué que des éléments de l'ensemble Σ_n auxquels certaines relations ont éventuellement été ajoutées. On construit ainsi une suite d'ensembles $(\Sigma_n)_{n\in N}$. Du point de vue de l'application, il faut montrer que le système de production se stabilise, c'est-à-dire qu'il ne produit plus de nouvelles contraintes. Ceci revient à montrer que la suite ainsi définie atteint un maximum. De cette façon, il existera un $k\in N$ tel que Σ_{k+1} sera égal à Σ_k (pas d'ajout de nouvelles relations), la stabilité est atteinte et Σ_{k+1} représente l'état final de la scène normalisée.

Afin de pouvoir comparer deux éléments d'une suite, une relation d'ordre est utilisée ; l'inclusion d'ensembles au sens ensembliste du terme, notée $\subset$, est considérée. Ainsi, un ensemble est inclus dans un autre si toutes les contraintes du premier appartiennent au second. Munie de cette relation d'ordre et du système de production de règles, la suite $(\Sigma_n)_{n\in N}$ se présente comme une suite strictement croissante.

Montrons à présent que la suite $(\Sigma_n)_{n\in N}$ est majorée. Soit n le nombre d'objets de base dans le graphe des contraintes, k_1 le nombre de relations unaires possibles et k_2 celui des relations binaires ; dans le cas le plus défavorable, chaque objet du graphe pourra être caractérisé par toutes les relations unaires, soit au total $k_1 \times n$. Il aura également toutes les relations binaires avec les autres objets, soit $k_2 \times (n-1)$ relations pour le premier nœud, $k_2 \times (n-2)$ relations pour le deuxième, etc., c'est-à-dire au total $k_2 \times n!$. Le majorant est alors égal à $k_1 \times n + k_2 \times n!$.

En conclusion, la suite $(\Sigma_n)_{n\in N}$ est discrète, croissante et majorée. Il existe ainsi un entier k tel que Σ_k soit égal à la limite de la suite $(\Sigma_n)_{n\in N}$, Σ_k représente l'unique graphe normalisé complet modélisant la situation considérée. Nous venons ainsi de prouver la terminaison de l'algorithme de normalisation ainsi que l'unicité de la solution.

Dans le cadre de notre application, le majorant peut être réduit à n+2n! compte tenu de certaines caractéristiques des contraintes considérées. En effet, les contraintes internes horizontal et vertical ne peuvent être affectées simultanément à un même objet ce qui limite à un le nombre de contraintes internes par objet. De manière similaire, les deux contraintes binaires parallèle et

perpendiculaire étant antagonistes, elles ne peuvent pas mettre en relation le même couple d'objets. Deux objets peuvent ainsi être liés par au maximum deux relations : une contrainte de tangence et soit une contrainte de parallélisme, soit une contrainte de perpendicularité. Cette remarque nous permet de réduire le majorant à n + 2n!.

2.3. Simplification du graphe normalisé complet

Le graphe complet obtenu après l'application de l'algorithme de normalisation est la représentation normalisée unique de la scène décrite par le graphe. Néanmoins ce graphe étant complet, il contient un certain nombre de redondances qui grossissent la structure de modélisation et saturent ainsi inutilement la mémoire ; de plus, elles compliquent les différents algorithmes nécessitant des parcours du graphe.

Certaines règles permettent de simplifier le graphe complet en supprimant les relations redondantes. Néanmoins la suppression des relations redondantes doit tenir compte des situations ambiguës ; ces dernières sont généralement dues aux règles de symétrie ou de transitivité.

Afin d'éliminer ce type de situations, il convient de supprimer uniquement les relations qui n'introduisent aucune ambiguïté. Un ordre de priorité est ainsi affecté aux contraintes topologiques et géométriques ; cet ordre est directement déduit des propriétés mathématiques et géométriques. Par exemple, si un segment S_1 est horizontal et un segment S_2 est vertical alors les deux segments sont perpendiculaires. Cependant, il est inutile d'inclure cette relation au graphe de contraintes car elle n'apporte aucune précision à la définition de la pièce.

Dans l'optique de définir une méthode de simplification du graphe complet en un graphe minimal unique, un ensemble de règles permettant de choisir sans ambiguïté entre deux informations redondantes, est défini. Ce système de règles permet d'obtenir à partir du graphe complet de la pièce, un graphe minimum identique quel que soit le graphe initial (figure 2.8).

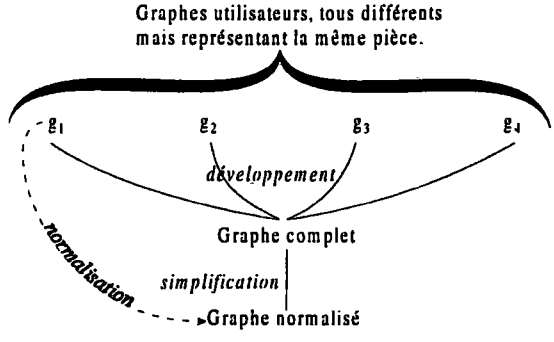


Figure 2.8. *Les différentes étapes de la normalisation*

Pour écrire le système de règles de suppression permettant de réduire le graphe complet, des priorités sur les contraintes manipulées sont prises en considération. C'est pourquoi, dans le cadre de notre domaine d'application, le système de règles privilégie le maintien des contraintes internes d'horizontalité et de verticalité. En effet, en plus des contraintes induites telles que le parallélisme entre deux segments verticaux, une information de positionnement du segment par rapport aux axes est sous-jacente. Ensuite, les règles de suppression tendent à conserver de préférence les contraintes de perpendicularité plutôt que des contraintes de parallélisme notamment parce que les contraintes de parallélisme conduisent à des redondances inévitables.

Explicitons cette dernière remarque ; lorsque trois droites D_1, D_2 et D_3 sont parallèles, deux relations parmi $D_1//D_2$, $D_1//D_3$ et $D_2//D_3$ suffisent à représenter la situation. Or, nous ne disposons pas d'une méthode fiable permettant de décider de la suppression dans tous les cas de la même relation.

Les règles sont donc les suivantes :

R_1	$(D_1 // D_2) \wedge D_1$ Horizontal $\wedge D_2$ Horizontal	$\Rightarrow$ suppression de $(D_1 // D_2)$
R_2	$(D_1 // D_2) \wedge D_1$ Vertical $\wedge D_2$ Vertical	$\Rightarrow$ suppression de $(D_1 // D_2)$
R_3	$(D_1 \perp D_2) \wedge D_1$ Vertical $\wedge D_2$ Horizontal	$\Rightarrow$ suppression de $(D_1 \perp D_2)$
R_4	$(D_1 \perp D_2) \wedge D_1$ Horizontal $\wedge D_2$ Vertical	$\Rightarrow$ suppression de $(D_1 \perp D_2)$
R_5	$(D_1 // D_2) \wedge (D_1 \perp D_3) \wedge (D_3 \perp D_4)$	$\Rightarrow$ suppression de $(D_1 // D_2)$

La suppression inexacte d'une relation de parallélisme mettrait en défaut la propriété d'unicité du graphe normalisé. Pour éviter cet inconvénient, une information redondante est conservée dans le graphe de façon à ne pas remettre en cause les propriétés du graphe normalisé.

Cette méthode souffre toutefois de deux handicaps principaux qui sont :

- la perte de temps due au passage par un graphe complet ; un certain nombre de relations sont ajoutées par le processus de développement pour être supprimées juste après par le processus de simplification ;
- la taille des structures manipulées ; le processus de normalisation est obligé de travailler sur le graphe complet de l'objet c'est-à-dire qu'il manipule le maximum de données possibles.

De plus, tout ajout ou retrait de contraintes oblige à réappliquer les différentes étapes du processus de normalisation, il est donc intéressant de connaître la fin exacte de la phase de conception afin d'appliquer la simplification sur une construction stable. Quant à la perte de

temps, elle est directement liée au nombre d'objets de la scène construite ; dans le cas le plus défavorable, l'algorithme de simplification est en O(n!).

Afin de résoudre ces problèmes, l'idée développée dans la partie suivante est d'appliquer une simplification au cours de la phase de normalisation. En effet, à chaque fois qu'un élément est construit, il serait judicieux de ne conserver dans le modèle normalisé que les contraintes nécessaires et d'éliminer au fur et à mesure les contraintes redondantes. Notons que le graphe utilisateur conserve toujours en parallèle toutes les contraintes demandées par le concepteur ; ce graphe reste présent dans le modèle afin de ne pas dérouter l'utilisateur pendant la phase de conception lors de la propagation d'une modification par exemple. Dans tous les cas, le graphe utilisateur reste intact de façon à pouvoir servir de référence comme nous l'avons déjà précisé.

2.4. Normalisation avec ajout et remplacement simultanés

Précédemment, nous avons présenté une méthode permettant d'obtenir une représentation unique et minimale d'une pièce. Cette méthode de normalisation du graphe utilisateur se décompose en deux étapes, une première étape de développement du graphe utilisateur conduisant au graphe complet suivi d'une étape de simplification de ce graphe maximal.

Dans cette partie, nous exposons une méthode basée sur un système de règles de remplacement et d'ajout permettant d'obtenir directement le graphe normalisé restreint. L'idée générale est de combiner les règles de production et de substitution afin de les appliquer à chaque étape de la conception aux nœuds du graphe. Ainsi, pendant la phase de conception, une suite d'ensembles de contraintes est construite et de la convergence de cette suite, découle la terminaison du processus de normalisation et l'unicité de la solution.

Comme nous l'avons démontré au paragraphe 2.2.2, la convergence de la suite constituée des ensembles de contraintes créés lors du processus de normalisation est assurée lorsque la suite est croissante et majorée. Or, si l'ajout de contraintes ne pose aucun problème de convergence, la suppression de contraintes par des règles rend plus difficile la définition de cette notion. La solution envisagée est de ne pas supprimer ces contraintes mais de les remplacer par d'autres.

Compte tenu des priorités des contraintes déjà énoncées, considérons l'exemple suivant : deux droites D_1 et D_2 sont parallèles et une droite D_3 est construite perpendiculaire à D_1. La contrainte de parallélisme est alors remplacée par une contrainte de perpendicularité entre D_2 et D_3.

Les règles de remplacement sont ainsi judicieusement choisies afin d'assurer une croissance qu'il reste encore à définir.

2.4.1. Croissance de la suite $(\Sigma_n)_{n\in N}$

La croissance de la suite $(\Sigma_n)_{n\in N}$ est prouvée par la définition possible d'une relation d'ordre entre deux éléments consécutifs Σ_k et Σ_{k+1} de la suite $(\Sigma_n)_{n\in N}$. Dans cette partie, nous allons définir une relation d'inclusion spécifique aux ensembles de contraintes manipulés ; nous nous attacherons à montrer que cette relation d'inclusion est une relation d'ordre. Toutefois, avant de réaliser cette démonstration, nous sommes obligés de passer par une phase assez fastidieuse d'énoncer de définitions et de lemmes indispensables.

Nous précisons la définition de Σ, l'ensemble de relations du graphe de manière à pouvoir :
- distinguer les relations unaires des relations binaires ;
- tenir compte des priorités des contraintes.

Définition 1. Soit p_i et p_i' deux entiers représentant les priorités respectives de deux relations d_i et d_i', on dira que d_i est moins prioritaire que d_i' si $p_i<p_i'$.

Définition 2. On définit Σ l'ensemble des relations du graphe par l'union de l'ensemble Σ_U des relations unaires et de l'ensemble Σ_B des relations binaires :

$\Sigma_U = \{ (c_i,p_i)$ où c_i est une contrainte unaire sur l'objet $_i$, de priorité $p_i \}$

$\Sigma_B = \{ (c_{ij},p_{ij})$ où c_{ij} une contrainte binaire entre les objets i et j, de priorité $p_{ij} \}$

$\Sigma = \Sigma_U \cup \Sigma_B$

D'une manière similaire au paragraphe 2.2.2, on appelle Σ_n l'ensemble des relations entre les différents éléments de la scène à l'étape n du processus de normalisation. Σ_n est l'union de l'ensemble des contraintes unaires et contraintes binaires à l'étape n.

Lors de l'application de l'ensemble des règles de production en un nœud, cet ensemble est transformé en un ensemble Σ_{n+1}. Σ_{n+1} est obtenu :
- en conservant la plupart des relations Σ_n ;
- en remplaçant certaines relations par d'autres équivalentes mais plus prioritaires ;
- en ajoutant des nouvelles déduites des nouvelles relations spécifications de la pièce.

Ces différentes étapes permettent la construction d'une suite d'ensembles $(\Sigma_n)_{n\in N}$. Nous allons montrer que le système se stabilise, il ne produit donc plus de nouvelles règles ; ceci revient à montrer que la suite ainsi définie converge et atteint un extremum. Cette conclusion prouve

l'existence d'un entier k tel que Σ_{k+1} soit égal à Σ_k ; Σ_{k+1} ne subit ni ajout, ni remplacement de relation. La stabilité de la suite est atteinte et Σ_k représente l'état final de la scène normalisée.

L'égalité d'ensembles de contraintes est obtenue lorsque l'on retrouve dans les deux ensembles exactement les mêmes contraintes unaires et les mêmes contraintes binaires. Cette notion intuitive peut être formalisée et permet d'écrire alors la définition ci-dessous.

Définition 3. Soit B l'ensemble des valeurs booléennes ; on définit l'égalité de deux ensembles de contraintes Σ, par la relation $= : \quad \Sigma \times \Sigma \rightarrow B$

telle qu'il existe une application $\varphi : \Sigma \rightarrow \Sigma$

- $\forall$ contrainte $(d_i,p_i) \in \Sigma_{1U}$, $\exists$ unique contrainte $(d_i',p_i') \in \Sigma_{2U}$, $\varphi(d_i,p_i)=(d_i',p_i')$
  vérifiant : $d_i=d_i'$ et $p_i=p_i'$ et réciproquement ($\varphi_{/U}$ est une bijection)
- $\forall$ contrainte $(d_{ij},p_{ij}) \in \Sigma_{1B}$, $\exists$ unique contrainte $(d_{ij}',p_{ij}') \in \Sigma_{2B}$, $\varphi(d_{ij},p_{ij})=(d_{ij}',p_{ij}')$
  vérifiant : $d_{ij}=d_{ij}'$ et $p_{ij}=p_{ij}'$ et réciproquement ($\varphi_{/B}$ est une bijection)

Lemme 1. Soit deux ensembles Σ_1 et Σ_2 tels que $\Sigma_1=\Sigma_2$ alors Card (Σ_1) = Card (Σ_2)

Lemme 2. L'égalité notée = est une relation d'équivalence.

Intuitivement, un ensemble Σ_1 de contraintes est inclus dans un ensemble Σ_2 de contraintes si :

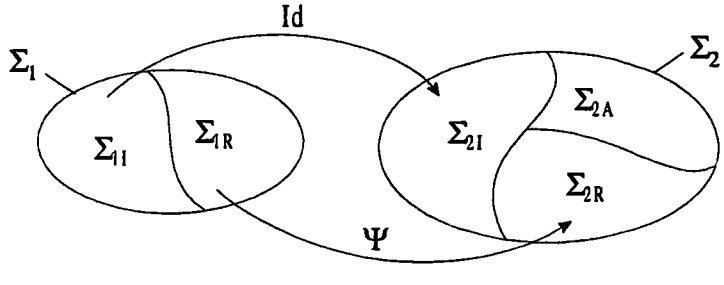
- une partie des contraintes de Σ_1 se retrouve inchangée dans Σ_2 ;
- les contraintes restantes de Σ_1 sont chacune remplacées par une nouvelle contrainte plus importante dans Σ_2 ;
- Σ_2 contient un certain nombre de nouvelles relations.

On peut formaliser cette définition de la façon suivante.

Définition 4. On définit l'inclusion de deux ensembles par la relation :

$\subseteq : \quad \Sigma \times \Sigma \rightarrow B$

tel qu'il existe une partition de l'ensemble Σ_1 (Σ_{1I},Σ_{1R}) et une partition de l'ensemble Σ_2 (Σ_{2I}, Σ_{2R},Σ_{2A}) (Σ_{2I} est le sous-ensemble des contraintes qui n'ont pas été modifiées, Σ_{2R} le sous-ensemble des contraintes modifiées et Σ_{2A} le sous-ensemble des contraintes ajoutées).



Ces partitions vérifient :

- Il existe une relation d'identité Id entre Σ_{1I} et Σ_{2I}
- $\exists$ bijection $\psi : \Sigma_{1R} \rightarrow \Sigma_{2R}$

$(d,p_i) \in \Sigma_{1R} \rightarrow (d',p_i') \in \Sigma_{2R}$ tq : $p_i < p_i'$ où d et d' sont des contraintes

Lemme 3. $\Sigma_1 \subseteq \Sigma_2 \Rightarrow Card(\Sigma_1) \leq Card(\Sigma_2)$

Propriété 1. La relation $\subseteq$ est une relation d'ordre.

Démonstration. Montrons que cette relation est réflexive, antisymétrique et transitive.

a. Réflexivité. ? $\Sigma_1 \subseteq \Sigma_1$

Démonstration : $\Sigma_1 = Id(\Sigma_{1I})$ et $\Sigma_{1R} = \varnothing$ et $\Sigma_{1A} = \varnothing$

b. Antisymétrie. ? $\Sigma_1 \subseteq \Sigma_2$ et $\Sigma_2 \subseteq \Sigma_1 \Rightarrow \Sigma_1 = \Sigma_2$

Démonstration : d'après le lemme 3,

$\Sigma_1 \subseteq \Sigma_2 \Rightarrow Card(\Sigma_1) \leq Card(\Sigma_2)$ et $\Sigma_2 \subseteq \Sigma_1 \Rightarrow Card(\Sigma_2) \leq Card(\Sigma_1)$

donc $Card(\Sigma_1) = Card(\Sigma_2) \Rightarrow \Sigma_{2A} = \varnothing$

Montrons que $\Sigma_{1R} = \Sigma_{2R} = \varnothing$

Soit $\Sigma_1 \subseteq \Sigma_2$, notons Ψ_1 l'application bijective: $\Sigma_{1R} \rightarrow \Sigma_{2R}$

Il existe $(d_{k1},p_{k1}) \in \Sigma_{1R}$ tel que p_{k1} minimum et $(d_{k2},p_{k2}) \in \Sigma_{2R}$ tel que p_{k2} minimum

$p_{k2} > p_{k1}$ car $\exists i$ / $(d_{k2},p_{k2}) = \Psi^{-1}(d_i,p_i)$ et $p_{k2} > p_i > p_{k1}$

De même si $\Sigma_2 \subseteq \Sigma_1 \Rightarrow p_{k1} > p_{k2}$ ce qui est impossible donc il n'existe aucun élément dans Σ_{1R}, ni dans $\Sigma_{2R} \Rightarrow \Sigma_1 = \Sigma_{1I}$ et $\Sigma_2 = \Sigma_{2I} \Rightarrow \Sigma_1 = \Sigma_2$

c. Transitivité. ? $\Sigma_1 \subseteq \Sigma_2$ et $\Sigma_2 \subseteq \Sigma_3 \Rightarrow \Sigma_1 \subseteq \Sigma_3$

Démonstration : On construit une partition Σ_3 (σ_{3I}, σ_{3R}, σ_{3A}) à partir des éléments de Σ_1.

σ_{3I} ={Id o Id (x) avec $x \in \Sigma_1$}={Id o Id (x) pour tout x de σ_{1I}}

σ_{3R} ={Id o Ψ (x) avec $x \in \Sigma_1$} $\cup$ { Ψ o Id (x) avec $x \in \Sigma_1$ }$\cup${ Ψ o Ψ (x) avec $x \in \Sigma_1$ }

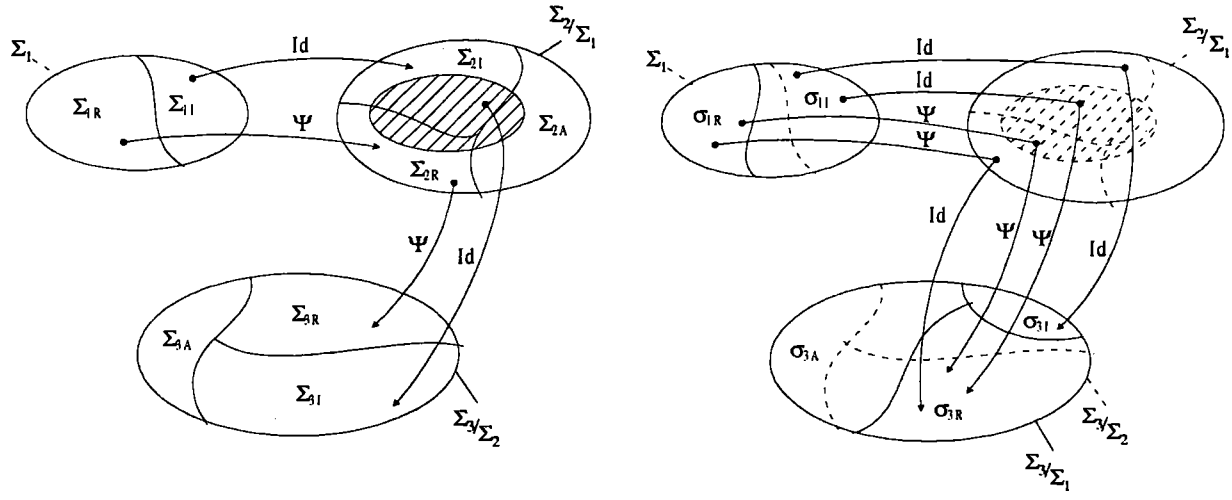
$\sigma_{3A} = \Sigma_3 - \sigma_{3I} - \sigma_{3R}$

Par construction $\Sigma_3 = \sigma_{3I} \cup \sigma_{3R} \cup \sigma_{3A}$

Montrons à présent que les intersections deux à deux sont disjointes :

- $\sigma_{3A} \cap \sigma_{3I} = \varnothing$ car σ_{3A} est obtenue en retirant de Σ_3 tous les éléments de σ_{3I} ;
- $\sigma_{3A} \cap \sigma_{3R} = \varnothing$ car σ_{3A} est obtenue en retirant de Σ_3 tous les éléments de σ_{3R} ;
- $\sigma_{3I} \cap \sigma_{3R} = \varnothing$ par construction car l'un contient les éléments inchangés et l'autre les éléments ayant été remplacés.

L'application $\varphi : \sigma_{1R} \rightarrow \sigma_{3R}$ est définie ; φ est une bijection car elle est la composition de bijections (Id o Ψ, Ψ o Id ou Ψ o Ψ) telle que $\forall(d,p) \in \sigma_{1R}, \varphi(d,p)$ a une priorité strictement supérieure à p. De plus comme Id : $\sigma_{1I} \rightarrow \sigma_{3I}$, l'inclusion de Σ_1 dans Σ_3 est définie.



Les lignes en pointillés correspondent aux partitions construites pour $\Sigma_1 \subseteq \Sigma_2$ et $\Sigma_2 \subseteq \Sigma_3$

Figure 2.9. *Transitivité de la relation* $\subseteq$

2.4.2. Majorant de la suite $(\Sigma_n)_{n\in N}$

Dans le cas le plus défavorable, le système ne fait qu'ajouter des contraintes et le majorant est alors le même que dans le cas de règles de production simple, soit $k_1 \times n + k_2 \times n!$ où k_1 est le nombre de contraintes unaires possibles, k_2 le nombre de contraintes binaires possibles et n le nombre d'objets dans le graphe. Donc la suite $(\Sigma_n)_{n\in N}$ est majorée.

2.4.3. Règles de production et de remplacement

Le système de règles appliquées au graphe peut laisser des contraintes inchangées ou remplacer des contraintes existantes par des contraintes de priorité supérieure ou encore ajouter de nouvelles contraintes. Si le système de règles respecte scrupuleusement ces contraintes, on assure la croissance de la suite ; en effet, appliqué à un ensemble de contraintes Σ_n, le système produira un ensemble Σ_{n+1} qui, par construction, sera plus grand (au sens de l'inclusion définie au paragraphe précédent). Ci-dessous, sont présentées en exemple, certaines règles qui peuvent être appliquées au graphe ($\succ$ signifie "est remplacé par") :

R_1 $(D_1 // D_2) \wedge D_1$ Horizontal $\Rightarrow (D_1 // D_2) \succ D_2$ Horizontal

R_2 $(D_1 // D_2) \wedge D_1$ Vertical $\Rightarrow (D_1 // D_2) \succ D_2$ Vertical

R_3 $(D_1 \perp D_2) \wedge D_1$ Vertical $\Rightarrow (D_1 \perp D_2) \succ D_2$ Horizontal

R_4 $(D_1 \perp D_2) \wedge D_1$ Horizontal $\Rightarrow (D_1 \perp D_2) \succ D_2$ Vertical

R_5 $(D_1 // D_2) \wedge (D_1 \perp D_3) \Rightarrow (D_1 // D_2) \succ (D_2 \perp D_3)$

R_6 $(D_1 \perp D_2) \wedge (D_2 \perp D_3) \wedge (D_1 \perp D_4) \Rightarrow (D_3 \perp D_4)$

R_7 $(D_1 // D_2) \wedge (D_2 // D_3) \Rightarrow (D_3 // D_4)$

Étant donné que la suite est discrète, on peut affirmer qu'il existe un $n \in N$ tel que la suite atteint sa limite en ce point. La limite étant unique, le système de règles aura fourni un unique graphe de contraintes normalisé.

2.4.4. Distinction contrainte implicites / contraintes explicites

Afin de pouvoir découvrir les similitudes de pièces construites de manière différente, deux types de contraintes vont être distinguées : les contraintes implicites et les contraintes explicites. Les contraintes explicites sont fixées par le concepteur (menu) et doivent absolument être vérifiées contrairement aux contraintes implicites qui sont déduites par le système à partir d'éléments non contraints a priori. Ces contraintes implicites sont considérées comme des préférences de l'utilisateur alors que les contraintes explicites représentent des obligations.

Cette distinction va nous permettre de trouver une égalité entre deux pièces même si les deux conceptions n'ont pas privilégié les mêmes contraintes. Nous notons avec un indice i ($//_i$, $Horizontal_i$, ...) les contraintes implicites et avec un indice e ($\perp_e$, $Vertical_e$, ...) les contraintes explicites.

La figure 2.10 montre deux pièces identiques mais dont la construction est différente. En effet, les contraintes fixées lors des deux conceptions divergent, les deux graphes de contraintes ne sont donc pas égaux, sauf si on fait abstraction de la distinction contrainte implicite et explicite.

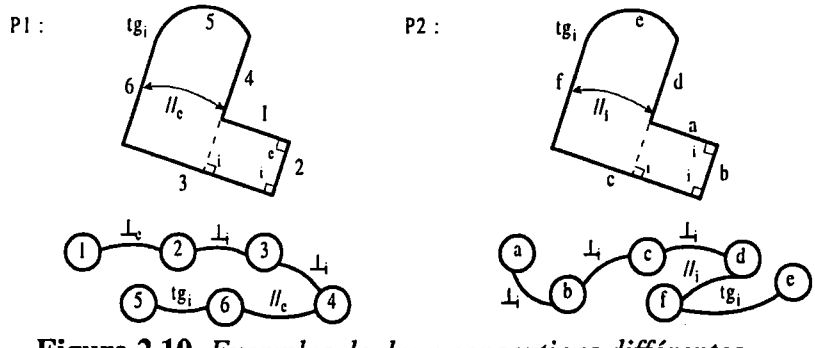


Figure 2.10. *Exemples de deux conceptions différentes*

Les règles de normalisation restent inchangées lorsque les différentes relations impliquées sont de même type. Par contre, des règles mixtes apparaissent, faisant intervenir à la fois des contraintes implicites et explicites.

Exemples de règles mixtes :

$(D_1 //_i D_2) \wedge (D_1 \perp_e D_3) \Rightarrow (D_1 //_i D_2)$ est remplacé par $(D_3 \perp_i D_2)$

$(D_1 //_e D_2) \wedge (D_1 \perp_i D_3) \Rightarrow (D_3 \perp_i D_2)$ est ajoutée

Dans les règles mixtes, les relations explicites sont toujours conservées car elles ont été imposées par l'utilisateur. Dans les deux règles mixtes ci-dessus, la droite D_1 étant parallèle à la droite D_2 et perpendiculaire à la droite D_3, on peut déduire que les droites D_2 et D_3 sont perpendiculaires. Dans la première règle, la contrainte de parallélisme est implicite, elle peut donc être remplacée par une contrainte de perpendicularité également implicite. Par contre, dans la seconde règle, la contrainte de parallélisme étant explicite, elle est conservée en plus de la contrainte de perpendicularité. En effet, nous ne nous permettons pas de supprimer une contrainte fixée explicitement par le concepteur.

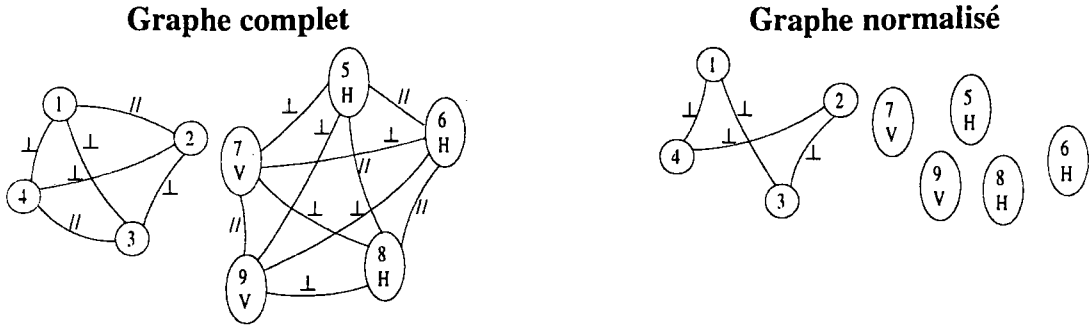
2.5. Synthèse

Nous venons de montrer que pour toute pièce, on peut trouver une modélisation unique et minimale, on entend par minimale, un graphe dont le nombre de nœuds et d'arêtes est minimum. Dans un premier temps, nous avons précisé le type de représentation choisie pour modéliser les différents objets et qui est basée sur l'utilisation de graphes. Trouver un modèle unique pour tout objet consiste donc dans notre application à trouver un graphe unique.

Une première approche de notre étude a consisté à créer le graphe complet de chaque objet qui constitue de manière évidente une représentation unique de l'élément. Cependant, cette structure étant assez lourde à manipuler, nous avons mis en œuvre une méthode permettant de simplifier cette représentation.

Finalement, nous avons proposé une méthode plus directe mais également plus délicate permettant d'obtenir une représentation unique et minimale. Cette méthode permet, en utilisant un système de règles de transformer au fur et à mesure de la conception, le graphe complet en un graphe normalisé restreint.

On peut se demander si la normalisation avec ajouts et remplacements simultanés est plus performante que la méthode qui créée dans un premier temps un graphe complet et qui dans un second temps simplifie ce graphe. La méthode avec ajouts et remplacement simultanés est plus intéressante car elle ne manipule pas le graphe complet qui est une structure de données importante. De plus, elle n'ajoute pas des contraintes dans une première phase qui sont supprimées dans une seconde phase. L'exemple de la figure 2.11 illustre ces propos.



*** Ajout puis remplacement**

Règles d'ajout utilisée pour obtenir le graphe complet sont :
R9, R9, R11, R5, R4, R4, R8, R11, R1, R7, R3, R10, R3

Les règles de suppression pour simplifier le graphe complet et obtenir le graphe normalisé sont :
R5, R5, R1, R3, R3, R4, R1, R1, R3, R3, R2, R3
Nombre total de règles utilisées : 25
Graphe le plus gros manipulé : 9 nœuds, 16 relations

*** Ajout et remplacement simultanés**

Règles d'ajouts et de remplacement simultanés utilisées :
R5, R6, R1, R3, R2

Nombre total de règles utilisées : 5
Graphe le plus gros manipulé : 9 nœuds, 4 relations

Figure 2.11. *Comparaison des méthodes de transformation du graphe complet en un graphe normalisé*

3. COMPARAISONS D'OBJETS DANS UN SYSTÈME DE CAO

La comparaison de modèles est utile dans les systèmes de CAO actuels pour proposer un certain nombre de fonctionnalités qui ont été présentées d'une part dans le premier chapitre de ce document mais également en introduction de ce chapitre. Lorsqu'elle est réalisée, elle utilise la représentation sous forme de graphe normalisé restreint de façon à retrouver, quand elles existent des égalités ou des inclusions de pièces. Dans cette partie, nous proposons deux méthodes permettant de réaliser la comparaison de modèles de pièces ce qui revient, dans le cadre de notre application, à comparer des structures de graphes.

La première méthode proposée est une méthode algorithmique qui gère un certain nombre de filtres permettant d'accélérer le processus de comparaison. La seconde méthode applique le formalisme CSP au problème de recherche d'égalité ou d'inclusion de graphes et dispose d'un certain nombre d'algorithmes de résolution des CSP. L'égalité de deux graphes est prouvée par une instanciation correcte de tous les nœuds du CSP suite à l'application d'un algorithme choisi.

3.1. Méthode algorithmique

La méthode algorithmique proposée procède par essai/erreur avec une remise en question du dernier choix réalisé en cas d'échec. De principe a priori assez simple, les performances de cette

méthode sont améliorées grâce à un certain nombre de tests judicieux, d'heuristiques et de structures de données permettant de guider le système dans sa recherche. Dans cette partie, nous présentons en détail l'égalité de graphes, la méthode d'inclusion s'inspirant fortement de celle d'égalité, elle fera simplement l'objet d'un paragraphe à la fin de cette partie.

3.1.1. Présentation de la méthode

Pour vérifier l'égalité entre deux pièces, on recherche l'isomorphisme de leur graphe de contraintes. Deux graphes sont isomorphes ou égaux, s'ils ont exactement les mêmes nœuds et les mêmes arêtes. On cherche à apparier chaque nœud d'un des graphes avec un nœud de l'autre ; deux nœuds pouvant être associés s'ils sont de même type, s'ils ont les mêmes contraintes internes et le même nombre d'arêtes de même type.

La figure 2.12 présente deux exemples de comparaisons de graphes. L'égalité des deux graphes g_1 et g_2 est vérifiée car tous les nœuds de g_1 peuvent être appariés à un nœud de g_2. En effet, les nœuds (1,2,3,4) de g_1 peuvent être appariés respectivement aux nœuds (d,c,a,b) de g_2 ; ces appariements sont corrects car ils lient des nœuds de même nature et possédant les mêmes contraintes. Par contre, lorsque l'on compare les deux graphes g_3 et g_4, le nœud 4 de g_3 ne peut être associé à aucun nœud de g_4 car il n'existe pas dans g_4 un nœud de type segment et possédant deux contraintes l'une de parallélisme et l'autre de perpendicularité.

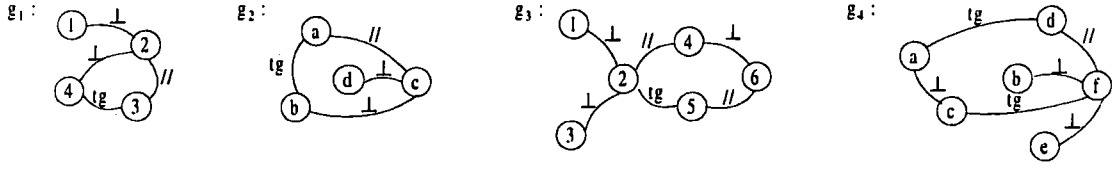


Figure 2.12. *Exemples d'appariement de graphes*

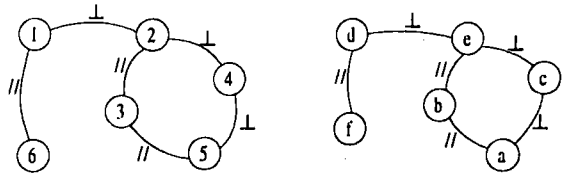
L'algorithme d'appariement est basé sur un parcours en profondeur avec un retour arrière en cas d'impossibilité, le principe de cet algorithme a été présenté pour la première fois par Tarjan [TAR 72]. Le principe général de recherche de l'égalité entre deux graphes est donc un algorithme basé sur le principe de "backtracking" avec recherche d'appariements de nœuds.

L'algorithme de base de la recherche d'appariement de deux graphes consiste à essayer d'associer deux nœuds jusqu'à trouver une association correcte, puis à vérifier si l'appariement de chaque suivant du nœud du premier graphe est possible avec un des suivants du nœud du second

graphe. S'il n'y a pas d'incompatibilité, la même méthode est répétée jusqu'à ce que tout le graphe soit parcouru, sinon les dernières associations réalisées sont détruites (retour arrière) jusqu'à ce que le système ait un autre choix. Le tableau de la figure 2.13 est une application de cet algorithme aux graphes g_1 et g_2 de la même figure. La première colonne de ce tableau représente le nœud de g_1 que l'on tente d'associer à un nœud de g_2 et la deuxième colonne contient les différents candidats à l'appariement qui sont :

- soit des nœuds ayant les mêmes arêtes que le nœud traité ;
- soit des suivants du nœud apparié au nœud précédemment traité.

La troisième colonne contient les nœuds suivants du nœud traité, la quatrième colonne précise l'appariement testé, le résultat de cet appariement figure dans l'avant dernière colonne et la dernière colonne contient quelques explications supplémentaires.



g_1 g_2

Nœud traité	Candidats	Suivants	Appariement	Résultat	Observations
1	a,d	2,6	1↔a	correct	a,d sont retenus car ils ont chacune deux arêtes : une arête ⊥ et une //.
2	b,c	3,4	2↔b 2↔c	échec correct	L'arête (1,2) est une perpendicularité et l'arête (a,b) est une arête de parallélisme.
3	e	5	3↔e	échec	3 et e n'ont pas le même nombre de contraintes. ⇒retour vers les autres candidats de 3 : aucun ⇒retour vers les autres candidats de 2 : 4
4	e	5	4↔e	échec	4 et e n'ont pas le même nombre de contraintes. ⇒retour vers les autres candidats de 4 : aucun ⇒retour vers les autres candidats de 2 : aucun ⇒retour vers les autres candidats de 1 : d
1	d	2,6	1↔d	correct	
2	e,f	3	2↔e	correct	(1,2)=(d,e)=⊥
3	b,c	5	3↔b	correct	(2,3)=(e,b)=//
5	a	4	5↔a	correct	(3,5)=(b,a)=//
4	c	\	4↔c	correct	(5,4)=(a,c)=⊥
6	f	\	6↔f	correct	(1,6)=(d,f)=//

Figure 2.13. *Exemple d'application de l'algorithme d'appariement*

On peut constater que cette méthode est laborieuse, non optimisée et coûteuse en temps de calculs sauf dans le meilleur des cas. Si une destruction ou un retour arrière ne sont pas nécessaires, ce cas optimal est obtenu, c'est-à-dire lorsque chaque tentative d'appariement de nœuds est réussie.

La complexité de cet algorithme peut être évaluée en O(n!). En effet, dans le cas le plus défavorable, seul la dernière arête étudiée permet l'association des nœuds ; et pour un graphe comprenant *n* nœuds, le premier aura *n* choix, le deuxième *n-1*, etc. Dans le meilleur des cas, le premier nœud étudié convient et *n* appariements seront ainsi nécessaires, d'où une complexité optimale en O(n).

Nous allons donc définir des heuristiques et des optimisations permettant de s'approcher de cette configuration optimale.

3.1.2. Optimisation : les classes d'équivalence

Des classes d'équivalence sont définies sur l'ensemble des nœuds d'un graphe en le partitionnant en sous-ensembles d'éléments ayant un certain nombre de points communs. Lors de la recherche d'appariement d'un nœud appartenant à une certaine classe, seul les nœuds de la même classe dans l'autre graphe seront testés, ce qui limite considérablement le nombre de candidats potentiels. La décomposition en classes d'équivalence est intéressante si la taille de chaque partition est raisonnable ; plus le nombre de représentants est faible, plus la probabilité d'associer correctement deux nœuds est grande.

Cette décomposition se fait selon des critères dont la précision détermine le nombre de représentants pour chaque classe. Si le nombre total de représentants n'est pas trop important, le nombre de critères peut rester limité. Par contre, plus le nombre de représentants augmente, plus la décomposition dynamique pourra être complexe et détaillée. On constate donc que la division en classes d'équivalence devrait se faire sur un nombre limité de critères en début de conception et de façon plus détaillée lorsque le nombre de représentants est devenu plus important ; il serait donc intéressant que cette décomposition se fasse de manière dynamique.

Les différentes classes de la partition peuvent être représentées dans une structure d'arbre ; la hauteur de cet arbre modifiera la complexité de l'algorithme d'appariement. Par exemple, dans le cas d'un arbre binaire équilibré de hauteur *p* et en limitant à *n* le nombre de représentants par classe, une complexité moyenne pourra être calculée. Ainsi, la complexité de l'algorithme d'appariement pourra s'exécuter en des temps raisonnables en choisissant une hiérarchie de critères telle que la hauteur *p* de l'arbre des classes d'équivalence et le nombre maximum *n* d'objets par classe fournissent une complexité raisonnable (le quotient $(n!)/2^p$ reste constant au cours du temps)

Dans le cas qui nous concerne, pour les pièces composées de peu d'éléments de base, deux critères de décomposition sont définis. On met en œuvre, successivement, une décomposition suivant le type de l'élément (segment ou cercle) puis une partition de l'ensemble des segments en fonction du nombre d'arêtes du graphe associées au segment (on procède de même pour les cercles). Cette classification des éléments segments et cercles est faite pour chaque composante connexe du graphe de la pièce complète (figure 2.14).

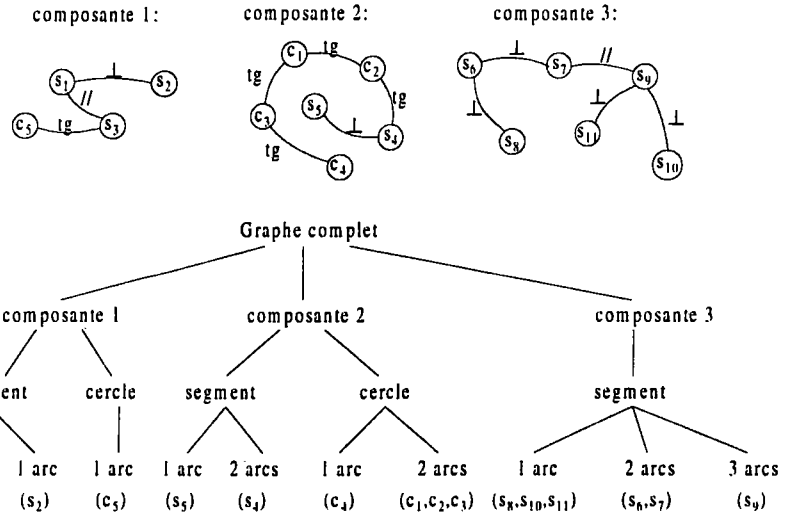


Figure 2.14. *Décomposition en classes d'équivalence d'un graphe constitué de trois composantes connexes.*

De nouveaux critères pourraient être envisagés pour une décomposition plus fine en un nombre plus important de classes d'équivalence :

- considérer le type de l'arête : segment avec deux arêtes perpendiculaire et une arête parallèle au lieu de segment avec trois arêtes ;
- tenir compte en plus, du type du nœud d'arrivée : cercle avec trois tangences, deux avec des segments et une avec un cercle.

Il est cependant difficile de trouver le meilleur choix car il est dépendant du nombre d'objets, du nombre de type d'objets manipulés, de la diversité des contraintes, ...

L'algorithme d'égalité de graphes de base est complété par des tests liés aux classes d'équivalence. Ces tests ainsi que le détail de l'algorithme sont présentés dans la partie suivante.

3.1.3. Algorithme d'égalité de graphes

Nous avons suggéré précédemment l'utilisation de certaines heuristiques et filtres permettant d'accélérer l'algorithme de recherche d'égalité en augmentant la probabilité d'associer

correctement dès le premier essai deux nœuds. L'algorithme développé restreint donc, au fur et à mesure par des tests peu coûteux l'ensemble des nœuds candidats.

Les objets sont classés en fonction de caractéristiques. Dans le cadre de notre application, la méthode consiste tout d'abord à comparer le nombre d'éléments de chaque graphe : s'ils n'ont pas le même nombre de segments ou le même nombre de cercles alors ils sont différents et l'algorithme se termine. S'ils ont le même nombre d'éléments, les différentes composantes connexes sont extraites des deux graphes ; si les deux graphes n'ont pas le même nombre de composantes connexes, à nouveau on peut conclure qu'ils sont différents sinon l'algorithme d'appariement de nœuds est appliqué à chaque composante connexe. L'algorithme complet d'égalité de graphes est résumé par la figure 2.15.

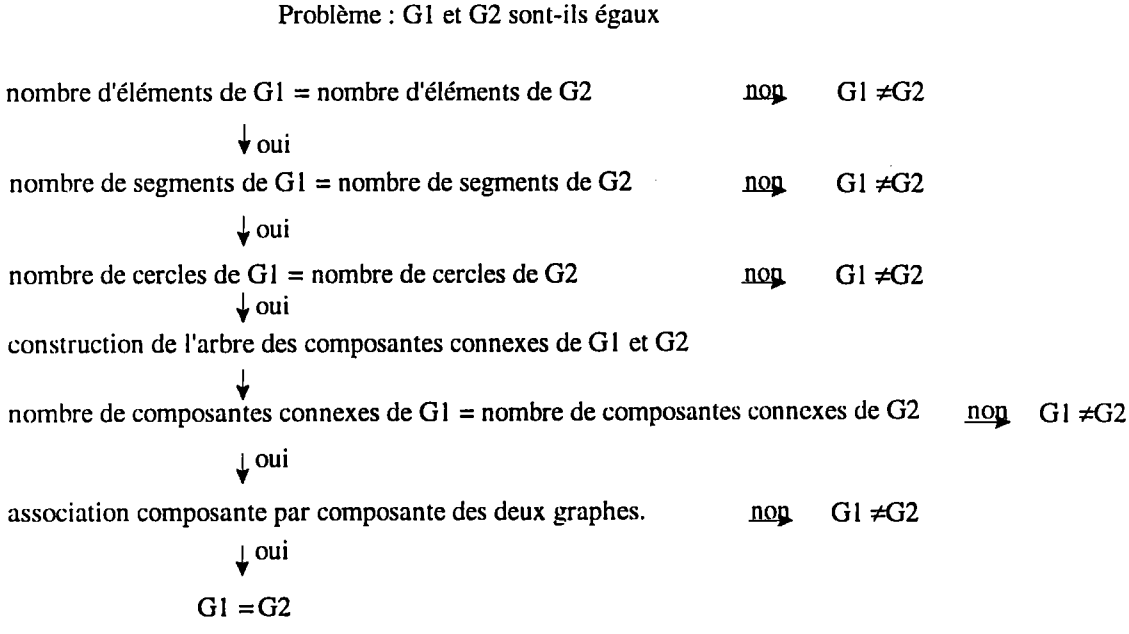


Figure 2.15. *Algorithme d'égalité de deux graphes*

3.1.4. Détermination de pièces communes : inclusion des graphes

Une fonctionnalité très intéressante d'un système de CAO sera de prévenir l'utilisateur lorsque la pièce qu'il réalise a des parties communes avec des pièces conçues précédemment et mémorisées. Pour cela, les graphes des pièces sont comparés en recherchant non plus l'égalité, mais l'inclusion du graphe en construction avec un des graphes de la bibliothèque. Pour l'utilisateur, savoir que la pièce qu'il veut construire ressemble fortement à une pièce déjà construite peut lui permettre :

- d'arrêter sa conception et de réutiliser la pièce mémorisée en y faisant d'éventuelles modifications ;
- d'extraire des sous-parties identiques de la pièce mémorisée ;

- de concevoir de façon différente la même pièce mais de pouvoir réutiliser les fonctionnalités définies après la conception, comme la préparation à la fabrication, les plans de fabrication, les gammes d'usinage, ... ;
- de définir des familles dans la bibliothèque de pièces ayant des points communs.

L'algorithme d'inclusion reprend les mêmes critères de sélection que l'algorithme d'égalité (nombre d'éléments, classe d'équivalence, composante,...), en recherchant non plus l'égalité mais l'inégalité des cardinaux des ensembles déterminés par les différents critères. Figure 2.16 : le graphe de la conception c est inclus dans celui de la pièce b mémorisée ; les deux pièces ont de forts points communs tout en étant différentes.

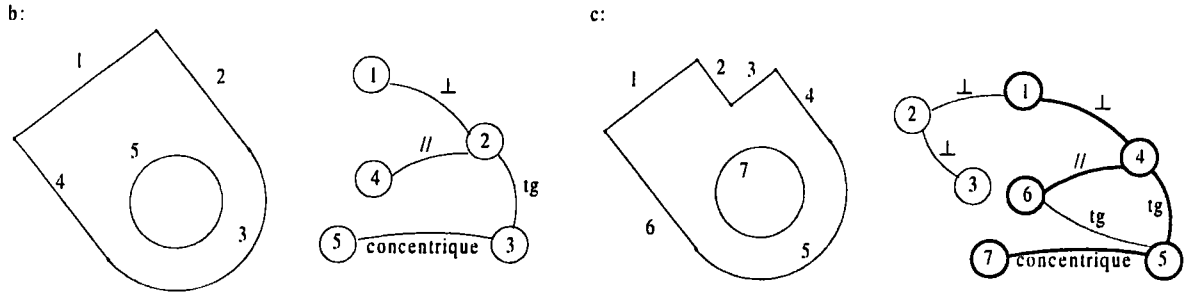


Figure 2.16. *Exemple d'inclusion de graphes*

Le principe de l'algorithme d'inclusion de graphes est résumé par la figure 2.17.

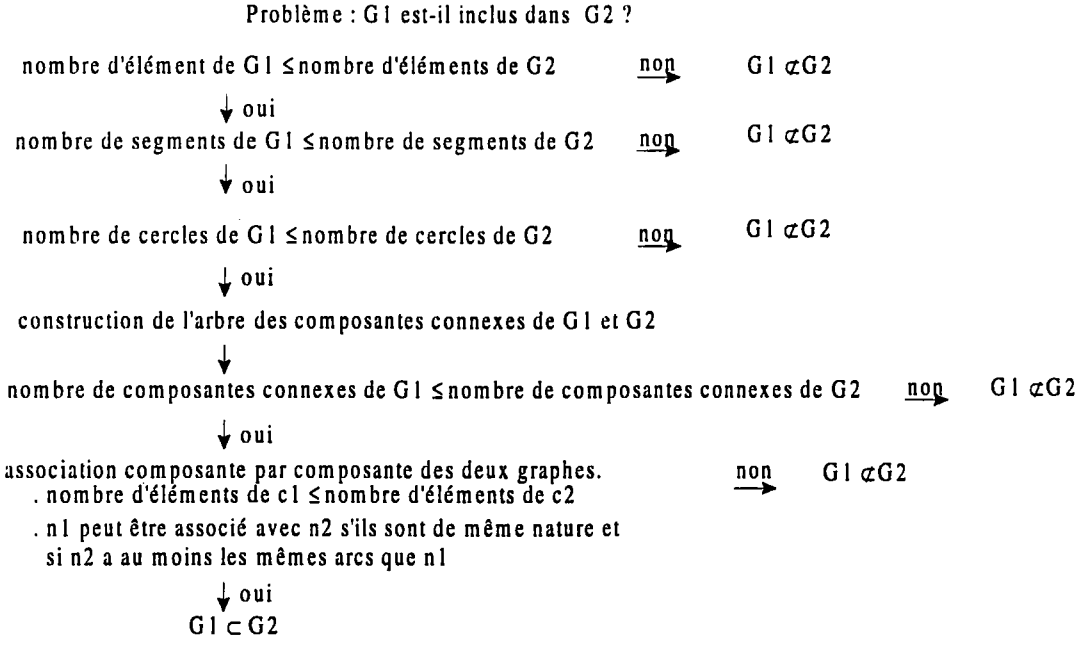


Figure 2.17. *Algorithme d'inclusion de deux graphes*

3.1.5. Synthèse

Cette méthode qui semble a priori relativement simpliste et assez lourde car fonctionnant par essais/erreurs, peut donner toutefois de bons résultats. Dans tous les cas, cette méthode permet de

détecter facilement l'inégalité de deux graphes grâce à un certain nombre de pré-traitements. Pour la recherche effective d'égalité, un certain nombre de tests concernant des données du modèle donc ne nécessitant pas de traitements ou de calculs particuliers permettent de réduire le nombre de tentatives d'appariement infructueuses. De plus, la répartition des différents éléments en classes d'équivalence permet de guider la recherche d'égalité de graphes.

Pour ces différentes raisons, les potentialités de cette méthode méritent d'être approfondies, notamment par son implémentation. Un reproche peut éventuellement être fait à cette méthode ; en effet, elle est très liée à l'application, les différents tests et décompositions sont basés sur des caractéristiques du problème à résoudre. Nous constaterons ultérieurement que cette caractéristique peut être un gros avantage de la méthode.

La deuxième méthode proposée, ne possède pas cet inconvénient. En effet, l'approche par le formalisme CSP et notamment sa représentation sous forme de graphes, permet de représenter clairement et simplement, un grand nombre de problèmes de satisfaction de contraintes et en particulier celui de recherche d'égalité de graphes.

3.2. Méthode utilisant le formalisme CSP

Dans cette partie, nous nous intéressons toujours à la comparaison d'objets, modélisés sous forme de graphes, dans les systèmes de CAO. Cette comparaison, réalisée sur les graphes normalisés des pièces, consiste en une recherche d'égalité et d'inclusion. L'objectif est d'apparier tous les nœuds des deux graphes pour l'égalité alors que pour l'inclusion, tous les nœuds d'un des deux graphes doivent être appariés à un nœud de l'autre graphe.

Précédemment (cf. également [LEI 95], [LEI 97]) une première méthode de comparaison a été réalisée en utilisant les techniques classiques de recherche d'isomorphisme de graphes. Dans cette partie, nous avons préféré présenter la seconde approche étudiée qui utilise une formalisation par les CSP du même problème. L'avantage de cette seconde approche par rapport à la première est de pouvoir être clairement formalisée (graphe) et de disposer d'algorithmes robustes de résolution.

Un problème peut être modélisé sous forme de CSP s'il est constitué d'un ensemble de variables, chacune associée à un domaine fini et discret de valeurs, et d'un ensemble de contraintes. Les contraintes mettent en relation ces variables et permettent de définir une

instanciation correcte des différentes variables. Une solution est une instanciation des variables qui satisfait toutes les contraintes. Nous verrons que la recherche d'égalité ou d'inclusion de deux graphes est typiquement un problème pouvant être modélisé par un CSP.

La modélisation sous forme de CSP peut être évitée si pendant une phase de pré-traitements (peu coûteuse en parcours, en tests et en calcul) constituée de différents tests, l'incompatibilité entre les deux graphes comparés a pu être mise en évidence.

3.2.1. Modélisation sous forme de CSP

Le problème d'égalité de deux graphes G_1 et G_2 (figure 2.18) peut être représenté assez naturellement sous forme de CSP. En effet, les nœuds d'un des graphes (ici G_1) correspondent aux nœuds du CSP. Pour chacun des nœuds du CSP, un domaine de valeurs possibles est défini ; cet ensemble (contenu dans un cercle) correspond initialement aux nœuds du second graphe (G_2). Il existe une arête entre deux nœuds du CSP, s'il existe une contrainte binaire (⊥, //, ...) entre ces deux nœuds dans le graphe G_1. Sur chaque arête du CSP sont conservées en extension (couple de valeurs) les contraintes identiques de G_2. Par exemple, comme entre les nœuds 1 et 3 de G_1 il existe une contrainte de perpendicularité, dans le CSP, sur l'arête entre 1 et 3, on trouvera en extension, toutes les contraintes de perpendicularité de G_2 c'est-à-dire (b,c) et (b,d).

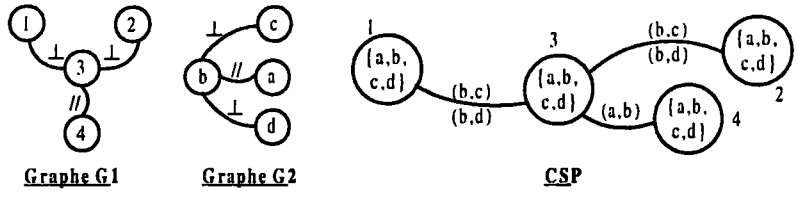


Figure 2.18. *Modélisation par un CSP de la comparaison de deux graphes*

Si, lors de la résolution du CSP, une valeur différente est trouvée pour chaque variable du graphe, l'égalité des deux graphes est montrée. Par contre, si seulement une partie des variables a pu être instanciée, le sous-graphe constitué par ces nœuds de G_1 est inclus dans le graphe G_2.

La modélisation sous forme de CSP des problèmes d'égalité ou d'inclusion de graphes permet d'utiliser pour sa résolution des algorithmes connus. Certains algorithmes de base ont été présentés dans la partie bibliographique et des études comparatives de ces différents algorithmes ont montré que l'algorithme de recherche en avant avait un bon rapport coût/efficacité.

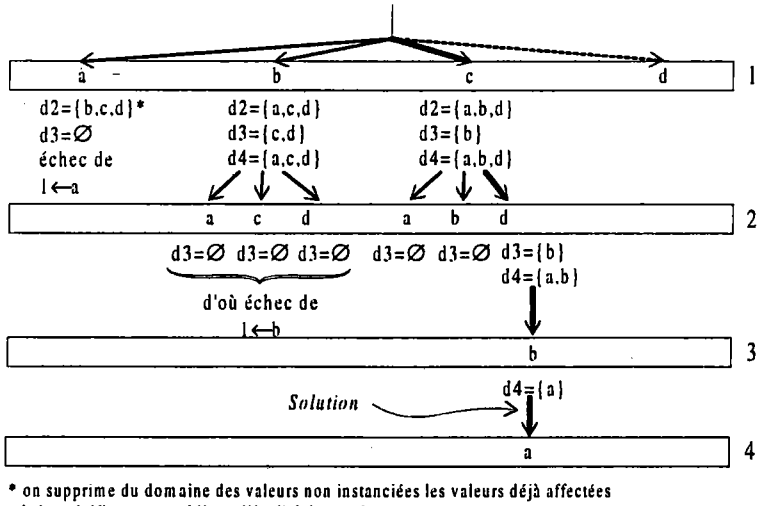
3.2.2. L'algorithme de recherche de consistance en avant (Forward-Checking)

L'algorithme de Forward-checking (FC) fonctionne avec un filtrage pendant la recherche, il consiste en une analyse du voisinage immédiat de la variable dernièrement instanciée. Le principe de cet algorithme présenté dans le chapitre 1 au §2.4.2 est explicité par la procédure récursive FC qui prend en argument une séquence de variables V à instancier et une instanciation $\mathcal{A}$. Elle utilise également une procédure auxiliaire check-forward.

```
FC(V,A)
 si V=∅ alors A est une solution
 sinon
 soit x_i ∈ V
   pour tout v ∈ d_i faire
      empiler(V-{x_i})
      si check-forward(x_i,v,V) alors
         FC(V-{x_i}, A∪{x_i→v})
      fin si
      dépiler(V-{x_i})
   fin pour
fin si
```

```
check-forward(x_i,v,V)
 soit consistant = true
 pour tout x_j ∈ V-{x_i} tant que consistant faire
    pour tout v' ∈ d_j faire
       si {x_i→v, x_j→v'} est non consistant alors
          d_j ← d_j-{v'}
       fin si
    fin pour
    si d_j = ∅ alors  consistant ← false  fin si
 fin pour
 retourner consistant
```

Figure 2.19. *Les procédures FC et check-forward*



* on supprime du domaine des valeurs non instanciées les valeurs déjà affectées (règle spécifique au problème d'égalité de graphes)

Figure 2.20. *Algorithme du forward-checking*

La figure 2.20 montre l'application de l'algorithme de FC à l'exemple de la figure 2.18. Les flèches noires correspondent à une instanciation testée (par exemple 1a), les flèches en gras représentent la solution trouvée (1c), et les flèches en pointillés signifient que l'instanciation n'a pas été testée.

Nous avons présenté dans cette partie une méthode permettant de comparer les graphes normalisés restreints des pièces. Cette représentation en utilisant le formalisme CSP, de notre problème, permet d'utiliser des algorithmes plus efficaces que le traditionnel retour arrière.

3.3. Comparaison des deux méthodes

Dans cette partie, les résultats des deux méthodes présentées précédemment sont comparés. Nous nous intéressons plus particulièrement aux temps d'exécution des deux méthodes implantées de manière itérative. Ces temps d'exécution sont examinés en fonction du nombre de nœuds des graphes considérés c'est-à-dire en fonction du nombre de variables du CSP. Les problèmes considérés sont des problèmes assez compliqués ; en effet, lors de la vérification de l'inclusion, de la non inclusion ou encore de l'inégalité de deux graphes, les deux graphes utilisés sont très peu différents ce qui signifie :

- pour la méthode algorithmique : le nombre de nœuds ainsi que le nombre d'arcs des deux graphes sont égaux à plus ou moins une ou deux valeurs ;
- pour la méthode utilisant le formalisme CSP : le nombre de variables du problème est presque identique au nombre de valeurs du domaine de chaque variable et le nombre de contraintes est proche du nombre de n_uplets potentiels.

La prise en compte de ce type de problèmes est certes plus délicate et donne de moins bons résultats que la comparaison de graphes très différents. Cependant, c'est ce type de situations qu'il est intéressant de détecter pour les logiciels de CAO puisque c'est précisément lorsque les graphes représentant les différentes pièces sont très peu différents que le système peut assister l'utilisateur en lui signifiant l'égalité ou l'inclusion de deux pièces en vue, par exemple, de la réutilisation d'un élément d'une bibliothèque.

Nous nous intéressons dans cette partie, aux temps d'exécution obtenus par les deux méthodes lors de la détection d'égalité ou d'inclusion de graphes, cet ensemble de situations correspond à un ensemble de problèmes satisfiables. L'ensemble des problèmes non satisfiables constitués des cas de non égalité ou de non inclusion de graphes est également étudié ici.

3.3.1. Problèmes satisfiables

Dans ce paragraphe, nous comparons les temps d'exécution des deux méthodes lors de la détection de l'égalité ou de l'inclusion de graphes en fonction du nombre de variables du problème c'est-à-dire du nombre de nœuds des graphes.

Les figures 2.21 et 2.22 montrent que lorsque le nombre de variables du problème devient grand, l'algorithme de résolution de CSP, Forward-Checking (FC) est beaucoup moins performant que l'algorithme spécifique au problème et correspondant à la première méthode présentée (Algo).

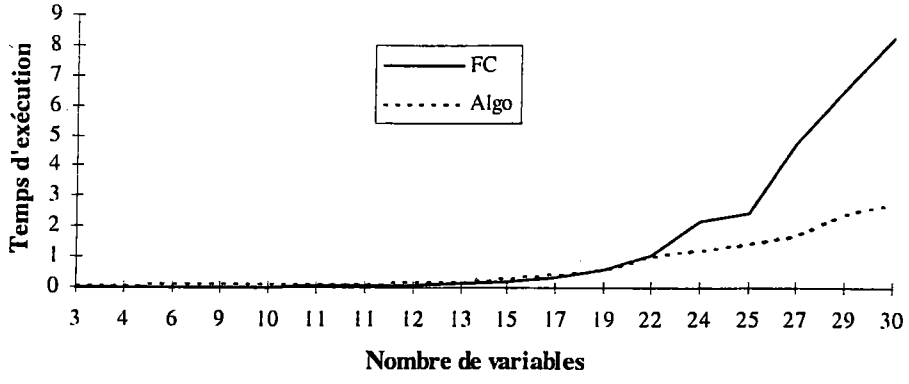


Figure 2.21. *Égalité de graphes*

Par contre, lorsque le nombre de variables du problème est assez faible, le Forward-Checking donne le résultat de la comparaison un peu plus rapidement que l'algorithme de backtracking spécifique (figures 2.21 et 2.23).

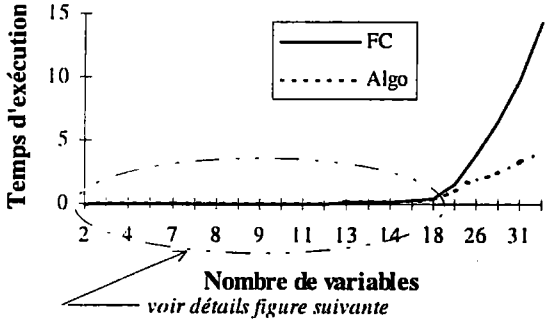


Figure 2.22. *Inclusion de graphes*

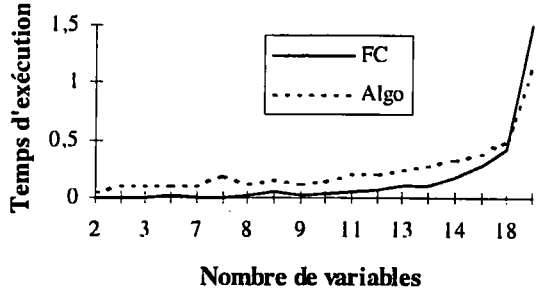


Figure 2.23. *Vérification de l'inclusion avec un nombre de variables faible*

Pour détecter l'égalité ou l'inclusion de deux graphes, les deux méthodes proposées sont toutes les deux très efficaces lorsque le nombre de variables n'est pas trop important. Expérimentalement, la méthode utilisant le formalisme CSP est plus efficace, pour ce type de problème que la méthode algorithmique. Cette tendance s'inverse lorsque le problème est constitué de plus d'une vingtaine de variables, FC devient alors rapidement beaucoup plus coûteux que l'algorithme spécifique. Les deux méthodes ont une croissance exponentielle cependant la croissance du temps d'exécution de la méthode algorithmique est bien moins rapide que celle de l'algorithme de Forward-checking.

3.3.2. Problèmes non satisfiables

Dans ce paragraphe, nous comparons les temps d'exécution des deux méthodes lors de la détection de l'inégalité ou de la non inclusion de deux graphes. Nous pouvons constater grâce aux figures 2.24 et 2.25, que la tendance est la même que pour la détection de l'égalité ou l'inclusion de graphes. Cependant, le point de croisement des deux courbes se situe beaucoup plus tôt sur l'axe représentant le nombre de variables. Ce fait signifie que l'algorithme de Forward-checking devient moins intéressant que l'algorithme spécifique plus rapidement que pour les problèmes satisfiables.

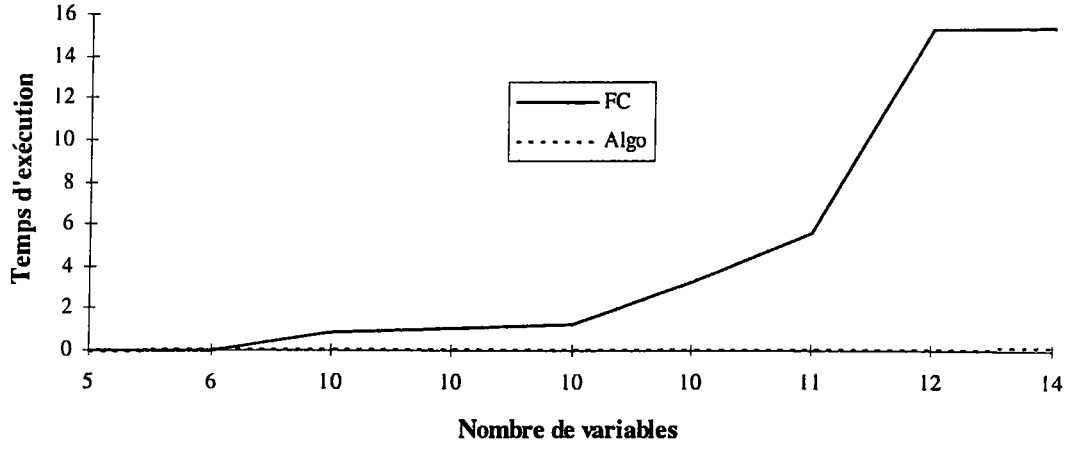


Figure 2.24. *Non inclusion de graphes*

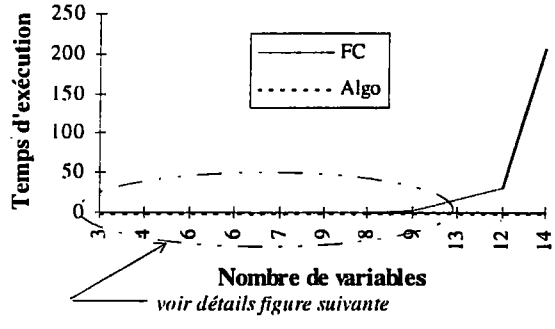


Figure 2.25. *Non égalité de graphes*

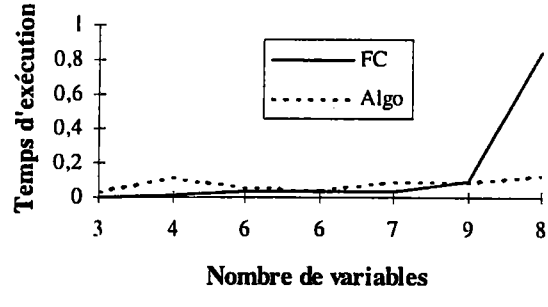


Figure 2.26. *Non égalité avec un nombre faible de variables*

Les deux figures 2.24 et 2.25 permettent de remarquer que le temps d'exécution de la méthode algorithmique est presque constant ; en fait il augmente très faiblement et régulièrement en fonction du nombre de variables du problème. Cette dernière remarque, ainsi que le fait que la méthode utilisant le formalisme CSP est très peu efficace pour ce type de problème, peuvent s'expliquer par les spécificités de la méthode algorithmique.

La méthode algorithmique étant dédiée aux problèmes de recherche d'égalité ou d'inclusion de graphes, nous avons vu qu'elle dispose d'un certain nombre de pré-traitements permettant de rejeter rapidement des problèmes non satisfiables. Par exemple, pour la vérification de l'égalité de deux graphes, un premier filtre consiste à comparer le nombre de nœuds de chacun des graphes, si ces deux nombres sont différents l'inégalité est ainsi rapidement détectée. Au contraire, l'algorithme de Forward-checking ne tenant pas compte de cette particularité du problème, va tenter d'instancier toutes les variables du problème avec une valeur différente du domaine de définition pour constater après l'échec de toutes les combinaisons possibles que le problème est insatisfiable.

Toutefois, pour les problèmes de moins d'une dizaine de variables, FC est plus efficace que l'algorithme spécifique comme on peut le constater sur le figure 2.26. Le temps d'exécution de l'algorithme spécifique est presque constant puisqu'il est réduit au temps nécessaire à la phase de pré-traitement qui consiste à l'évaluation et à la comparaison de certaines constantes comme le nombre d'éléments, le nombre de contraintes, de classes d'équivalence, ...

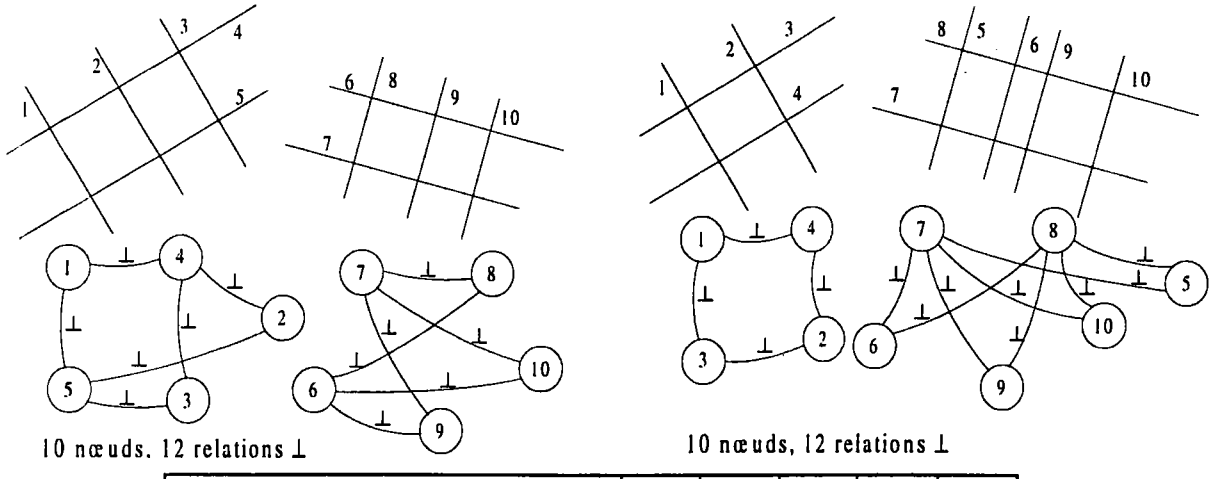
3.3.3. Synthèse

Les deux parties précédentes ont mis en évidence le fait qu'il n'existe pas une méthode très efficace quel que soit le problème traité. En effet, l'efficacité des deux méthodes proposées est fonction non seulement du nombre de variables du problème mais également du résultat de la comparaison. Le choix de la méthode devient donc très compliqué car si le nombre de variables du problème est connu a priori et peut ainsi orienter le choix de la méthode, le résultat escompté qui constitue en fait le facteur déterminant pour définir la meilleure méthode ne peut évidemment pas être connu a priori.

Pour les problèmes étudiés, lorsque le nombre de variables est proche de dix, l'algorithme de Forward-checking est, quel que soit le résultat, le plus efficace. Par contre, lorsque le nombre de variables est supérieur, il devient risqué d'utiliser cet algorithme car en cas de problème non satisfiable il peut devenir réellement très mauvais. On peut citer, à titre indicatif, la détection de la non inclusion de deux graphes constitués de quatorze et quinze éléments, qui a pris 4107,23 secondes à l'algorithme de Forward-checking contre 0,21 secondes à la méthode algorithmique. Cependant, la dégradation des performances de l'algorithme de FC pourrait être retardée par l'ajout de certains pré-traitements semblables à ceux utilisés par la méthode algorithmique.

On peut se demander, si finalement cette comparaison entre les deux méthodes proposées est très équitable. Nous pensons qu'elle l'est et ceci pour plusieurs raisons. Nous rappelons que les pré-traitements réalisés par la première méthode consistent à comparer le nombre de nœuds ou encore le nombre de contraintes de perpendicularité ou de parallélisme des graphes considérés. Ainsi en cas d'égalité ou d'inclusion des graphes comparés, la comparaison des méthodes est équitable puisque, au contraire, les filtres sont évalués et n'apportent rien.

Par contre, si les graphes considérés ne sont pas égaux, l'évaluation des différentes méthodes peut être faussée. En effet, si l'inégalité des deux graphes peut être montrée par une différence de leur nombre de noeuds ou de leur nombre de contraintes d'un certain type, alors la méthode utilisant les filtres est grandement avantagée, ce qui n'est pas le cas si la différence consiste en une association différente des contraintes. La figure 2.27 présente des exemples où les filtres ne permettent pas de conclure sur l'égalité ou l'inégalité des graphes comparés.



Temps de recherche avec notre méthode	0,12	0,11	0,12	0,12	0,11
Temps de recherche avec CSP	3,78	3,77	3,78	3,77	3,78

Figure 2.27. *Comparaison CSP/méthode spécifique*

4. IMPLÉMENTATION

Le travail d'implémentation a été très différent pour les deux méthodes ; la première méthode a été intégralement développée pour sa partie gestion des contraintes dans le cadre de ce travail et intégré dans le logiciel développé au LRIM appelé SACADO (Système Adaptatif de Conception et d'Aide au Développement par Ordinateur). La seconde méthode a bénéficié de travaux réalisés par un membre du LRIM ; en effet, des travaux ont été menés sur la comparaison de différentes

méthodes de résolution pour les CSP. De nombreux algorithmes ont été développés dans le cadre de cette étude, de façon à comparer leur temps d'exécution et leur complexité. Notre travail a consisté à écrire une interface permettant de traduire nos représentations des pièces à comparer sous forme de CSP en respectant un format prédéfini ; ces informations sont ensuite transmises à l'algorithme de résolution de CSP choisi.

Dans cette partie, nous allons donc présenter dans un premier temps, les différents concepts du système SACADO. Nous détaillerons ensuite l'implémentation de la méthode itérative de comparaison de modèles en ajoutant certaines précisions par rapport à la présentation générale faite en début de chapitre. Finalement, nous exposerons la méthode permettant de transformer la modélisation des objets contraints issue de cette maquette en un format compréhensible par les algorithmes de résolution de CSP existants déjà.

4.1. Présentation générale

La maquette réalisée a pour objectif d'illustrer les principes énoncés précédemment. Elle permet notamment de tester le système de règles proposé permettant d'obtenir une représentation unique d'une situation. Les algorithmes permettant de vérifier l'inclusion et l'égalité des graphes représentatifs de deux pièces ont également été implantés ; la correction de ces algorithmes est éventuellement vérifiable dans le cas de pièces encore assez simples grâce à la visualisation simultanée des deux pièces avec toutes les contraintes liant leurs éléments de base. L'efficacité de la méthode de parcours du graphe et de la décomposition en classes d'équivalence est mise en évidence par la maquette, par des temps d'exécution et de calculs tout à fait raisonnables.

Les nouvelles fonctionnalités proposées par notre étude ont été incorporées à la maquette SACADO existante et présentées dans [PIP95]. Nous avons pu ainsi réutiliser toute la partie interface homme-machine permettant une construction conviviale de pièces contraintes et l'intégration de nouveaux menus respectant la philosophie de dialogue homme-machine propre à SACADO (menus locaux, immédiats, différés) a pu se faire de façon assez simple. Le modeleur de SACADO conservant toutes les informations sur les objets de base (coordonnées des points, rayon des cercles, ...) est lié au graphe de la pièce modélisant les relations entre les objets de base (figure 2.28).

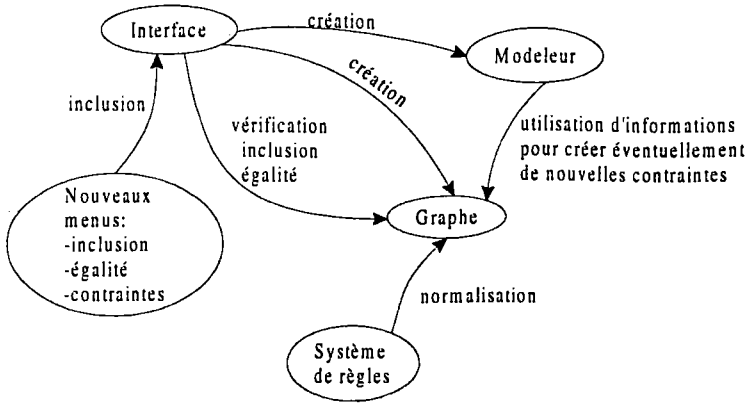


Figure 2.28. *Flux des données dans la maquette réalisée.*

4.2. Distinction contraintes implicites / contraintes explicites

Le problème qui apparaît lors de toute conception de pièces dans un système de CAO est dû à la complète liberté laissée au concepteur. En effet, si cette caractéristique permet une meilleure utilisation du système en ne restreignant pas le concepteur dans ses choix, la modélisation devient par contre bien plus délicate. Une même pièce pourra être construite de manière totalement différente par plusieurs personnes, voire par la même personne pour peu que cette conception soit faite à des moments distincts. Et ce problème devient plus aigu dès que deux systèmes sont amenés à communiquer.

Afin de pouvoir découvrir les similitudes de pièces construites de différentes manières, deux types de contraintes sont distingués, les contraintes explicites et les contraintes implicites ; cette distinction nous amène à parler d'égalité et d'inclusion forte et faible, d'arêtes explicites et d'arêtes implicites et de classes d'équivalence distinguant les contraintes explicites et les contraintes implicites.

Nous avons vu précédemment qu'une arête représente une contrainte explicite lorsque cette dernière est fixée par l'utilisateur à la conception à l'aide, par exemple, d'un menu. Une arête représente une contrainte implicite lorsqu'elle est détectée par le système. Par exemple, l'utilisateur en traçant à main levée un segment peut le dessiner perpendiculaire à un segment existant avec une petite imprécision autorisée, la contrainte de perpendicularité est alors conservée par le système en la distinguant toutefois d'une contrainte perpendiculaire imposée explicitement par l'utilisateur. Par la suite, en cas de modification donc de propagation de contraintes, les contraintes explicites seront considérées comme des contraintes fortes qui

doivent absolument être vérifiées contrairement aux contraintes implicites qui peuvent être considérées comme fortuites et qu'il n'est donc pas nécessaire de maintenir absolument.

Deux types d'égalité pourront ainsi être définis, l'égalité forte et l'égalité faible. Deux scènes sont fortement égales si elles ont le même nombre de segments et de cercles. De plus, les contraintes du graphe normalisé des deux pièces sont les mêmes ; les éléments des deux pièces n'ont peut-être pas été construits dans le même ordre et liés par les mêmes contraintes mais le résultat de la normalisation est le même. L'égalité forte est donc réalisée s'il existe exactement les mêmes contraintes explicites et les mêmes contraintes implicites dans les deux graphes normalisés ; même nombre de contraintes de même type (perpendiculaire, parallèle, ...) et de même nature (explicite, implicite).

L'égalité faible est réalisée entre deux graphes normalisés s'ils ont le même nombre de contraintes de même type ; par contre ces différentes contraintes peuvent être de natures différentes. On vérifie donc, pour l'égalité faible, uniquement l'égalité du nombre de contraintes de même type sans tenir compte de la nature de ces différentes contraintes.

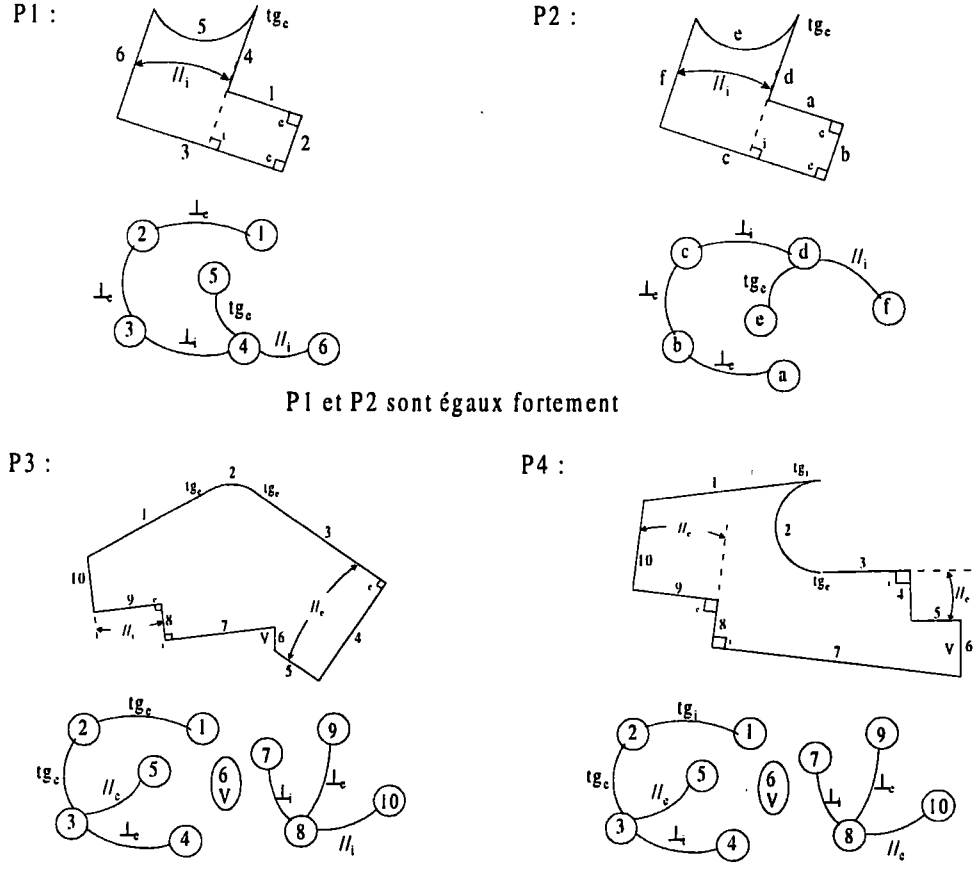


Figure 2.29. *Exemples de graphes égaux fortement et faiblement.*

Les deux types d'égalité définis, comparent les graphes d'un point de vue topologique. Il serait également intéressant de combiner à ces comparaisons de topologie, des comparaisons concernant la géométrie. On pourrait, par exemple, rechercher une composition de transformations affines permettant de transformer une des scènes en l'autre et définir ainsi un nouveau type d'égalité vérifié lorsqu'il existe une telle relation géométrique associée à une relation d'égalité forte (figure 2.30).

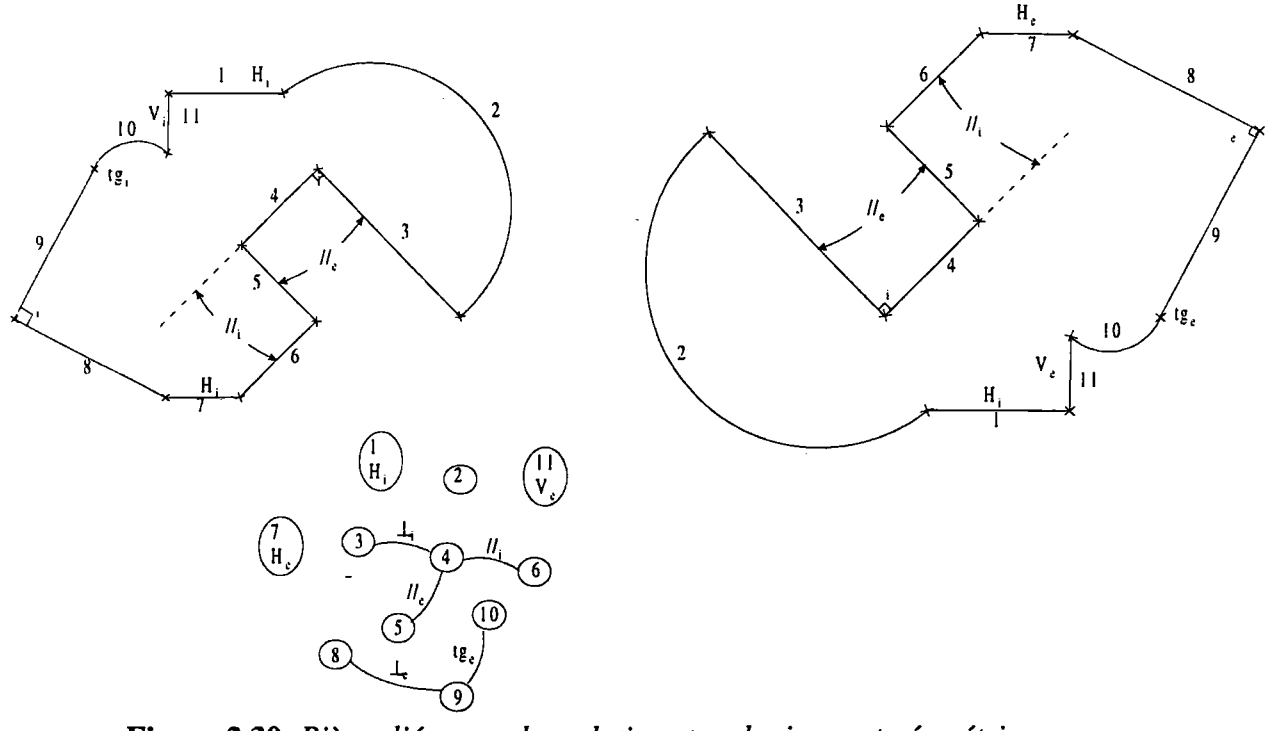


Figure 2.30. *Pièces liées par des relations topologiques et géométriques*

Comme on a distingué deux types d'égalité, on distingue également deux types d'inclusion. On peut rechercher l'inclusion forte du graphe normalisé de la scène en construction dans le graphe normalisé de la scène extraite d'une bibliothèque. Il y a inclusion si toutes les contraintes explicites et implicites du graphe normalisé courant sont incluses dans le graphe normalisé chargé. On doit associer chaque contrainte implicite et explicite du graphe chargé à une contrainte de même type et de même nature du graphe normalisé courant.

L'inclusion faible est réalisée dans les mêmes conditions que l'inclusion forte mais on a la possibilité d'associer chaque contrainte explicite ou implicite du graphe normalisé courant avec soit une contrainte topologique soit indifféremment, une contrainte explicite ou implicite de même type du graphe normalisé.

Pour vérifier l'égalité ou l'inclusion de deux graphes, il a été montré précédemment qu'il est nécessaire d'associer chaque nœud du graphe normalisé courant avec un nœud du graphe normalisé chargé. Pour optimiser la recherche de correspondance correcte entre deux nœuds, les nœuds sont regroupés en classes. Les classes d'équivalence sont définies par le type des éléments et le nombre d'arêtes. Par exemple, la classe des segments possédant deux arêtes regroupe les numéros de nœud ayant cette propriété.

La distinction explicite/implicite fait apparaître deux représentations en classes d'équivalence :
- une classe ne considérant que les arêtes explicites ;
- une classe considérant les arêtes implicites.

Par exemple, un segment avec trois arêtes explicites et deux arêtes implicites appartiendra :
- dans la représentation explicite à la classe des segments ayant trois arêtes ;
- dans la représentation implicite à la classe des segments ayant cinq arêtes.

Dans ce paragraphe, une distinction a été faite en fonction de la manière utilisée pour définir une contrainte. Le fait de distinguer des contraintes implicites et des contraintes explicites pourra conduire à des résultats différents en cas de propagation de contraintes. Sur la figure 2.31 la modification du point P conduit à une réévaluation des différents composants de la pièce or on peut constater que le système étant sur-contraint, toutes les contraintes ne peuvent être maintenues. Dans les deux cas, une contrainte implicite donc de priorité faible est relaxée et deux pièces différentes sont ainsi obtenues.

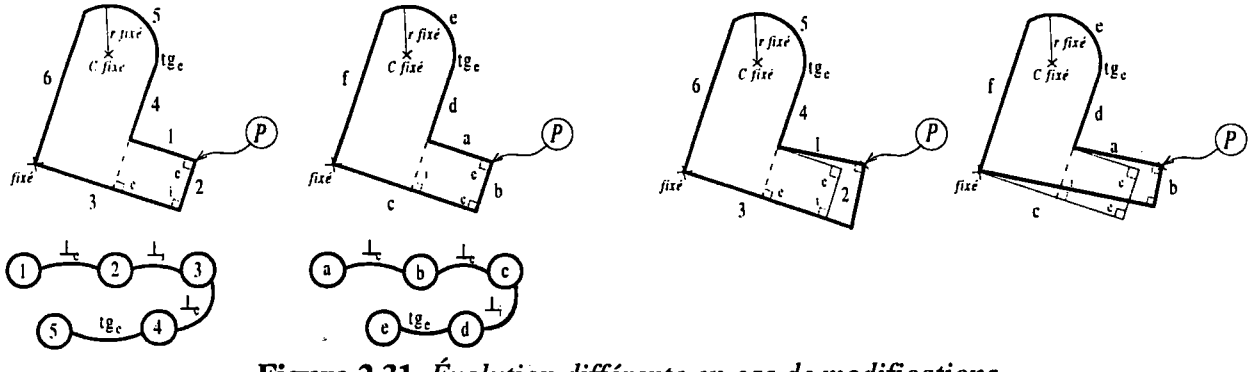


Figure 2.31. *Évolution différente en cas de modifications*

4.3. Égalité de graphes par une méthode algorithmique

Dans cette partie, nous présentons les différents algorithmes mis en œuvre par la méthode de comparaison de graphes algorithmiques présentée au paragraphe 3.1.

Cette méthode consiste tout d'abord à comparer le nombre d'éléments de chaque graphe, s'ils n'ont pas le même nombre de segments ou le même nombre de cercles alors ils sont différents et l'algorithme se termine. S'ils ont le même nombre d'éléments, les différentes composantes connexes sont extraites des deux graphes ; si les deux graphes n'ont pas le même nombre de composantes connexes, à nouveau on peut conclure qu'ils sont différents.

Pour chaque composante, le système calcule ensuite les classes d'équivalence pour les segments et pour les cercles. Le critère permettant de définir les différentes classes d'équivalence est dans notre cas, le nombre d'arêtes de chaque nœud ; il existe donc une classe pour les segments ayant une arête, trois arêtes ou encore celle des cercles ayant deux arêtes, ...

Le test suivant, permettant de déceler l'inégalité des deux graphes en comparaison, porte donc sur les classes d'équivalence, si leur nombre ou leur nature sont différents alors les graphes sont différents. Par exemple, soit g_1 et g_2 deux graphes ayant chacun trois classes :

- g_1 : segment 2 arêtes, segment 1 arête et cercle 2 arêtes ;
- g_2 : segment 3 arêtes, segment 2 arêtes et cercle 2 arêtes.

Les graphes g_1 et g_2 sont forcément différents car les classes sont de nature différente. Si les deux graphes ont le même nombre de classes d'équivalence et qu'elles sont de même nature, l'inégalité des deux graphes n'a pas été montrée mais cela n'implique aucunement leur égalité.

À cette étape, le système associe les deux classes d'équivalence de même nature, elles doivent avoir le même nombre de représentants. Si les deux classes ont le même nombre de représentants alors, l'égalité étant toujours possible, le système peut associer un nœud de chaque graphe appartenant à la classe d'équivalence traitée. Deux nœuds peuvent être associés s'ils ont le même nombre d'arêtes et si les arêtes sont de même type (algorithme d'association de nœuds figure 2.32). Un nœud n_1 de g_1 représentant un segment ayant deux arêtes perpendiculaires doit être associé à un nœud segment de g_2 appartenant à la classe des segments avec deux arêtes. Tous les nœuds de cette classe ne sont toutefois pas des candidats corrects à l'association avec n_1 car ils doivent avoir deux arêtes perpendiculaires. En effet, le candidat n_2 ayant deux arêtes perpendiculaires est correct ; par contre, le nœud n_3 possédant une arête perpendiculaire et une arête parallèle ne convient pas.

Association:
État initial : n_1 et n_2 sont associés.
État final : n_{11} un suivant de n_1 et n_{22} un suivant de n_2 sont associés.
On choisit n_{11} un des suivants de n_1 (il existe une arête liant n_1 et n_{11}).
On recherche tous les suivants de n_2 et on les stocke dans une liste $\mathcal{L}$.
On supprime de $\mathcal{L}$ tous les nœuds qui n'ont pas le même nombre d'arêtes que n_{11}.
On prend le premier nœud de la liste $\mathcal{L}$ des candidats possibles restants et on vérifie si ses arêtes sont de même type que celles de n_{11}. Si c'est le cas, on associe les deux nœuds sinon on essaye un autre nœud de la liste.

Figure 2.32. *Association de deux nœuds*

On procédera ensuite de la même façon avec tous les suivants du nœud jusqu'à ce que tous les nœuds soient associés : l'égalité des deux composantes est alors prouvée et la même technique est appliquée aux autres composantes. Si on arrive à un blocage lors de l'appariement des nœuds, et que toutes les combinaisons n'ont pas été réalisées alors il y a retour arrière avec destruction de certaines associations, et d'autres couplages sont testés. Par contre, si tous les couples de nœuds essayés des deux composantes ont conduit à l'échec alors d'autres associations de composantes sont vérifiées. Si en définitive, une composante du premier graphe ne peut être associée avec aucune composante du second alors les deux graphes sont différents.

ÉgalitéGraphe (G_1,G_2) : booléen
{Algorithme recherchant l'égalité entre deux graphes G_1 et G_2, le résultat de cette comparaison est un booléen mis à vrai si les graphes sont égaux et faux sinon.}
début
Calcul de l'arbre des composantes de G_1
Calcul de l'arbre des composantes de G_2
Tantque il existe une composante non associée de G_1 et que l'inégalité n'est pas montrée faire
Choisir une composante c_1 non associée de G_1
Tantque c_1 n'est pas associée et qu'il existe une composante non associée de G_2 non testée faire
Choisir une composante c_2 non associée de G_2
Vérifier l'association des composantes c_1 et c_2
fin tantque
Si c_1 n'a pas pu être associée alors
les deux graphes ne sont pas égaux
fin si
fin tantque
Si toutes les composantes ont pu être associées alors
les deux graphes sont égaux
fin si
fin

Figure 2.33. *Égalité de graphes*

La figure 2.33 représente l'algorithme d'égalité de deux graphes G_1 et G_2. Pour se faire, il utilise, dans un premier temps, un algorithme lui permettant d'extraire les différentes

composantes des graphes (figure 2.36). Dans un second temps, il recherche l'égalité des différentes composantes en testant leur association deux à deux (figure 2.35).

```
AssocierComposante (c1,c2) : booléen
    {L'algorithme vérifie si les deux composantes c1 et c2 peuvent être associées, si c'est possible le résultat est
    un booléen égal à vrai sinon le booléen est mis à faux.}
début
    Calcul des classes d'équivalence de c1
    Calcul des classes d'équivalence de c2
    Si      nombre de classes de c1 ≠ nombre de classes de c2 ou
            nature des classes de c1 ≠ nature des classes de c2          alors
            les deux composantes ne peuvent pas être associées
    sinon
            Tantque il existe une classe non associée de c1 et
                    que l'incompatibilité des deux composantes n'a pas été montrée          faire
                    Choisir une classe cl1 non associée de G1
                    Choisir la classe cl2 de G2 correspondant à cl1
                    Vérifier l'association des deux classes cl1 et cl2
                    Si les deux classes ne peuvent pas être associées alors
                            l'association des deux composantes est impossible
                    fin si
            fin tantque
    fin si
fin
```

Figure 2.34. *Égalité de composantes*

L'algorithme vérifiant l'association possible de deux composantes utilise un algorithme permettant de vérifier l'association possible de deux classes ; cet algorithme est celui de la figure 2.35.

```
AssocierClasse (cl1,cl2) : booléen
    {Algorithme vérifiant la validité de l'association des deux classes cl1 et cl2, si elles peuvent effectivement
    être associées, le booléen résultat sera mis à vrai sinon il sera mis à la valeur faux.}
début
    Tantque     il existe un nœud de cl1 non associé et
                que l'incompatibilité des deux classes n'a pas été montrée faire
            Choisir un nœud n1 de cl1 non associé
            Tantque n1 n'est pas associé et qu'il existe des nœuds non associés n2 de cl2 non testés faire
                    Choisir un nœud n2 non associé de cl2
                    Vérifier l'association des deux nœuds n1 et n2
            fin tantque
            Si n1 n'a pas pu être associé alors
                    les deux classes ne peuvent pas être associées
            fin si
    fin tantque
fin
```

Figure 2.35. *Égalité de classes*

ExtractionComposante

{Algorithme réalisant l'extraction des différentes composantes connexes dans un graphe}

début

Conserver dans une liste L les numéros de tous les nœuds du graphe

Tantque L n'est pas vide faire

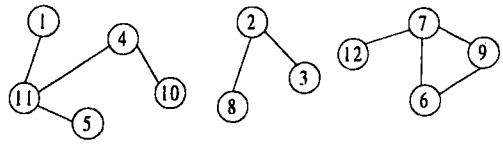
Premier élément de la liste n

Recherche de la composante à partir de n : retirer n de L ainsi que tous ses suivants et suivants des suivants, ... qui constituent une composante (à chaque fois qu'un nœud est ajouté à la composante, il est retiré de L)

Quand tous les suivants ont été parcourus, si la liste L n'est pas vide, la définition d'une nouvelle composante commence à partir du premier élément de L.

fin tantque

fin



L = 1 2 3 4 5 6 7 8 9 10 11 12

Départ première composante : 1

suivant de 1 : 1 11	L = 2 3 4 5 6 7 8 9 10 12
suivant de 11 : 1 11 4	L = 2 3 5 6 7 8 9 10 12
suivant de 4 : 1 11 4 10	L = 2 3 5 6 7 8 9 12
10 n'a pas de suivant, 4 n'a plus de suivant, suivant de 11 : 1 11 4 10 5	L = 2 3 6 7 8 9 12

5 n'a pas de suivant, 11 n'a plus de suivant, 1 n'a plus de suivant.

La première composante est : 1 11 4 10 5

Départ deuxième composante : 2

suivant de 2 : 2 3	L = 6 7 8 9 12
3 n'a pas de suivant, suivant de 2 : 2 3 8	L = 6 7 9 12

2 n'a plus de suivant.

La deuxième composante est : 2 3 8

Départ troisième composante : 6

suivant de 6 : 6 7	L = 9 12
suivant de 7 : 6 7 12	L = 9
12 n'a pas de suivant, suivant de 7 : 6 7 12 9	L =

9 n'a plus de suivant, 7 n'a plus de suivant, 6 n'a plus de suivant.

La troisième composante est : 6 7 12 9

L est vide, toutes les composantes sont extraites.

Figure 2.36. *Extraction des différentes composantes connexes.*

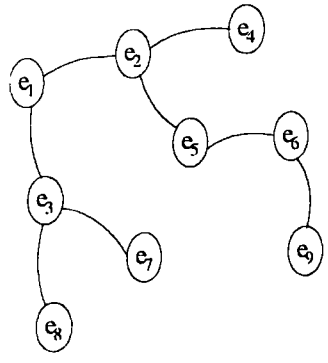
4.4. Parcours de graphe lors de l'application des règles

On pourrait appliquer systématiquement toutes les règles à tous les éléments jusqu'à obtenir la stabilité du modèle (toutes les relations restent inchangées entre les différents éléments). En suivant cette méthode des objets sont étudiés inutilement et de nombreuses règles sont appliquées inutilement. C'est pourquoi, une stratégie différente a été définie de façon à étudier uniquement les éléments susceptibles d'être modifiés et à appliquer uniquement les règles pouvant entraîner un éventuel changement.

Cette stratégie consiste à appliquer à partir du nouvel élément ajouté ou du dernier élément modifié, l'ensemble des règles correspondant au type de l'objet et traitant les contraintes liées à

l'objet. Puis tant qu'il y a une modification du graphe, les règles adéquates sont appliquées uniquement aux éléments liés à l'objet ayant entraîné la modification. On utilisera pour gérer cette méthode une pile où on empile les suivants d'un élément ayant entraîné une modification et on dépile tout élément une fois traité : le traitement s'achève lorsque la pile est vide.

Exemple :



élément testé (=sommet de la pile)	modification du graphe de contraintes		éléments liés à l'élément testé et non déjà traités	état de la pile
	oui	non		
état initial				e_1
e_1	×		e_2, e_3	e_2 e_3
e_3		×	(non nécessaire)	e_2
e_2	×		e_4, e_5	e_4 e_5
e_5	×		e_6	e_4 e_6
e_6		×	(non nécessaire)	e_4
e_4	×		aucun	∅

4.5. Extension de l'implémentation

4.5.1. Ajout et suppression du parallélisme

Il faut ensuite, définir une priorité plus faible pour les deux contraintes parallèle et perpendiculaire, en privilégiant plutôt l'une que l'autre. On va attribuer une priorité plus grande à la contrainte perpendiculaire qu'à la contrainte parallèle car c'est une contrainte moins ambiguë (problème posé par la symétrie).

En effet, on peut constater sur la figure 2.37 que lorsqu'une droite est perpendiculaire à deux droites parallèles, deux contraintes définissent complètement les relations entre ces trois droites. Toutefois, si on choisit comme relations les deux contraintes perpendiculaires, il n'y a pas d'ambiguïté alors que si on utilise une contrainte parallèle et une contrainte perpendiculaire, la représentation n'est pas unique et l'unicité de la représentation normalisée est remise en question. Il n'est pas nécessaire d'attribuer une priorité à la contrainte de tangence car elle n'est jamais en conflit avec aucune autre contrainte.

Exemple de règles gérant le parallélisme :

R_5 $(D_1 // D_2) \wedge (D_1 \perp D_2) \Rightarrow (D_3 \perp D_4) \wedge$ suppression $(D_1 // D_2)$

R_6 $(D_1 // D_2) \wedge (D_1 // D_3) \Rightarrow (D_2 // D_3)$

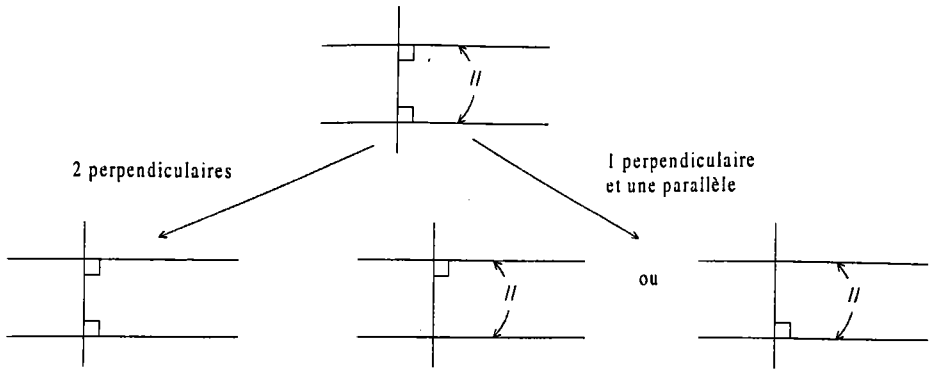


Figure 2.37. *Redondance d'informations*

Dans le cas très particulier de figure 2.37 où on crée D_1 puis $D_2//D_1$ puis $D_3//D_2$ et finalement $D_4 \perp D_1$, deux règles peuvent être appliquées :

R_6 $D_1//D_2$ et $D_2//D_3$ donc $D_1//D_3$

R_5 $D_4 \perp D_1$ et $D_1//D_3$ donc $D_4 \perp D_3$ et suppression $D_1//D_3$

Il y a bouclage du système (ajout $D_1//D_3$ suppression de $D_1//D_3$) si et seulement si en partant de D_1 on ne traite pas de D_2. En effet dans ce cas :

- R_5 est appliquée entre D_1,D_3 et D_4, on supprime $D_1//D_3$;
- R_5 est appliquée entre D_1,D_2 et D_4, on supprime $D_1//D_2$;
- R_7 ne peut plus rien ajouter.

D'où l'importance de l'ordre d'application des règles et de traitement des nœuds.

4.5.2. Contraintes supplémentaires traitées

Les pièces gérées par le système sont de deux types : droites et cercles (éventuellement segments et arcs). Nous avons vu que les différents éléments de base peuvent être mis en relation deux par deux par les contraintes de parallélisme et de perpendicularité. Dans le cadre de la maquette nous avons défini d'autres contraintes qui n'influencent pas le système de règles défini et qui sont :

- tangent ;
- expression : expressions grapho-numériques représentées par les menus ;
  - → même abscisse : $x_1 = x_2$ pour les deux points extrémités d'un segment ;
  - → même ordonnée : $y_1 = y_2$ pour les deux points extrémités d'un segment ;
  - → tangence donnée par une expression

$y_1 = y_2 = y_c+R$	(dessus)	$x_1 = x_2 = x_c+R$	(droite)
$y_1 = y_2 = y_c-R$	(dessous)	$x_1 = x_2 = x_c-R$	(gauche)

Ces contraintes sont appelées des contraintes externes par opposition aux contraintes internes ne concernant qu'un seul objet (horizontal, vertical).

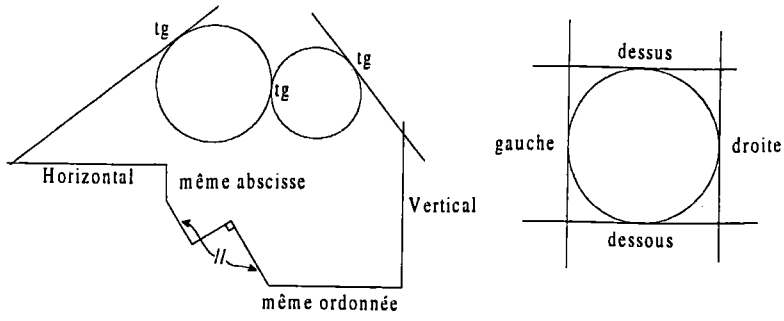


Figure 2.38. *Exemple de toutes les contraintes applicables sur les éléments de base*

Les différentes règles de remplacement sont :

- l'expression "même ordonnée" pour s_1 est remplacée par Horizontal pour s_1 ;
- l'expression "même abscisse" pour s_1 est remplacée par Vertical pour s_1 ;
- les expressions Dessus, Dessous, Gauche, Droite entre s_1 et c_1 sont remplacées par la contrainte de Tangence entre s_1 et c_1.

4.6. Interface maquette / algorithmes de résolution de CSP

Dans cette partie, nous présentons l'interface mise en œuvre pour réaliser la comparaison de deux modèles de pièces par un algorithme de résolution de CSP. Pour ce faire, il est nécessaire de traduire les modèles des pièces stockés sous forme de graphes dans le logiciel de CAO en une représentation du graphe CSP du problème. La représentation du graphe CSP choisie est un fichier texte ayant certaines caractéristiques que nous préciserons ultérieurement.

4.6.1. Présentation générale

La figure 2.39 présente de manière générale l'interface à réaliser pour transformer une structure de données du logiciel de CAO en des données utilisables par un algorithme de résolution de CSP. Au paragraphe 3.2 de ce chapitre nous avons montré comment traduire de manière formelle, la comparaison de deux graphes en un graphe représentant le problème sous forme de CSP. Dans cette partie, nous présentons cette même transformation mais en considérant les structures de données informatiques et des méthodes algorithmiques mises en œuvre.

La figure 2.39 montre que les structures de graphe représentant les modèles des pièces à comparer sont utilisées pour créer une nouvelle structure de données triant les informations utiles sous une forme facilement exploitable en vue de l'écriture d'un fichier texte contenant toutes les données du CSP. Ce fichier texte est ensuite traduit par un processus propre aux algorithmes de résolution de CSP en une structure de données spécifiques à l'application de tels algorithmes.

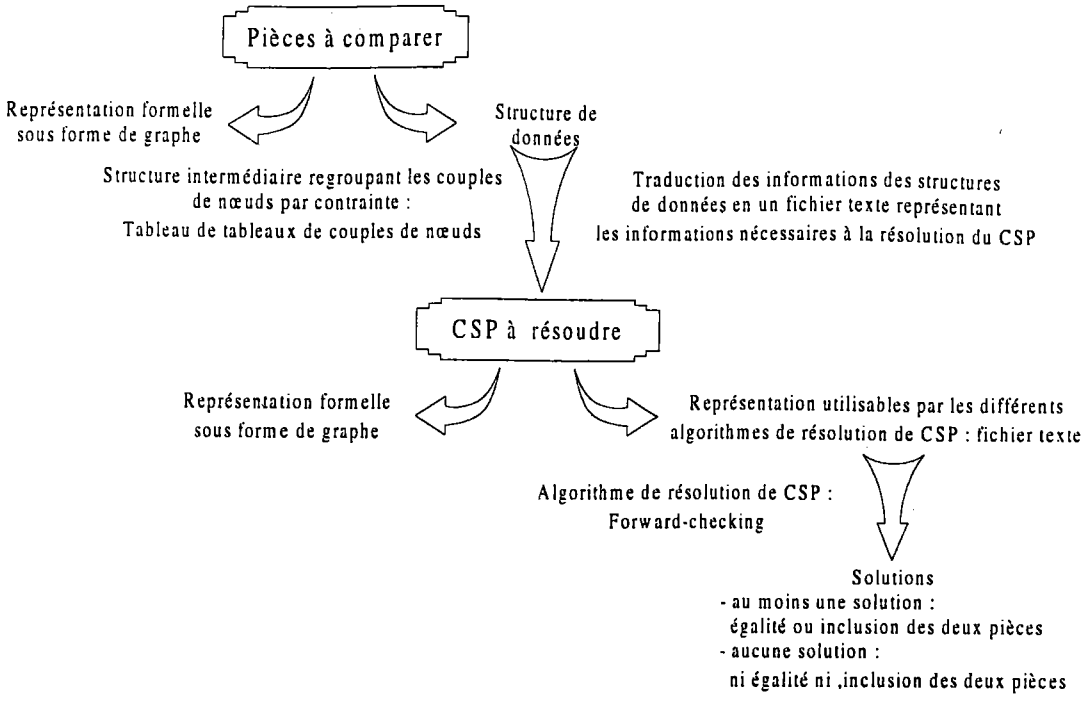
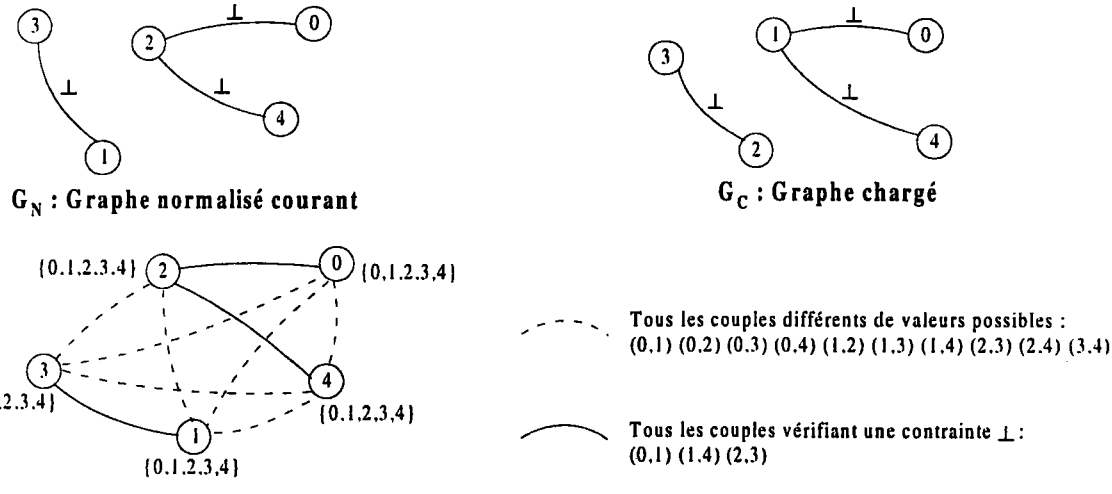


Figure 2.39. *Présentation générale de l'interface*

Finalement, si l'algorithme de résolution de CSP choisi qui est dans notre application l'algorithme de Forward-checking, retourne au moins une solution alors, suivant le problème modélisé, on peut conclure l'égalité ou l'inclusion des deux graphes considérés. Au contraire, si aucune instanciation ne satisfait le problème, les graphes comparés ne sont ni égaux, ni inclus.

Dans toute cette partie, le même exemple est utilisé pour illustrer les différentes étapes de l'interface. Cet exemple est celui de la figure 2.40, représentant les deux pièces comparées ainsi que la formalisation en un graphe CSP du problème.



CSP représentant le problème d'égalité des deux graphes G_N et G_C

Figure 2.40. *Représentation formelle d'un exemple particulier*

Les deux pièces considérées sont représentées par leur graphe de contraintes ; le graphe normalisé courant G_N représentant la pièce en cours de construction et le graphe chargé G_C modélise, par exemple, un élément d'une bibliothèque.

Le graphe CSP du problème est conforme à la définition donnée au paragraphe 3.2.1 cependant un certain nombre de contraintes implicites jusqu'alors apparaissent maintenant dans le graphe. Il s'agit des contraintes permettant de modéliser une spécificité du problème d'égalité ou d'inclusion de graphes qui est la suivante : un nœud de G_C ne peut être apparié qu'à un seul nœud de G_N ou autrement dit une valeur du domaine de définition des différentes variables, une fois utilisée, ne peut plus servir à aucune autre instanciation de variables. Ces contraintes sont les contraintes en pointillés de la figure 2.41, ajoutées entre tous les couples de nœuds non encore contraints et constituées de tous les couples possibles constitués de deux entiers différents.

4.6.2. Structure de la maquette : le graphe

Ce paragraphe présente de manière simplifiée, la structure de graphe utilisée par le logiciel de CAO pour modéliser les différentes constructions créées. Comme le montre la figure 2.41, la structure de graphe est constituée de deux entités liées et qui sont : une table de nœuds ainsi qu'une table d'arêtes. La compréhension détaillée de cette structure n'est pas primordiale, par contre, il est important de noter :

- qu'à partir d'un numéro de nœud on peut connaître toutes les arêtes ayant pour origine ou extrémité ce nœud ;
- qu'à partir d'un numéro d'arête on dispose du numéro du nœud origine et du nœud extrémité ainsi que du type de contrainte modélisé par cette arête.

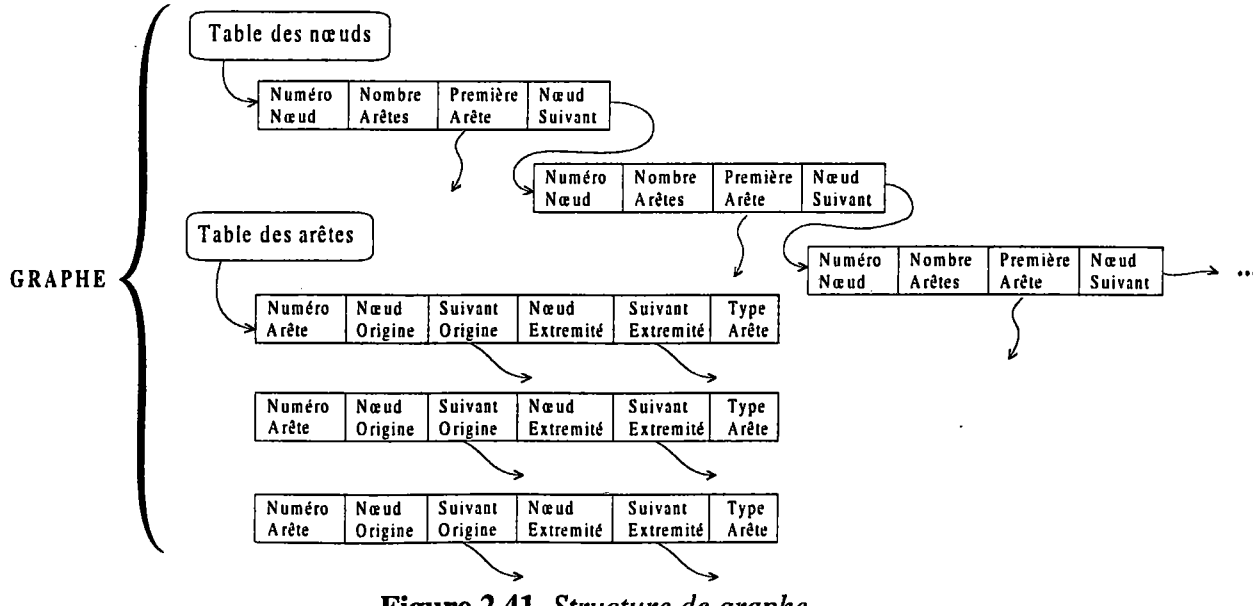


Figure 2.41. *Structure de graphe*

La figure 2.42 représente la structure de graphe de l'exemple considéré et plus particulièrement la modélisation de la pièce en cours de construction.

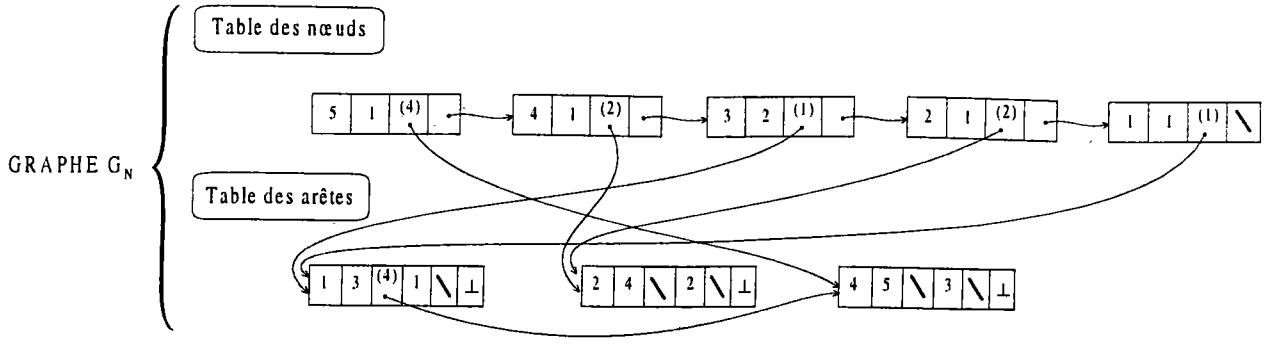


Figure 2.42. *Structure représentant le graphe G_N*

4.6.3. Structure intermédiaire : la table des contraintes

La mise sous forme de CSP du problème d'égalité de graphes nécessite de connaître par type de contrainte tous les couples de nœuds la vérifiant. En effet, nous avons vu que lors de la création du graphe CSP, lorsqu'il existe, par exemple, une contrainte de perpendicularité entre deux nœuds du premier graphe, dans le CSP, l'arête liant ces deux nœuds est étiquetée par tous les couples du deuxième graphe vérifiant cette même contrainte.

La structure de graphe présentée précédemment ne fournissant pas directement ces informations, nous présentons une structure intermédiaire permettant d'optimiser les traitements ultérieurs. Cette structure est une table de contraintes regroupant les couples de nœuds contraints par type de contrainte. Elle est obtenue en réalisant le parcours de la table des arêtes et est celle de la figure 2.43 pour l'exemple considéré.

Contrainte	Couples de nœuds vérifiant la contrainte
$\perp$	(1,2) (2,5) (3,4)
//	*aucun*
tangent	*aucun*

Figure 2.43. *Tableau des contraintes du graphe G_N*

4.6.4. Structure destinée aux algorithmes de résolution de CSP : le fichier texte

Les données nécessaires à l'utilisation d'un algorithme de résolution de CSP sont contenues dans un fichier texte structuré de la façon présentée par la figure 2.44. Ce fichier est constitué de deux parties. La première partie contenant les différentes variables du CSP suivies de leur domaine de définition respectif. Dans le cas particulier que nous considérons, ce domaine de définition est identique pour toutes les variables puisque tout nœud du premier graphe peut être

apparié à n'importe quel nœud du deuxième graphe. La deuxième partie du fichier est constituée de toutes les contraintes du CSP suivies des différents couples de nœuds possibles.

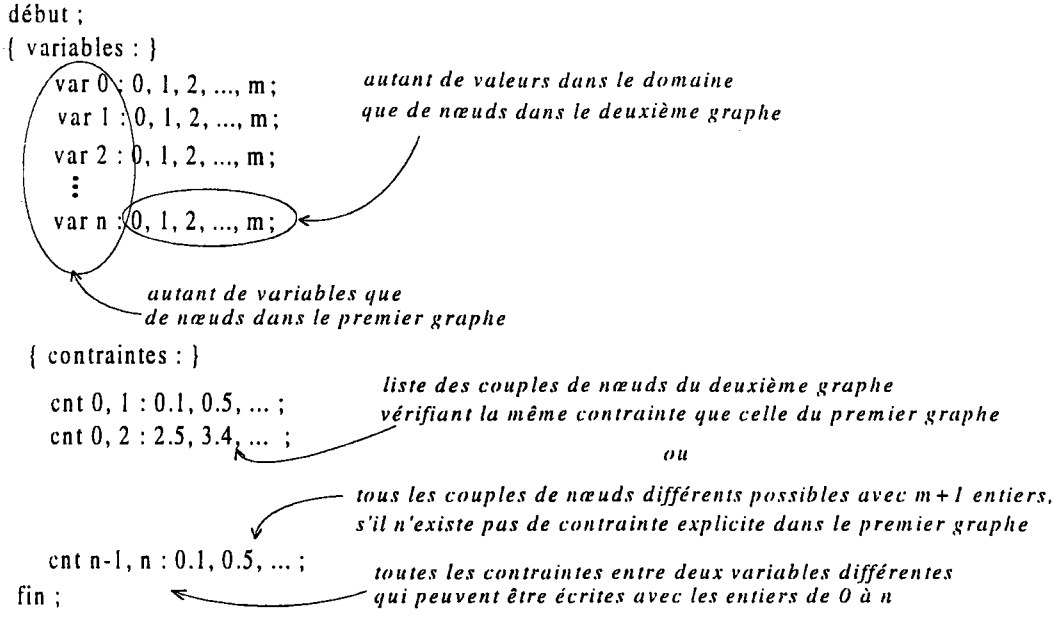


Figure 2.44. *Construction du fichier représentant le CSP*

CréationFichierCSP
{Traduction sous forme de CSP du problème de comparaison de 2 graphes G_D et G_V.
Données : - TableCouplesContraintesD : le tableau des couples de contraintes du graphe G_D ;
- TableCouplesContraintesV : le tableau des couples de contraintes du graphe G_V ;
- NbNœudsD : le nombre de nœuds du graphe G_D ;
- NbNœudsV : le nombre de nœuds du graphe G_V }

```
début
    Ouvrir le fichier en écriture
    Écrire dans le fichier ("début ;")
    Écrire dans le fichier ("{variables : }")
    Pour i de 0 à NbNœudsV-1 faire
        Écrire dans le fichier ("var ", i , " :")
        Pour j de 0 à NbNœudsD-2 faire
            Écrire dans le fichier (j , " ,")
        fin pour
        Écrire dans le fichier (NbNœudsD-1, " ;")
    fin pour
    Écrire dans le fichier ("{contraintes : }")
    Pour i de 0 à NbNœudsV-1 faire
        Pour j de i+1 à NbNœudsV-1 faire
            Si  Couple(i,j) appartient à TableCouplesContraintesV alors
                Écrire dans le fichier ("cnt", i, " ,", j, " : ") suivi de tous les couples de la même
                contrainte de la table TableCouplesContraintesD
            Sinon
                Écrire dans le fichier ("cnt",i, "," ,j,":") suivi des couples possibles avec NbNœudsD
            fin si
        fin pour
    fin pour
    Écrire dans le fichier ("fin ;")
    Fermer le fichier
fin
```

Figure 2.45. *Algorithme de création du fichier modélisant le CSP*

Le fichier texte de la figure 2.44 est obtenu en parcourant notamment la table des contraintes des deux graphes à comparer, plus précisément en appliquant l'algorithme de création du fichier texte représentant le CSP présenté par la figure 2.45. La figure 2.46 est le fichier texte représentant le CSP de la figure 2.44, on constate que les contraintes n'étant pas orientées dans notre problème, tous les couples (n,m) figurent également sous la forme (m,n).

```
{Fichier texte contenant les variables et les contraintes nécessaires à la mise sous forme de CSP du problème. }
début ;

{variables : }
        var 0 : 0 ,1 ,2 ,3 ,4;
        var 1 : 0 ,1 ,2 ,3 ,4;
        var 2 : 0 ,1 ,2 ,3 ,4;
        var 3 : 0 ,1 ,2 ,3 ,4;
        var 4 : 0 ,1 ,2 ,3 ,4;

{contraintes : }
        cnt 0, 1 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
        cnt 0, 2 : 1.0, 0.1, 3.2, 2.3, 4.1, 1.4;
        cnt 0, 3 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
        cnt 0, 4 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
        cnt 1, 2 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
        cnt 1, 3 : 1.0, 0.1, 3.2, 2.3, 4.1, 1.4;
        cnt 1, 4 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
        cnt 2, 3 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
        cnt 2, 4 : 1.0, 0.1, 3.2, 2.3, 4.1, 1.4;
        cnt 3, 4 : 0.1, 1.0, 0.2, 2.0, 0.3, 3.0, 0.4, 4.0, 1.2, 2.1, 1.3, 3.1, 1.4, 4.1, 2.3, 3.2, 2.4, 4.2, 3.4, 4.3;
fin ;
```

Figure 2.46. *Fichier CSP de l'exemple*

5. CONCLUSION

Dans ce chapitre, nous avons présenté deux méthodes permettant aux systèmes de CAO de proposer à l'utilisateur, pendant la phase de conception d'une nouvelle pièce, des constructions ayant des points communs avec cette dernière. Cette fonctionnalité du logiciel va permettre au concepteur d'utiliser efficacement des bibliothèques importantes de pièces existantes.

La maquette réalisée implémente, dans le cadre de la géométrie 2D, la méthode de comparaison de graphes proposée qui s'appuie sur une modélisation des contraintes à l'aide de graphes. En effet, pour montrer l'égalité ou l'inclusion de deux pièces, on prouve l'égalité ou l'inclusion de leurs graphes de contraintes. Les graphes utilisés sont des graphes normalisés, ce qui permet d'avoir un seul représentant pour toute une classe de solutions. Le nombre d'objets à stocker sera donc moins grand et la réalisation des bibliothèques de pièces est ainsi optimisée.

La méthode de normalisation développée est basée sur un système de règles d'ajout ou de remplacement de contraintes qui sont implémentées par des procédures. La suite des ensembles de contraintes obtenue lors du processus de normalisation est une suite convergente ce qui prouve la terminaison de ce processus et l'unicité de la solution.

Les contraintes géométriques ainsi exprimées, pourraient être vues comme un modèle syntaxique qui sera efficace pour déterminer des pièces différentes, mais l'information géométrique sera indispensable pour conclure positivement quant à la ressemblance de deux pièces. En fait, tout dépend du type d'égalité recherchée, si une égalité topologique (même relation entre les différents constituants des deux pièces comparées) est souhaitée, la première étape évoquée est suffisante. Par contre, si une égalité géométrique est désirée, il est nécessaire de consulter les données géométriques relatives aux deux pièces considérées. La première étape permettant d'apparier deux à deux les composants correspondants des deux pièces comparées, il suffit alors de rechercher, pour tous les couples exhibés, une relation dimensionnelle. Cette relation est généralement une composition de transformations affines (rotations, translations, homothéties, ...).

Finalement, nous avons évoqué la possibilité d'une phase de pré-traitements à la modélisation sous forme de CSP ; il serait nécessaire de trouver une série de traitements constituant un bon compromis entre le temps de calcul et le gain effectif. L'optimisation la plus intéressante, mais aussi la plus délicate, est le passage à un modèle 3D. La difficulté majeure étant alors de prouver la complétude du système de règles.

CHAPITRE 3.

MODÉLISATION ET GESTION DES CONTRAINTES FONCTIONNELLES PAR LES SYSTÈMES DE CAO

1. INTRODUCTION

Dans ce chapitre, nous abordons l'expression, la modélisation et la résolution des contraintes fonctionnelles. Les contraintes fonctionnelles comme leur nom l'indique sont liées aux fonctions ou fonctionnalités de l'objet qu'elles contraignent. Elles permettent, dans la plupart des cas d'obtenir une utilisation optimale de l'élément contraint. Elles sont très importantes et présentes dans de nombreux problèmes mais elles sont également difficiles à exprimer et à gérer car elles portent souvent sur des notions relativement abstraites.

Nous avons donc choisi de préciser notre domaine de réflexion en étudiant ce type de contraintes au travers d'une application typiquement CAO qui est l'aménagement spatial d'objets contraints. L'aménagement spatial consiste à disposer dans l'espace les éléments composant la solution du problème posé. Ce problème a déjà souvent été étudié, en aménagement spatial d'atelier, de cuisine mais aussi dans le cadre de l'électronique ; la difficulté consiste alors à répartir sans chevauchement des composants interconnectés, fréquemment assimilés à des rectangles, sur une carte électronique représentée par une surface bornée. Cependant, compte tenu de la forte combinatoire du problème et de la nature conflictuelle de certaines contraintes, il n'existe pas une méthode systématique donnant toujours la meilleure solution. Des méthodes heuristiques, généralement spécifiques au problème à résoudre, sont indispensables et rendent difficile la comparaison de l'efficacité des différentes méthodes.

On peut distinguer deux types d'aménagement, l'aménagement topologique qui consiste à trouver une disposition relative des objets (à droite de, au dessus de, ...) et l'aménagement géométrique qui recherche une disposition des objets dans un espace absolu. Une bonne stratégie de recherche d'aménagement spatial consiste à définir, dans un premier temps, un aménagement topologique puis de l'affiner dans un second temps de manière à obtenir un aménagement géométrique satisfaisant.

Dans une première partie, nous présentons différentes méthodes de résolution utilisées pour résoudre les problèmes d'aménagement spatial. Toutes les données impliquées dans un problème, pour être prises en compte par un logiciel de CAO doivent être modélisées par ce même système. Dans une deuxième partie, nous présentons différentes modélisations envisageables des données de tout problème d'aménagement spatial d'objets contraints c'est-à-

dire, l'espace de placement, les objets et les contraintes. Pour chaque méthode, nous précisons si elle nous semble plus ou moins appropriée à notre problème.

La troisième partie est consacrée à la présentation des méthodes de modélisation mises en œuvre pour les données du problème particulier d'aménagement spatial considéré. Les quatrième et cinquième parties proposent deux méthodes de résolution différentes d'un problème particulier d'aménagement spatial qui est celui de conception de cuisine intégrée. La première méthode est une méthode algorithmique qui disposent de plusieurs optimisations techniques. Quant à la seconde méthode, elle est basée sur le concept de CSP déjà mis en œuvre au chapitre précédent.

Finalement, nous présentons le problème de conception d'une nouvelle usine qui fait intervenir un grand nombre de facteurs fonctionnels moins classiques et pouvant par exemple avoir un impact économique important. Ces différents facteurs sont recensés et de cette liste sont extraits ceux portant plus particulièrement sur la réalisation de l'organisation des différents constituants d'un atelier.

2. DIFFÉRENTES MÉTHODES DE RÉSOLUTION DES CONTRAINTES FONCTIONNELLES

Les méthodes de résolution des problèmes d'ordonnancement sont très nombreuses et souvent très différentes car elles sont inspirées de domaines variés. Toutes les méthodes développées sont dédiées à des problèmes spécifiques et intègrent donc cette spécificité. Cependant, pour éviter d'en faire une présentation par trop liée à la nature du problème, on peut les répartir en quatre groupes : les méthodes dites itératives, les méthodes constructives, les méthodes modulaires et les méthodes utilisant une approche par résolution de contraintes.

2.1. Les méthodes itératives

Dans les méthodes itératives ([HER 95], [MIS 96], [CAU 95], [KYU 92], ...), un placement initial brut, peu coûteux mais souvent non réalisable est amélioré pas à pas. Chaque amélioration individuelle est peu coûteuse et le processus itère cette étape jusqu'à obtenir une configuration acceptable. Différents critères de convergence sont applicables.

Les méthodes itératives procèdent donc par étapes successives et tentent d'améliorer au fur et à mesure une solution initiale. Plus précisément, on distinguera deux types de solutions, les

solutions réalisables et les solutions admissibles [HER 95]. Une solution est réalisable lorsqu'elle satisfait l'ensemble des contraintes du problème ; elle est admissible lorsqu'elle satisfait seulement un sous-ensemble de cet ensemble de contraintes. L'objectif des méthodes itératives est de trouver une solution réalisable qui minimise une fonction F permettant d'évaluer chaque solution réalisable sachant que cet ensemble de solutions réalisables est supposé fini et qu'il est défini par l'ensemble des contraintes du problème.

Ces méthodes itératives essaient donc, à chaque étape de trouver une nouvelle solution réalisable qui améliore la valeur de la fonction objectif, ceci jusqu'à ce qu'un critère d'arrêt soit atteint. Les différents critères d'arrêt peuvent être [HER 95]:

- l'ensemble des solutions admissibles a été testé dans son intégralité ;
- le nombre maximum d'itérations est atteint ;
- le nombre maximum d'itérations sans amélioration est atteint ;
- la valeur de la meilleure solution réalisable rencontrée est suffisamment proche de la borne inférieure connue de la fonction objectif.

Nous venons de présenter de manière générale les méthodes itératives mais il existe autant de méthodes différentes que de façon d'aborder le problème (domaines liés : génétique, mathématiques, logique, physique, métallurgie, géologie, ...). Nous allons présenter brièvement certaines de ces méthodes itératives.

2.1.1. Méthode Tabou

La méthode Tabou exposée dans [HER 95] tente de pallier un des inconvénients des méthodes itératives. En effet, avec ce type de méthode, on rencontre un problème si un minimum local se trouve "au fond d'une vallée profonde" car il est alors impossible de ressortir en une seule itération de cette "vallée" ; la méthode risque de boucler autour de ce minimum local. Pour éviter ce problème, la méthode Tabou propose de garder en mémoire les dernières solutions visitées et d'interdire le retour vers celle-ci pour un nombre fixé d'itérations : on appelle ces solutions, des solutions tabous. La place mémoire requise par cette méthode est importante car elle implique le stockage d'un certain nombre d'informations. Ce stockage de données Tabous est également utilisé par le système ECOPLAN ([MIS 96]) qui s'intéresse aux problèmes d'ordonnancement de "coupe-claire" dans une forêt.

Le problème résolu par le système ECOPLAN peut être formulé de la façon suivante : étant donné une forêt subdivisée en un ensemble de parcelles, c'est-à-dire de régions qui sont

considérées comme homogènes en respect de certains paramètres comme la fertilité des sols, les espèces de bois, l'âge moyen, ...

Le but est de calculer, pour chaque année dans une période de planification spécifiée, un ordonnancement des ensembles de parcelles à couper. Cet ordonnancement doit satisfaire certaines contraintes ainsi qu'un ensemble de critères associés :

- contrainte 2-m : tous les sites voisins d'une parcelle coupée doivent contenir des arbres de plus de deux mètres ;
- consommation uniforme : le volume total de la forêt coupée doit être aussi uniforme que possible d'une année à l'autre ;
- vieille forêt : l'aire totale de vieille forêt doit être conservée au-dessus d'un niveau donné durant une période entière ;
- temps de récolte optimal : la forêt coupée doit être aussi "mûre" que possible, ni trop vieille, ni trop jeune ;
- impact visuel : étant donné un ensemble de points de vue prédéfini dans la campagne, la distribution spatiale des clairières doit minimiser la réduction de la qualité esthétique de la scène.

Pour ce faire, une méthode d'amélioration itérative a été employée, complétée par l'utilisation d'une fonction d'évaluation, d'un opérateur de voisinage, d'un ordonnancement initial, d'un critère Tabou et d'un critère d'aspiration. Ces différentes fonctionnalités incluses par le système permettent de quantifier, d'évaluer et donc de gérer les contraintes énoncées ci-dessus.

Deux inconvénients, pour nous, s'opposent à l'utilisation d'une méthode définissant des solutions tabous. D'une part, une telle méthode permet de ne pas considérer plusieurs fois une mauvaise solution, cependant elle n'empêche pas le système de faire plusieurs fois le même raisonnement aboutissant à une mauvaise solution. D'autre part, la solution courante est comparée aux solutions tabous conservées or nous avons déjà évoqué les problèmes soulevés par la comparaison de modèles de pièces lorsque ces représentations ne sont pas normalisées.

2.1.2. Méthode génétique

Dans [CAU 95], une autre méthode itérative, également coûteuse en espace mémoire, est présentée ; il s'agit des algorithmes génétiques. L'idée sous-jacente aux algorithmes génétiques est de reproduire l'évolution naturelle d'organismes, génération après génération, en respectant les phénomènes d'hérédité et la loi de survie énoncée par Darwin. Selon ces principes, dans une

population d'individus, ce sont les plus forts, c'est-à-dire les mieux adaptés au milieu, qui survivront et pourront donner une descendance.

Dans un algorithme génétique, un individu (une solution) est caractérisé par une structure de données qui représente son empreinte génétique. La force d'un individu est représentée par la valeur de la fonction objectif. Les opérateurs génétiques sont des algorithmes qui permettent de parcourir l'espace des solutions du problème. Une itération de l'algorithme génétique implique un renouvellement (création/destruction d'individus : sélection naturelle) de la population de l'ensemble des solutions courantes.

L'exécution d'un tel algorithme doit conduire, à partir d'une population initiale, après de nombreuses générations, à une population où les individus sont tous très forts, en d'autres termes, à un ensemble de "bonnes" solutions au problème considéré. Pour mettre en œuvre un algorithme génétique, il est nécessaire de disposer [CAU 95] :

- d'une "représentation génétique" du problème (codage approprié des solutions) ;
- d'un moyen de créer la population initiale (tirage au hasard, heuristiques, mélange de solutions heuristiques et aléatoires, ...) ;
- d'une fonction d'évaluation permettant de calculer la force d'un chromosome. La performance de l'algorithme génétique peut être sensible au choix de cette fonction ;
- d'un mode de sélection des chromosomes à reproduire : le principe de la roulette (probabilité d'apparition) permet de retenir les individus les plus forts ; un individu fort peut être sélectionné plusieurs fois, un individu faible a moins de chance d'être sélectionné ;
- d'opérateurs génétiques adaptés au problème (mutation, croisement, ...) ;
- des valeurs pour les paramètres qu'utilise l'algorithme.

La structure de données utilisée pour représenter un individu est un chromosome qui est un vecteur dont les éléments sont des gènes qui appartiennent à l'ensemble {0,1}. Ce mode de représentation permet de décrire des relations d'appartenance, des graphes, des entiers, des réels mais il peut devenir parfois très difficile et très lourd de coder des solutions de cette manière et la place mémoire requise peut devenir prohibitive. L'inconvénient important et constituant un frein pour ce type de méthode est donc l'utilisation du codage binaire pour représenter les solutions ; par contre les algorithmes génétiques ont des particularités intéressantes notamment :

- l'utilisation d'un codage des paramètres et non des paramètres eux-mêmes ;
- la recherche d'un optimum à partir d'un population et non d'un unique individu ;
- les informations sont apportées par la fonction objectif elle-même et non par ses fonctions dérivées ou son gradient ;
- les règles de transitions probabilistes permettant de sortir d'un optimum local.

Une extension possible des algorithmes génétiques est une transformation de ces algorithmes en algorithmes évolutionnistes pour lesquels le codage des solutions et la structure de l'algorithme ne sont pas imposés. Deux types d'extensions peuvent alors être envisagées, les extensions vers des représentations des solutions non binaires et des extensions basées sur des variantes de l'algorithme classique.

À notre avis, les méthodes génétiques s'appliquent assez difficilement aux problèmes traités par les logiciels de CAO comme, par exemple, la création de pièces mécaniques. En effet, le principe général de ces méthodes est de définir un ensemble initial de solutions plus ou moins au hasard et de le faire évoluer vers un ensemble de solutions correctes. Or, cette stratégie pourrait déstabiliser tout utilisateur de logiciels de CAO car elle est contraire à ses habitudes de conception, un concepteur a comme habitude de construire à partir de rien, une solution correcte et non de transformer un élément quelconque de manière à obtenir le résultat souhaité.

2.1.3. Algorithme de placement dirigé par les forces

L'objectif de cet algorithme présenté notamment dans [VER 92], est de contrôler le déplacement ou l'échange des différents objets. Un vecteur de force est associé à chaque objet ; il rend compte de l'ensemble des contraintes portant sur cet objet et indique l'emplacement vers lequel l'objet tend.

La somme des forces de tous les objets définit le coût de la solution. L'algorithme de relaxation dirigée par les forces consiste à choisir un premier objet, le déplacer vers l'emplacement d'équilibre indiqué par son vecteur force, puis à recommencer avec l'objet "délogé". Le processus est répété jusqu'à l'obtention d'un état stable.

En général, les techniques dirigées par les forces fournissent un placement ponctuel qui est modifié pour tenir compte de la taille des blocs.

2.1.4. Algorithme de placement par recuit simulé

Présenté dans [BRO 88], cet algorithme consiste en une génération d'un placement aléatoire des modules puis ce placement est modifié par des perturbations également aléatoires. Ce changement en "énergie" est calculé, si la nouvelle configuration a une énergie plus basse que la précédente, elle est utilisée comme base pour les prochaines perturbations. Par contre, si la nouvelle configuration a une énergie plus grande que les précédentes, elle n'est pas

systématiquement rejetée car elle peut permettre au système de s'extraire d'un minimum local. En effet, le système dispose d'un critère d'acceptation/rejet stochastique qui lui permet de s'extraire lui-même de minima énergétiques locaux. Contrairement aux apparences, expérimentalement, cet algorithme a déjà donné des résultats tout à fait satisfaisant.

2.1.5. Algorithme de "croissance de groupe adaptatif"

Dans [KYU 92], un algorithme de "croissance de groupe adaptatif" (ACG) dans une région rectilinéaire quelconque est décrit. Cet algorithme est destiné au placement de circuits dans une région en raffinant un placement initial. Le principe de cet algorithme peut être comparé au développement d'un cristal dans une cavité. L'algorithme ACG qui requiert comme prédécesseur un processus de placement global, permet donc d'obtenir pour un circuit, un placement optimal détaillé qui considère simultanément, la connectivité du circuit, la taille et la forme du composant électronique, ainsi que la géométrie de la région. Le principe de l'algorithme peut être résumé en six étapes :

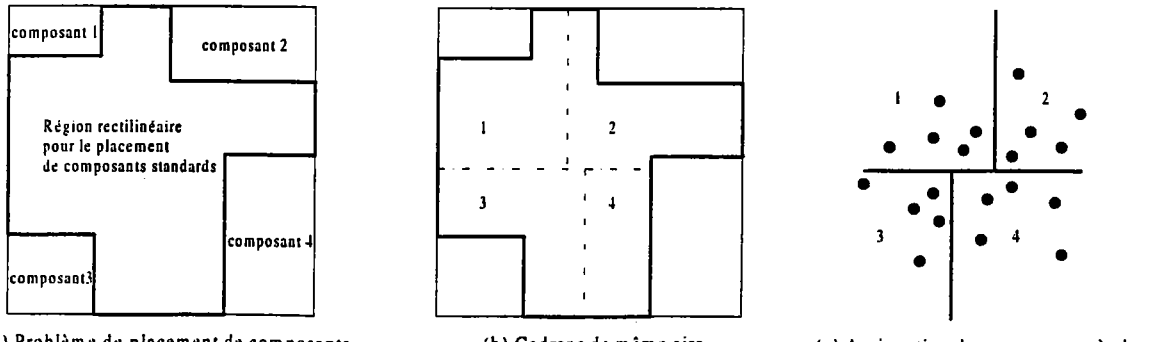
Étape 1 : Après le placement global, la position du centre de masse de chaque composant (figure 3.1a) est représentée par un point.

Étape 2 : L'échantillon de points ainsi obtenu est divisé en quatre cadrans tels que les sommes des aires des composants dans chaque cadran soient environ les mêmes (figure 3.1b).

Étape 3 : Le polygone rectilinéaire est divisé en quatre cadrans tel que chacun d'eux puissent contenir tous les composants qui lui sont assignés. Cette décomposition est très importante pour maintenir la connectivité globale et réduire le temps de calcul (figure 3.1c).

Étape 4 : Alors le point (composant) placé le plus près du centre de masse du polygone rectilinéaire est choisi comme cellule germe, et il est placé au centre de masse.

Étape 5 et 6 : Le reste de la procédure ACG consiste à répéter la sélection des emplacements et celle du composant à placer au site sélectionné jusqu'à ce que tous les modules soient placés.



(a) Problème de placement de composants standards sur une puce rectilinéaire

(b) Cadrans de même aire

(c) Assignation des composants à chacun des cadrans

Figure 3.1. *L'algorithme de croissance de groupe adaptatif*

La région totale est divisée en un certain nombre de carreaux (figure 3.2) et parmi tous les sites à la périphérie du cristal grandissant, le site p qui produit la valeur maximale pour $f_S(p)$ est choisi. fs(p) est le minimum de la distance de l'emplacement p au mur face à chaque direction d'accroissement valide (figure 3.3).

Si dr(p) est la distance au mur opposé à partir de l'emplacement p dans la direction r, pour k par exemple on a : $f_S(k) = \min(d_n(k),d_e(k),d_s(k)) = d_e(k)$

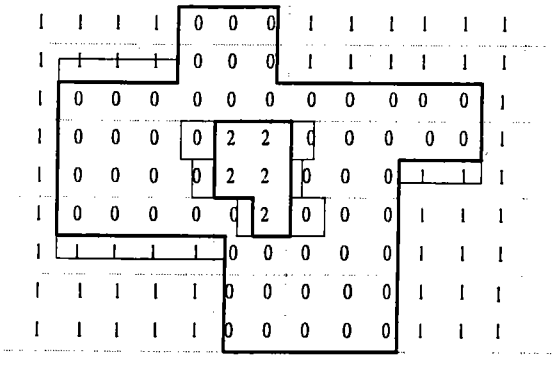


Figure 3.2. *Division d'une région en carreaux*

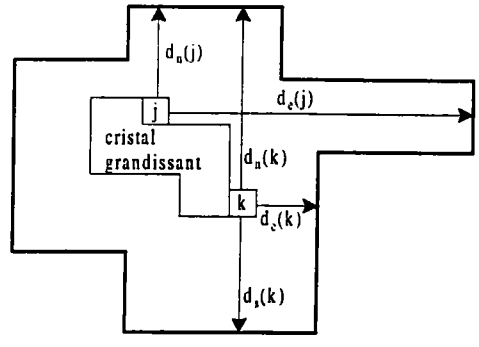


Figure 3.3. *Directions d'accroissement*

1 = tous les carreaux à l'extérieur de la région rectilinéaire (l'aire perdue incluse)

2 = les carreaux entièrement recouverts par le groupe physique

0 = les autres

Ce choix implique que l'accroissement du cristal se fait dans une direction tel que sa forme s'adapte toujours à la forme de la cavité. De plus, la modélisation pour cette méthode des espaces par des carreaux permet :

- une réduction de la place mémoire nécessaire ;
- une réduction du temps de calcul d'une distance à la frontière de la région. Il suffit de traverser dans une direction jusqu'à un carreau avec un 1 ou un 2 et d'additionner le nombre de carreaux traversés à l'occupation ;
- une réduction du temps de recherche du module le plus proche d'un périmètre donné.

Cette méthode nous apparaît comme très intéressantes car elle permet de définir le placement d'un composant rectilinéaire (mais pas obligatoirement rectangulaire) dans un espace déjà bien encombré. Simplement, on peut regretter de ne pas disposer d'une modélisation exacte de l'espace car la division en carreaux implique d'inévitables imprécisions (cf. figure 3.2).

La méthode utilisée par le système présenté dans [VER 90] réalise le placement de rectangles par répartition dans une surface bornée en seulement deux étapes : la première étape recherche un placement ponctuel (dimensions des blocs non prises en compte) qui reflète la connectivité du circuit (minimisation de la longueur totale des liaisons entre les blocs) , la seconde étape est une amélioration itérative du placement initial, les blocs sont déplacés suivant leurs adjacences respectives.

Le principe des méthodes itératives est suivi par ce système. À chaque itération, tous les blocs sont déplacés et la répartition obtenue doit respecter au mieux la configuration initiale. Chaque bloc est déplacé au centre de sa zone de déplacement (figure 3.4). Le processus s'arrête lorsque les déplacements des blocs ne sont plus significatifs. Cette méthode fournit assez rapidement une répartition correcte des blocs à l'intérieur de la surface de placement.

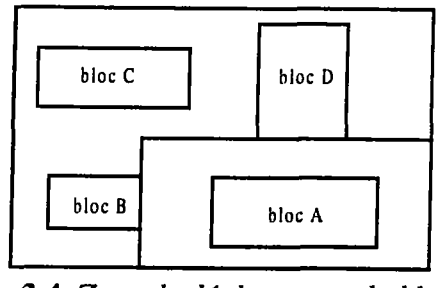


Figure 3.4. *Zone de déplacement du bloc A*

Pour les problèmes que nous traitons, les méthodes itératives n'apportent pas de réels avantages. En effet, elles sont très intéressantes essentiellement pour les systèmes qui peuvent être interrompus en cours de recherche ce qui n'est généralement pas le cas des logiciels de CAO. De plus, l'efficacité de la méthode dépend essentiellement de la définition de la fonction objectif qui est constituée en grande partie des informations contenues dans le graphe de contraintes que nous manipulons. Ce type de méthodes nous demanderait donc un travail de reformulation supplémentaire non trivial pour un apport qui ne constitue par une fonctionnalité réellement nécessaire pour les logiciels de CAO.

2.2. Les méthodes constructives

Ces méthodes désirent structurer la recherche d'une solution optimale à un problème donné ce qui permet notamment des retours arrières plus faciles lors d'un échec. Il existe différents "découpages" possibles du problème d'ordonnancement. Une méthode utilisant un découpage chronologique du problème sera appelée méthode constructive.

Dans [BRO 88], un algorithme constructif est défini de la façon suivante : c'est un algorithme qui essaie d'obtenir des choses justes du premier coup. Il passe une grande partie de son temps à calculer le placement de chaque module, ces modules sont placés l'un après l'autre à une position qui n'est généralement pas très éloignée de la solution finale.

Le système ORANOS ([KWA 97]) est un solveur de contraintes incrémental qui utilise une technique de propagation d'intervalles pour maintenir un ensemble de contraintes hiérarchiques qui sont nécessaires à la résolution de problèmes sur-contraints. La méthode présentée permet à un utilisateur de décrire simplement les relations entre objets lors de la conception d'une scène 3D. Cette méthode est à la fois déclarative et interactive ; déclarative car le concepteur peut exprimer des relations complexes entre objets grâce aux différents types de contraintes dont il dispose et interactive car il est possible de supprimer ou ajouter des contraintes par l'intermédiaire d'une interface.

Le système présenté se propose de résoudre le problème de placement d'objets. Le concepteur construit une scène pas à pas, cette scène est alors représentée comme un ensemble d'objets contraints sur lesquels on applique des contraintes spatiales. ORANOS se charge alors de satisfaire ces contraintes en respectant les priorités. Le problème de placement est un problème NP_complet pour lequel le système peut trouver un ensemble de solutions, il peut les afficher successivement au concepteur de façon à ce qu'il puisse en choisir une. Après avoir fait son choix, il est possible de le modifier en ajoutant des contraintes dérivées.

Les auteurs définissent le système ORANOS comme déclaratif, interactif et convivial, nous émettons quelques réserves sur la notion de convivialité. En effet, supposons un problème constitué de dix contraintes (neuf de priorités i et une de priorité i+1), si la contrainte de priorité i+1 ne peut être satisfaite, aucune solution ne sera proposée même s'il existe une solution satisfaisant les neuf contraintes de priorités i. Cette situation peut être considérée comme frustrante pour l'utilisateur même s'il a la possibilité en diminuant la priorité de la contrainte d'obtenir éventuellement une solution.

Le système OSC-ART ([CLE 87]) propose de réaliser également un placement d'objets, les objets considérés sont des composants de cartes électroniques. Le problème est résolu en deux étapes, une étape d'ordonnancement puis une étape de placement. Le placement effectué par OSC-ART est constructif, ce qui consiste à placer les objets les uns après les autres dans un ordre déterminé. Une liste de modules est construite en prenant en compte différents critères précisés

par l'utilisateur. Ces critères peuvent être aussi divers que la taille, le poids, le degré de connectivité, la dissipation de chaleur, ...

Les méthodes constructives élaborent donc pas à pas une solution correcte. Leur inconvénient majeur est qu'elles ne peuvent pas être interrompues durant la recherche contrairement aux méthodes itératives qui peuvent fournir à tout instant une solution plus ou moins optimisée. De plus, elles peuvent être bloquées par la non résolution d'une contrainte.

2.3. Les méthodes modulaires

Il existe finalement un troisième type de méthodes constitué des méthodes modulaires qui représentent en quelque sorte une méthode hybride. Ces méthodes considèrent un découpage fonctionnel du problème, chaque niveau de résolution fournissant une solution intermédiaire. Ainsi, à chaque niveau de résolution, la solution peut être recherchée de manière itérative.

2.3.1. Distinction de deux phases : analyse / décision

Certains systèmes de CAO scindent leur processus de résolution en deux étapes. Généralement, la première étape correspond à une phase d'analyse et la deuxième à une phase de décision. Ces deux phases peuvent être considérées quel que soit le domaine d'application du logiciel de CAO. Nous présentons dans cette partie, des systèmes employant cette décomposition.

Le système CADOO [AND 86] dispose de connaissances expertes qui lui permettent de restreindre sensiblement l'espace de recherche. Il a également des stratégies générales qui lui permettent de sélectionner parmi les alternatives restantes, celles qui satisfont le mieux les contraintes considérées, tout en contraignant le moins possible la suite de résolutions.

CADOO gère un plan d'aménagement qui lui permet d'éliminer par construction, les solutions physiquement impossibles et d'exprimer une part de l'expertise en terme de concepts spatiaux. Ce système est capable également de générer automatiquement des contraintes et de visualiser dynamiquement le processus de construction de la solution.

De façon plus précise, le système procède de la façon suivante : à chaque étape de la recherche, un paramètre de conception est sélectionné ainsi qu'une contrainte qui le conditionne et ceci à l'aide d'un degré d'incertitude associé dynamiquement à chaque contrainte. Ce degré

d'incertitude exprime le nombre d'alternatives possibles satisfaisant la contrainte en question. Les conséquences d'un choix sont alors propagées ; si au cours de cette propagation, un échec intervient, on effectue un retour arrière (backtrack) pour choisir un autre paramètre de conception. Grâce à l'interface conviviale du système, lors d'un processus de retour arrière dû à un échec de la propagation, l'utilisateur peut en suivre toutes les "péripéties" ; il est même envisagé de permettre à ce dernier d'intervenir, par exemple, en désignant l'élément qui lui semble préférable de déplacer.

La réalisation du placement est obtenue à l'aide d'un ensemble d'heuristiques dépendantes des contraintes et qui permettent de focaliser la recherche de places sur des zones privilégiées du compartiment. Les places envisagées sont évaluées par rapport à chaque contrainte non-impérative et à certains critères d'optimisation. Pour économiser l'espace libre et donc contraindre le moins possible le placement des autres éléments, on va essayer de partager les espaces de manutentions, de coller aux bords et d'occuper les places isolées. Le mécanisme qui permet la sélection d'un élément par l'application successive de critères est un tri multi-critère, cet algorithme de tri multi-critères est appliqué sur la liste des places obtenues. La meilleure place est occupée et les meilleures alternatives sont mémorisées en prévisions d'éventuels retours arrières du processus en cas d'échec.

Ce système coopérant avec l'utilisateur nécessite de manière impérative de stratégies heuristiques et plutôt qu'une méthode de propagation de contraintes classiques, c'est une méthode de résolution guidée par des stratégies générales influencées par les contraintes.

L'approche présentée par le système CADOO, nous semble tout à fait pertinente, notamment la prise en compte de connaissances expertes et la définition de méta-critères. Les connaissances expertes permettent d'assister la stratégie générale de résolution de problèmes. De plus, nous pensons que solliciter l'utilisateur final pour la prise de certaines décisions comme le fait CADOO n'est pas une mauvaise chose. En effet, ceci permet de guider précisément la recherche tout en permettant au concepteur de suivre l'évolution de la construction. Cette consultation est surtout possible si l'utilisateur du système est un expert du domaine car il sera tout à fait disposé à s'impliquer dans une construction ; si au contraire l'utilisateur n'est pas un spécialiste, le système devra éventuellement prendre plus d'initiatives.

Nous émettons simplement quelques réserves quant à la possibilité de déduire toutes les contraintes du problème à partir d'une description de chaque élément. En effet, le système

CADOO se propose lors d'une première étape de générer pour chaque élément à placer, les contraintes qui pèsent sur lui. Les données sont alors, pour chaque cas d'aménagement la description des éléments à placer. Cette description se fait à l'aide d'une liste d'attributs qui n'est pas fixée a priori de sorte que l'utilisateur peut en créer autant qu'il le souhaite. Un ensemble de contraintes génériques du domaine permettent de générer les contraintes propres à chaque élément. Il nous semble difficile, pour les contraintes génériques, de gérer des attributs quelconques : il n'existe apparemment pas de lien entre un ensemble d'attributs possibles et un domaine d'application particulier. Sauf si, l'utilisateur en créant son attribut crée également la contrainte générique associée.

Le système CADOO présenté ci-dessus, distingue deux phases dans son processus de résolution mais aucune de ces dernières n'autorise la prise en compte d'événement extérieurs. Dans le but d'améliorer l'efficacité de la conception, le système de CAO doit vérifier interactivement les contraintes durant le processus de conception. L'objectif du système présenté dans [KAM 92] est de mettre à jour automatiquement les données et de vérifier les contraintes ou d'évaluer la conception tandis que le concepteur dessine un plan ; il peut dans ces conditions, se concentrer sur la partie créative de la conception. Cette conception est très complexe, le concepteur doit décider des positions des zones d'aménagements et des équipements tels que les positions satisfassent un grand nombre de contraintes (distance, position, adjacence, orientation, espace, ...). Ainsi que, la plupart des contraintes résultant des requêtes des autres processus de conception, telle que la construction de conception et d'autres aspects de conception d'ingénierie tel que la condition du site, la maintenance du matériel, la sécurité des opérations et les coûts de construction. Lorsque les relations entre ces contraintes sont très complexes, il est difficile et coûteux en temps pour un unique concepteur de vérifier toutes les contraintes pendant le processus de dessin. Un système de CAO qui peut vérifier interactivement les contraintes durant le processus de conception est donc dans ce cas fortement requis.

Le système présenté a pour vérifier en temps réel les contraintes, trois types de mécanisme :

- un mécanisme pour analyser et interpréter les actions de dessin du concepteur : c'est une manipulation directe de l'interface utilisateur qui permet de retrouver les différentes données à traiter ;
- un mécanisme pour la mise à jour des données de conception c'est-à-dire une mise à jour de la topologie ;

- un mécanisme conduit par les événements : le système dispose d'un mécanisme qui observe toujours les données de conception et exécute automatiquement une procédure associée si aucune violation de contrainte apparaît.

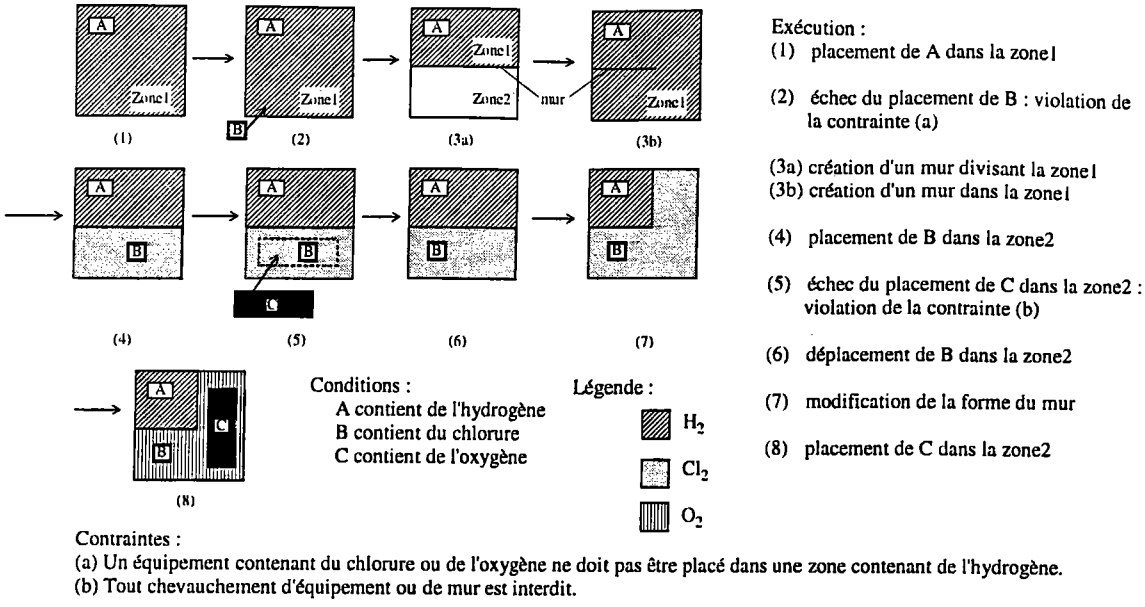


Figure 3.5. *Exemple d'aménagement industriel*

Ce système est comme beaucoup de systèmes basés sur l'utilisation de règles propres à un domaine d'application : pointu pour un problème de son domaine et inefficace pour un problème d'un autre domaine. On peut regretter que ce système se contente de valider ou d'invalider des placements choisis par un utilisateur et qu'il ne propose pas ses propres solutions. En effet, dans l'exemple de la figure 3.5, c'est le concepteur qui trouvent les différentes "astuces" (construction d'un mur, modification de sa forme, ...) de manière à satisfaire toutes les contraintes de son problème.

2.3.2. Approche metadesign

Comme nous l'avons vu précédemment, en utilisant ABD, un système pour architectes de conception automatisée de constructions, le concepteur ne génère pas un plan, mais un ensemble de règles, qui a leur tour, sont utilisées pour générer un ensemble de plans satisfaisant. Cette approche de metadesign est importante pour trois raisons :

- il est plus facile de modifier l'ensemble des règles que le plan ;
- le système peut générer des solutions correctes auxquelles l'utilisateur n'avait pas pensé ;
- les différents plans peuvent être évalués.

L'algorithme du système ABD va permettre de construire un plan ce qui consiste à représenter les murs définissant une pièce. Le plan calculé par l'algorithme du chemin le plus long est

représentatif d'une classe entière de plans, chacun d'eux correspond au même vecteur de valeurs de décision (chaque vecteur de valeurs de décision correspond à un plan particulier). Les autres plans de cette classe ont les mêmes relations topologiques que celles du plan calculé. Cependant, ils vont différer dans les dimensions et la localisation des pièces. Pour obtenir les autres plans, il suffit de placer certains murs différemment en respectant leur intervalle de tolérance.

Dans le processus de recherche, il est possible de créer un état illégal, qui est un état avec des contraintes conflictuelles qui ne peuvent pas être satisfaites simultanément. Quand on trouve un état de recherche illégal, nous n'élaguons pas l'arbre. Si l'état illégal est dû à des valeurs de décision fixées, alors tous les états dans le sous-arbre ayant pour racine cet état particulier seront illégaux également.

La fonction heuristique donne une faible évaluation à de tels états et ces branches ne seront donc pas explorées en priorité. Si malgré la faible évaluation, un état illégal devient l'état de recherche le plus prometteur, certaines des contraintes en conflits seront supprimées.

Un facteur crucial dans l'approche présentée est la fonction heuristique. Mieux la fonction évalue les différentes étapes de recherche, le plus rapidement elle guide la recherche vers l'état but. L'efficacité du système est donc dépendante de la qualité de la fonction heuristique or une fonction très performante pour un problème donné peut l'être beaucoup moins pour un autre. D'où une efficacité constante de la méthode non garantie.

L'espace de recherche étant très grand, on cherche à accélérer la recherche (2 méthodes indépendantes). On peut supprimer des portions substantielles de l'espace de recherche quand elles sont considérées comme (trop étroites) peu prometteuses. La mise en évidence de caractéristiques de base du problème implique une mise en évidence de la divisibilité de l'arbre de recherche ; le problème initial peut donc être décomposé en plusieurs sous-problèmes.

2.3.3. Modèle de planification distribué

Le système présenté dans [CAO 90] traite le même problème d'aménagement spatial que le système ABD en utilisant une planification à différents niveaux. Trois types de tâches : la prise de décision, l'inférence de contraintes et la résolution de conflits. Prendre une décision produit un nouvel état de planification en introduisant une nouvelle supposition, tandis que les inférences de contraintes réalisent des déductions à partir de la supposition nouvellement introduite.

La méthode proposée pour trouver une solution est similaire à la résolution par un groupe de concepteurs d'un problème important et compliqué. Le groupe principal extrait un sous-problème pour chaque concepteur. Après des négociations avec les autres concepteurs une solution rudimentaire est définie et le groupe principal attribut à chaque concepteur un ensemble de contraintes acceptées par le processus de négociation. Chaque concepteur trouve une solution et la soumet au groupe principal, si un des concepteurs ne parvient pas à résoudre son sous-problème, il prévient le leader qui immédiatement renégocie le problème. Le leader n'a pas besoin de connaître les méthodes utilisées par chacun des concepteurs, il est uniquement intéressé par les solutions proposées.

Ce modèle de planification distribué nous semble bien complexe et coûteux pour la conception de plans de maisons. En effet, sur un tel exemple, la résolution en parallèle qui impose une gestion des consultations des données mais aussi de nombreuses négociations et vérifications lors de la centralisation des différents résultats. De plus, un tel problème peut difficilement être décomposé en sous-problèmes indépendants car toute décision influence les décisions ultérieures. On peut donc supposer que les échecs du processus n'interviendront pas forcément en début de processus mais lors du regroupement des résultats fournis par les différents processus.

Dans cette partie, nous avons présenté un certain nombre de méthodes permettant de résoudre les problèmes d'aménagement ou d'ordonnancement. Il n'existe pas une méthode permettant de trouver toujours la meilleure solution. En effet, à cause de la forte combinatoire de ce type de problème, toutes les solutions ne peuvent pas être recherchées, évaluées et triées. Des heuristiques et des méthodes algorithmiques doivent être choisies pour éliminer un certain nombre de solutions impossibles ainsi que pour guider la recherche vers la meilleure solution le plus rapidement possible. Le caractère heuristique de ces méthodes entraîne une forte liaison de l'efficacité de la méthode et du domaine d'application. En effet, plus les heuristiques sont générales, plus le nombre de problèmes résolus par la méthode est important mais moins l'efficacité est grande. Ou encore, plus les heuristiques utilisées concernent des caractéristiques bien précises du problème, plus elles sont efficaces mais moins elles sont réutilisables pour d'autres problèmes.

2.4. Les méthodes par satisfaction de contraintes

Dans [VER 92], le problème d'aménagement spatial est résolu par des techniques de satisfaction de contraintes. Pour ce faire, un module de satisfaction de contraintes géométriques capable de supprimer les configurations inconsistances dans le domaine des objets en fonction des contraintes géométriques à satisfaire, est spécifié. Ce module doit également intégrer l'aspect dynamique du processus de résolution : la mise à jour du domaine des objets est effectuée lors de la modification (ajout ou retrait) des contraintes et/ou des variables.

La génération de contraintes est confiée à un système à base de connaissances dont les règles expriment des connaissances générales d'aménagement spatial et d'expertise propre au domaine d'application. À partir des spécifications du problème et de l'état d'avancement de la résolution, les décisions provoquent l'ajout de contraintes reflétant à la fois les processus de génération d'hypothèses et de complément des spécifications du problème.

Un module de relâchement est appelé lorsque l'ensemble des contraintes est inconsistant. Son rôle consiste à choisir une ou plusieurs contraintes à relâcher afin de résoudre le conflit courant. Pour ce faire, il réalise un diagnostic de l'échec puis sélectionne des substitutions de contraintes parmi celles proposées par une base de connaissances de substitutions. La figure 3.6 expose l'architecture générale du système proposé.

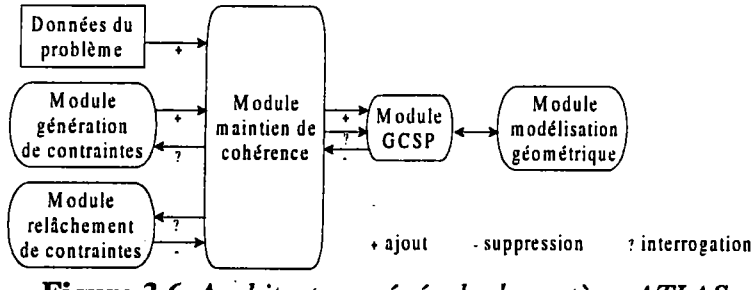


Figure 3.6. *Architecture générale du système ATLAS*

D'un point de vue théorique, l'approche proposée est très intéressante, de part sa forte formalisation, elle apparaît comme une méthode claire et structurée. Nous émettons quelques craintes quant à son efficacité sur des problèmes de grande taille. En effet, de nombreux modules utilisant différentes structures de modélisation coopèrent par l'intermédiaire d'interfaces. Les multiples représentations des données et les diverses transformations ralentissent probablement considérablement l'exécution de la méthode.

De plus, de part notre propre expérience (cf chapitre 2), nous avons pu constater que les méthodes algorithmiques qui apparaissent a priori comme plus confuses peuvent être plus efficaces que les méthodes utilisant le formalisme CSP et a fortiori la notion de CSP dynamique qui demande de fréquentes vérifications de cohérence des différents domaines. Néanmoins, ces remarques sont particulières à un cas étudié et ne peuvent être généralisées à tous les problèmes compte tenu de la forte influence des heuristiques et des méthodes de simplification sur l'efficacité de la méthode de résolution mise en œuvre.

2.5. Synthèse

Dans cette partie, nous avons pu constater que quel que soit le type d'objets manipulés (pièces, composants, parcelles) ils sont souvent représentés simplement par des rectangles. Nous nous attacherons à manipuler, par la suite, des objets de forme plus complexe. La gestion des contraintes géométriques et topologiques est relativement bien maîtrisée contrairement à celle des contraintes fonctionnelles qui demandent encore un gros effort de modélisation et de prise en compte lors de la recherche de solution.

Les méthodes de résolutions, pour la plupart paramétriques, sont nombreuses et comme nous l'avons vu précédemment elles peuvent difficilement être évaluées. Cependant, on peut dire que les méthodes itératives sont très efficaces pour les problèmes d'électronique et que les méthodes modulaires sont bien adaptées aux problèmes d'ordonnancement. Pour le problème d'aménagement spatial qui nous intéresse, une méthode efficace serait probablement une méthode hybride décomposant le problème en étapes, incluant des heuristiques efficaces et une technique tantôt itérative tantôt constructive.

3. LES COMPOSANTS DANS UN PROBLÈME D'AMÉNAGEMENT SPATIAL SOUS CONTRAINTES

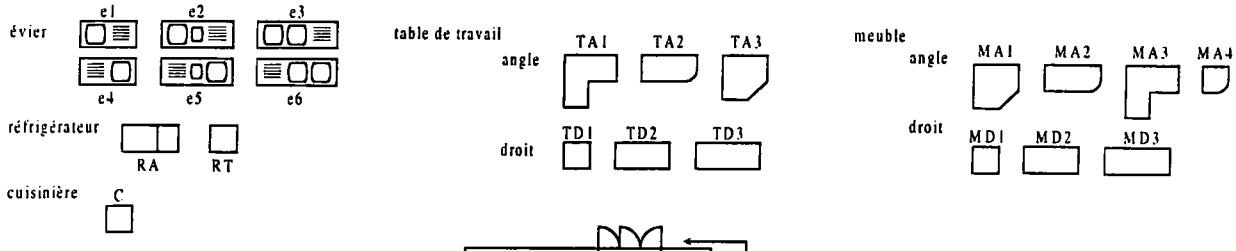
Notre objectif est de réaliser un système permettant de placer des éléments dans un espace libre en respectant au mieux un certain nombre de contraintes géométriques, topologiques et fonctionnelles. Le problème d'aménagement spatial d'objets contraints est constitué de trois types de données : des objets à placer, un espace de placement et un ensemble de contraintes liant les deux éléments précédents. Or tout problème pour être correctement résolu doit être précisément modélisé. C'est pourquoi, dans cette partie, nous présentons différentes méthodes permettant de

modéliser ces trois types de données après avoir donné un exemple du type de problème que nous nous attacherons à résoudre dans les parties suivantes.

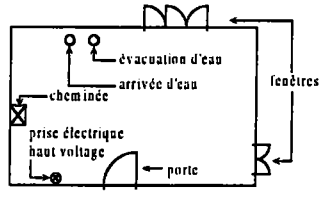
3.1. Présentation générale du problème

Dans cette partie, nous présentons brièvement un problème d'aménagement spatial qui est celui de placement de meubles ou d'appareils dans un espace représentant une cuisine. Ces éléments peuvent être liés par des contraintes ou encore contraints par certaines caractéristiques de l'espace de placement devant contenir l'organisation finale. Nous avons besoin pour résoudre ce problème, de structures permettant de représenter les différentes entités à manipuler ; il est donc indispensable de disposer d'un modèle de représentation pour chaque type de données considéré. Trois sous-parties sont ainsi consacrées à une présentation de modèles possibles dédiés respectivement à la représentation des objets, des espaces et des contraintes.

Les objets



L'espace



Les contraintes

priorité forte : - distance nulle entre deux meubles droits consécutifs
- distance nulle entre un meuble droit et un meuble d'angle consécutifs
- hotte au-dessus de la cuisinière
- évier près de l'arrivée d'eau
- au moins une table de travail près de l'évier

priorité moyenne : - cuisinière près de l'alimentation haut voltage
- évier près de l'évacuation d'eau
- hotte près de la cheminée

priorité faible : - évier sous fenêtre
- distance non nulle entre la cuisinière et le réfrigérateur

Aménagement spatial des objets en respectant les différentes contraintes

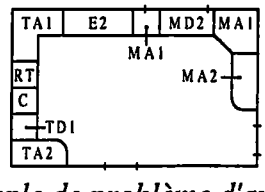


Figure 3.7. *Un exemple de problème d'aménagement spatial*

La figure 3.7 présente un exemple du type de problèmes étudiés par la suite et permet de se familiariser avec les différentes données manipulées ultérieurement. Elle contient pour ce problème d'aménagement spatial, une présentation :

- d'un échantillon d'objets disponibles : différents types de meubles et appareils ;
- d'un espace de placement possible : cuisine ;
- des différentes caractéristiques importantes de la cuisine pour le problème qui nous intéresse : évacuation et arrivée d'eau, cheminée, prise haut voltage, fenêtres, portes, ...
- d'un sous-ensemble des contraintes de priorités différentes utilisées ;
- d'une solution possible.

Il est important de remarquer que ce problème d'aménagement de cuisine peut facilement être transposé dans le domaine industriel ; en effet, on pourrait résoudre de la même façon les problèmes d'aménagement d'atelier.

3.2. Modélisation des contraintes en aménagement de cuisine

Comme nous l'avons vu dans l'exemple d'aménagement de cuisine qui a été présenté en introduction de cette partie, un certain nombre de contraintes doivent être prises en compte lors de la résolution de tout problème d'aménagement spatial. Nous avons pu constater lors de notre étude bibliographique que les différentes contraintes sont souvent réparties en trois catégories (les contraintes géométriques, topologiques et fonctionnelles) et peuvent être de priorités différentes. Nous allons préciser ci-dessous ce que nous entendons dans notre cas par contraintes géométriques ou dimensionnelles, topologiques et fonctionnelles et illustrer ces définitions par des exemples significatifs.

3.2.1. Les contraintes fonctionnelles

Parmi les contraintes à respecter, les contraintes fonctionnelles correspondent souvent à des notions relativement abstraites et elles sont liées à l'environnement ou éventuellement à d'autres objets. Elles permettent d'exprimer relativement directement des demandes de l'utilisateur si toutefois elles ont été prévues par le système et si en conséquence, une structure de données leur a été dédiée. Dans le cas contraire, il est courant d'interpréter la contrainte fonctionnelle et de la réécrire sous la forme d'une combinaison de contraintes topologiques et géométriques. La prise en compte des contraintes fonctionnelles par les systèmes n'est donc pas encore systématique car il est difficile de modéliser pratiquement des notions souvent abstraites.

Par exemple, la fonction d'une hotte est d'aspirer les émanations dues au chauffage d'aliments par les plaques de cuisson. On peut en déduire la contrainte topologique suivante : "la hotte doit se trouver au-dessus des plaques de cuisson" ou par la suite pour une solution plus précise une contrainte géométrique telle que "la distance optimale entre une hotte et une plaque de cuisson est de 60 cm".

3.2.2. Les contraintes topologiques

Ces contraintes permettent d'exprimer des contraintes de positionnement entre deux éléments sans préciser de valeur numérique. Elles sont généralement exprimées à l'aide de notions prédéfinies comme "au-dessus de", "en dessous de", "à gauche de", "à droite de", ... :

- la poubelle en dessous de l'évier ;
- la table de travail à côté de l'évier ;
- la hotte au-dessus de la cuisinière ;
- l'évier devant la fenêtre ou une source lumineuse.

3.2.3. Les contraintes dimensionnelles

Ces contraintes permettent de préciser les valeurs numériques correspondant aux différents paramètres dimensionnels liant les divers éléments. Ces paramètres peuvent être spécifiques à un objet ou à une relation objet/objet ou encore objet/espace.

Dans le cas de paramètres spécifiques à un objet, ils font partie, soit des informations de base définissant l'objet, soit des données complémentaires destinées à la spécification du problème traité. Par exemple, la valeur numérique correspondant à la longueur, à la largeur ou à la hauteur d'un meuble est une contrainte dimensionnelle portant sur une information de base de l'entité et elle est fixée par le fabricant. Par contre, préciser que la surface de la table de la cuisine réalisée doit avoir une certaine valeur est une information destinée à l'application et cette évaluation de la surface n'est réalisée que pour vérifier cette contrainte.

Les relations entre objets sont souvent des contraintes portant sur les distances entre les différentes entités. Par exemple, la contrainte "h = 10cm où h est la hauteur entre la hotte et la cuisinère" est un exemple de relations objet/objet. Par contre, "d < 1m où d est la distance entre l'évier et l'arrivée d'eau de la pièce" est une relation objet/espace.

Dans certains cas, l'utilisateur ne désire pas fixer une relation de ce type de manière précise (par une valeur numérique) mais préfère définir un positionnement "flou" des entités les unes par rapport aux autres. On définira alors des contraintes topologiques.

3.3. Modélisation de l'espace

Comme nous avons pu le constater précédemment, la modélisation géométrique de l'espace de placement est indispensable ; par contre, dans la plupart des cas, elle peut se permettre de ne représenter qu'une approximation de l'espace considéré. Nous allons, dans cette partie, présenter trois approches de modélisation différentes : la modélisation par énumération spatiale, celle par partitionnement et finalement, celle par placement topologique.

3.3.1. Modélisation par énumération spatiale

La modélisation géométrique par énumération spatiale consiste à décomposer l'espace d'aménagement en cellules élémentaires occupées ou vacantes. Une telle modélisation d'une forme géométrique provoque généralement une approximation de sa forme réelle. On peut distinguer, la discrétisation à pas fixe où les cellules sont identiques de celles à pas adaptatif où la taille des cellules dépend de la forme modélisée.

Dans une discrétisation à pas fixe, le découpage correspond à la répétition d'un motif permettant d'obtenir une précision géométrique suffisante de l'approximation. En général, la décomposition est orthogonale ce qui revient à utiliser des cellules carrées en 2D. Bien que non utilisées à notre connaissance en aménagement spatial, notons qu'il existe d'autres types de discrétisations (figure 3.8) facilitant le placement d'objets bénéficiant d'orientations particulières à 45°, 60° ou 120°.

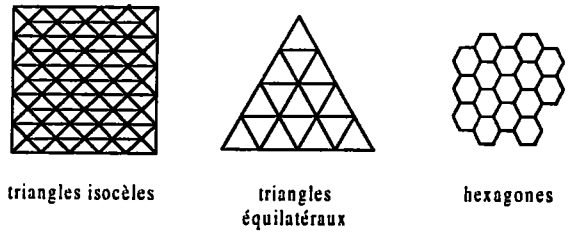


Figure 3.8. *Discrétisation à pas fixe*

La figure 3.9 propose deux représentations de cette discrétisation à pas fixe : par une matrice de valeur ou par une modélisation hiérarchique et récursive appelée quadtree.

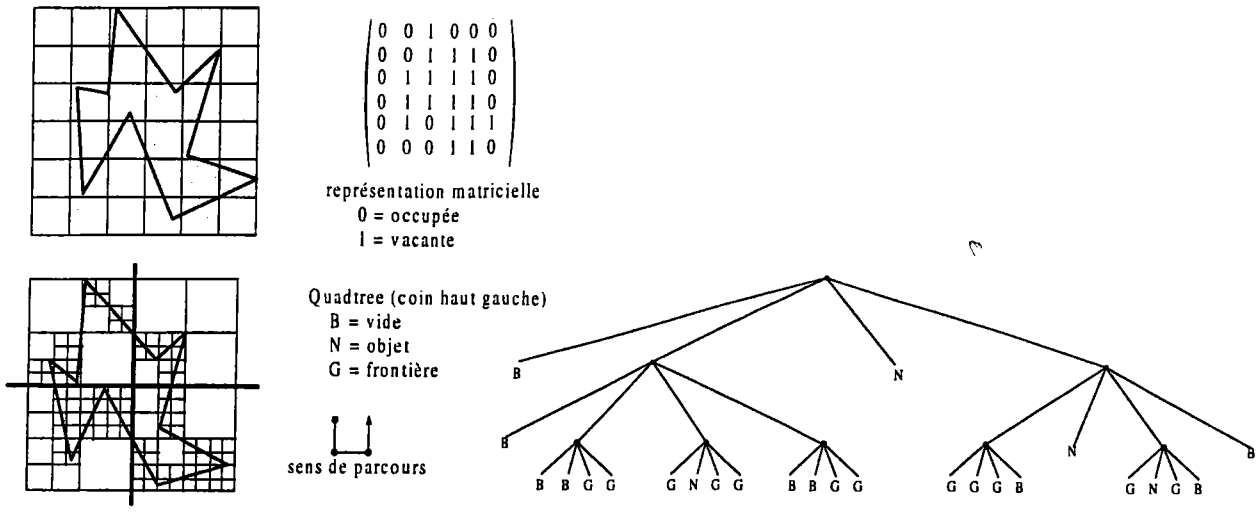


Figure 3.9. *Modèles de mémorisation*

Dans la discrétisation à pas adaptatif, le découpage dépend de la forme modélisée et les cellules sont des rectangles de dimensions variées. La représentation par chaîne est une représentation possible de ce type de discrétisation. Chaque ligne est représentée par sa hauteur suivie entre parenthèses d'une liste de valeurs correspondantes au nombre de cellules de même type consécutives (figure 3.10).

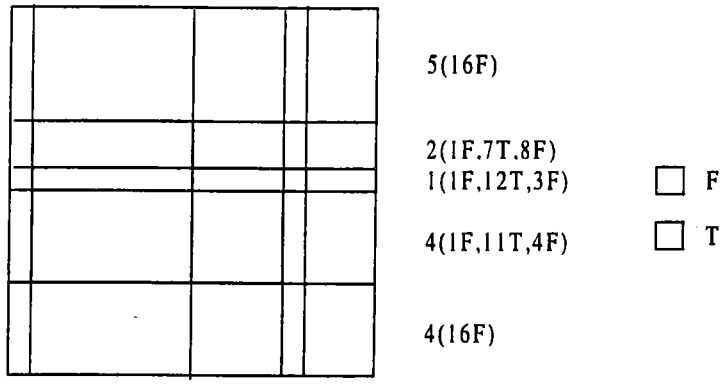


Figure 3.10. *Représentation par chaîne*

La modélisation par discrétisation à pas fixe ou variable permet de représenter à peu près toutes les formes possibles en adaptant le pas de la discrétisation à la précision souhaitée. Les deux types de modélisations suivantes sont plus particulièrement dédiées à la représentation des relations topologiques entre des entités rectangulaires.

3.3.2. Modélisation par arbre de partitionnement

Un espace de placement peut être représenté par un ensemble de rectangles liés par différentes relations. Lorsque les relations correspondent à un certain partitionnement d'un espace initial, les informations peuvent être conservées par un arbre qui représente la stratégie de décomposition. Cet arbre est appelé arbre de partitionnement. Les feuilles de l'arbre représentent des objets alors

que les autres nœuds modélisent des groupes d'objets. Le type de chaque nœud précise le type de découpage utilisé qui peut être soit horizontal, soit vertical. La figure 3.11 donne un exemple d'espace de placement et de son arbre de partitionnement.

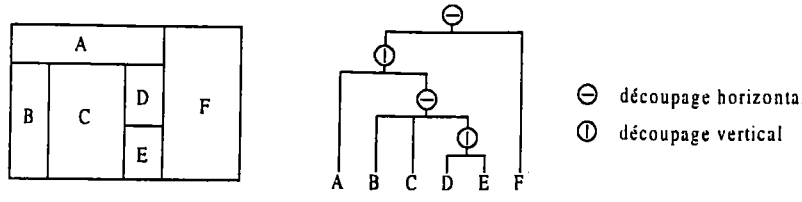


Figure 3.11. *Arbre de partitionnement*

3.3.3. Modélisation d'un placement topologique

La modélisation d'un placement topologique est une modélisation des relations d'adjacence entre les différents objets plutôt qu'une modélisation géométrique de l'espace de placement. Elle ne suffit pas à définir totalement l'espace mais elle peut constituer une source d'informations facilement exploitable notamment par l'utilisation des algorithmes sur les graphes. En effet, ce type de modélisation, couramment utilisé en électronique et en architecture, utilise comme structure de modélisation la structure de graphe. Dans le graphe d'adjacence (figure 3.12), les nœuds représentent les différentes zones de l'espace de placement et les arcs liant les nœuds matérialisent les deux types de relations : adjacence horizontale et adjacence verticale.

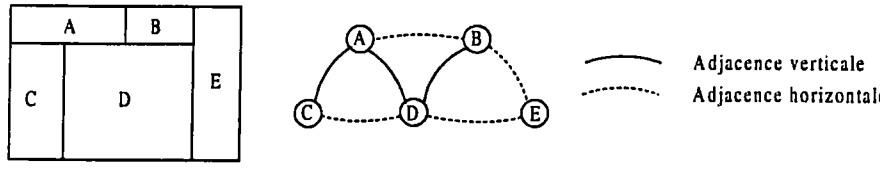


Figure 3.12. *Graphe d'adjacence*

3.3.4. Synthèse

Les représentations par énumération spatiale à pas fixe ne nous semble pas très bien adaptées à la modélisation des espaces de placements qui changent au cours du processus d'aménagement et qui sont constitués de zones denses en objets placés mais également vides. Ce type de modélisation à tendance à être soit trop détaillée ce qui implique une sur-occupation mémoire inutile soit trop imprécise et peut alors entraîner des erreurs.

Au contraire, l'énumération spatiale à pas adaptatif nous paraît tout à fait appropriée au type de problème que nous avons à gérer. La bonne représentation mémoire de ce type d'énumération est, pour nous, une structure incluant les différents avantages des méthodes proposées sans engendrer de redondances inutiles ou coûteuses.

4. MODÉLISATIONS MISE EN ŒUVRE DANS LE PROBLÈME D'AMÉNAGEMENT DE CUISINE

Dans cette partie, nous approfondissons la présentation des modèles de représentation des trois types de données manipulés dans notre application. Ces données considérées sont toujours les objets (meubles ou appareils électroménagers), l'espace de placement (cuisine) et les contraintes permettant, soit de préciser les caractéristiques propres d'une donnée, soit d'associer deux données qui peuvent être, deux objets ou encore un objet et l'espace de placement.

4.1. Modélisation détaillée des objets

Pour modéliser les objets dans notre système, nous avons choisi une représentation intermédiaire aux deux types de modèles présentés dans la partie précédente. La modélisation d'un objet consiste, dans un premier temps, à modéliser son enveloppe puis dans un second temps à modéliser ses caractéristiques.

4.1.1. Modélisation de l'enveloppe

L'enveloppe d'un objet peut être modélisée par une union d'éléments de base (rectangles, parallélogrammes, cercles, ... en 2D et parallélépipèdes, sphères, cylindres, ... en 3D). Plus la variété des objets de base est grande, plus le modèle de l'objet est fidèle à l'objet réel mais plus les traitements deviennent longs ; il s'agit de trouver un compromis en fonction de l'application étudiée de façon à obtenir une modélisation satisfaisante.

Les meubles d'une cuisine peuvent être modélisés de façon satisfaisante par des rectangles. En effet, la plupart des meubles ont approximativement cette forme et dans les autres cas, l'information topologique est plus pertinente que l'information dimensionnelle. On peut ainsi définir de manière formelle l'ensemble des objets à placer par Σ_o. Σ_o est composé de l'union des différents types d'éléments. Dans l'application considérée, on définit deux ensembles : O_1 l'ensemble des meubles et O_2 l'ensemble des appareils électroménagers. Ainsi Σ_o est égal à l'union de ces deux ensembles c'est-à-dire : $\Sigma_o = O_1 \cup O_2$.

Sur la figure 3.13, on constate que les meubles devant se trouver à une extrémité d'un ensemble ont souvent des coins coupés ou arrondis (esthétisme, commodité, sécurité) mais le

volume retiré au parallélépipède de base n'a aucune importance pour le placement, l'information importante est que ce meuble soit placé à une extrémité du plan de travail.

Certains effets de style sont souhaités dans les modèles de cuisine contemporaine comme le placement d'une table en bout de plan de travail (figure 3.14). À nouveau, on peut prendre en compte cette particularité sans modéliser la forme spéciale des deux éléments. En effet, on peut modéliser cette configuration, soit en utilisant deux rectangles R1 et R2 liés par une forte contrainte d'adjacence, soit en considérant un seul meuble composé d'une union de rectangles. Dans tous les cas, dans l'esprit du concepteur, l'important est que les deux meubles ne peuvent pas être séparés.

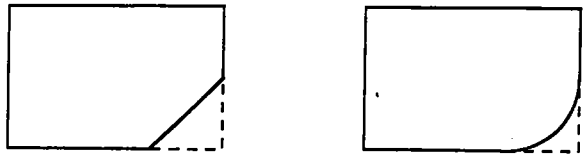


Figure 3.13. *Meubles d'angles*

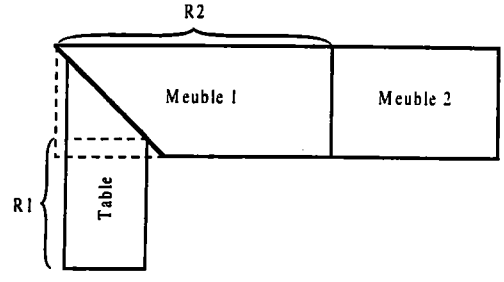


Figure 3.14. *Configuration spéciale*

En aménagement de cuisine, une tolérance sur certaines dimensions peut être envisagée. Par exemple, pour les zones de passage qui peuvent avoir des dimensions un peu changeantes. C'est le cas de la zone de passage autour d'une table ronde modélisée par un carré (figure 3.15). La zone de passage pourra être un peu plus petite ou un peu plus grande que la largeur préconisée.

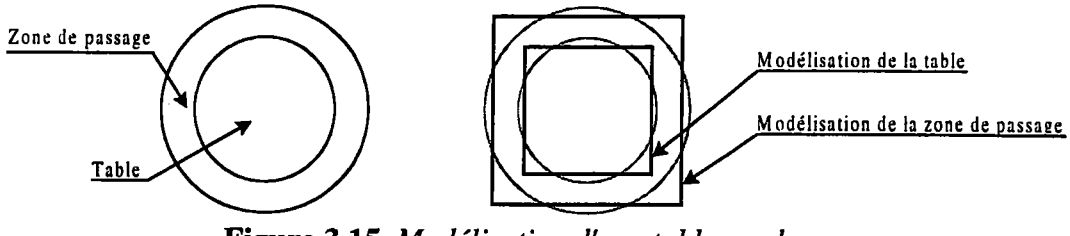


Figure 3.15. *Modélisation d'une table ronde.*

Chaque élément de l'ensemble Σ_o des objets à placer est représenté par un ou plusieurs rectangles en 2D. Ainsi, si on considère l'ensemble R des rectangles tel que :

R = { r / r = (x, y, l, L) où x, y, l, L appartiennent à l'ensemble des réels et
x = abscisse du point de référence du rectangle
y = ordonnée du point de référence du rectangle
l = largeur du rectangle
L = longueur du rectangle
}

il existe une application f_1 associant à chaque objet simple un rectangle de R et une application f_2 qui à tout objet complexe associe une union d'éléments de R.

On a donc :

$$f_1 : \Sigma_o \rightarrow R \qquad o \rightarrow r \qquad \text{et} \qquad f_2 : \Sigma_o \rightarrow R \times R \times \ldots \times R \qquad o \rightarrow r \cup r \cup \ldots \cup r$$

4.1.2. Modélisation des caractéristiques

Toutes les entités modélisées disposent d'un certain nombre de caractéristiques qui définissent précisément l'objet. Elles peuvent être liées à la géométrie, à la topologie ou aux fonctionnalités de l'élément considéré.

Le calcul de l'encombrement de chaque élément est indispensable à la vérification du placement des objets les uns par rapport aux autres. Il est donc nécessaire de modéliser les différents types d'ouvertures possibles de portes ou de tiroirs. Une ouverture de porte de meuble est représentée par un quart de disque (de rayon la longueur de la face avant du meuble) et l'ouverture d'un tiroir par un rectangle (de longueur identique à celle du meuble et de largeur égale à la profondeur du meuble ou du tiroir). La figure 3.16 résume les différents types d'ouverture possibles.

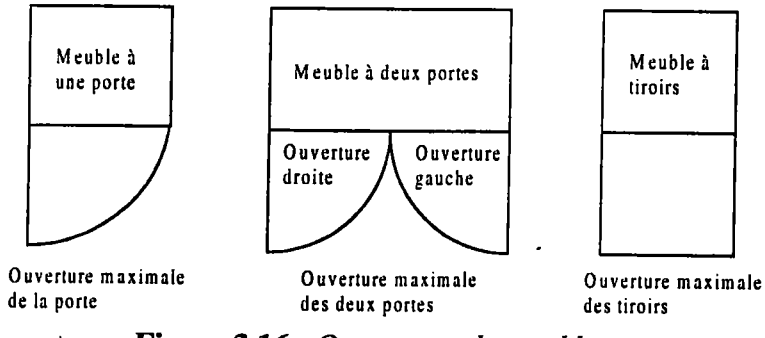


Figure 3.16 . *Ouvertures de meuble*

Distinguer deux types d'ouvertures, l'ouverture droite et l'ouverture gauche intervient peu dans le placement des meubles mais a une grande importance fonctionnelle dans la configuration finale de la cuisine. Par exemple, si au-dessus et à gauche de la plaque de cuisson se trouve le meuble à épices, il est bien plus astucieux de lui imposer une ouverture gauche qu'une ouverture droite. On peut également tolérer une ouverture qui ne soit pas maximale dans certains cas exceptionnels ceci seulement si la différence entre l'ouverture effective et l'ouverture maximale n'est pas trop importante.

Un certain nombre d'autres informations pertinentes doivent être conservées. Pour un meuble, généralement rectangulaire, il est important de différencier la face arrière de la face latérale car la longueur de la face arrière définira l'encombrement de l'élément sur le mur (figure 3.17).

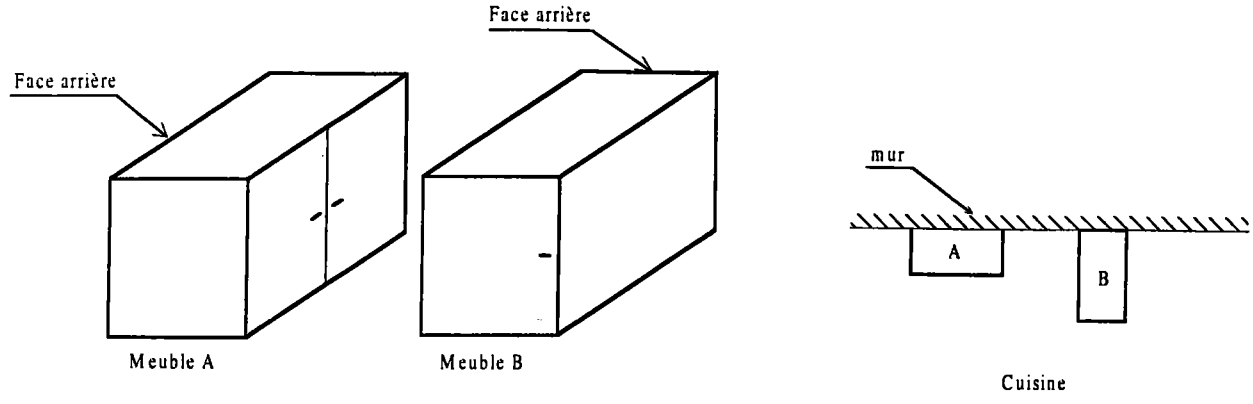


Figure 3.17. *Encombrement d'un meuble*

Les meubles sont destinés à certains placements (figure 3.18), un même meuble ne pourra pas être placé indifféremment :

- au mur (meuble haut) ou au sol (meuble bas) ;
- en bout de table de travail (meuble d'extrémité), dans un coin de la pièce (meuble d'angle) ou entre deux autres meubles (meuble intermédiaire).

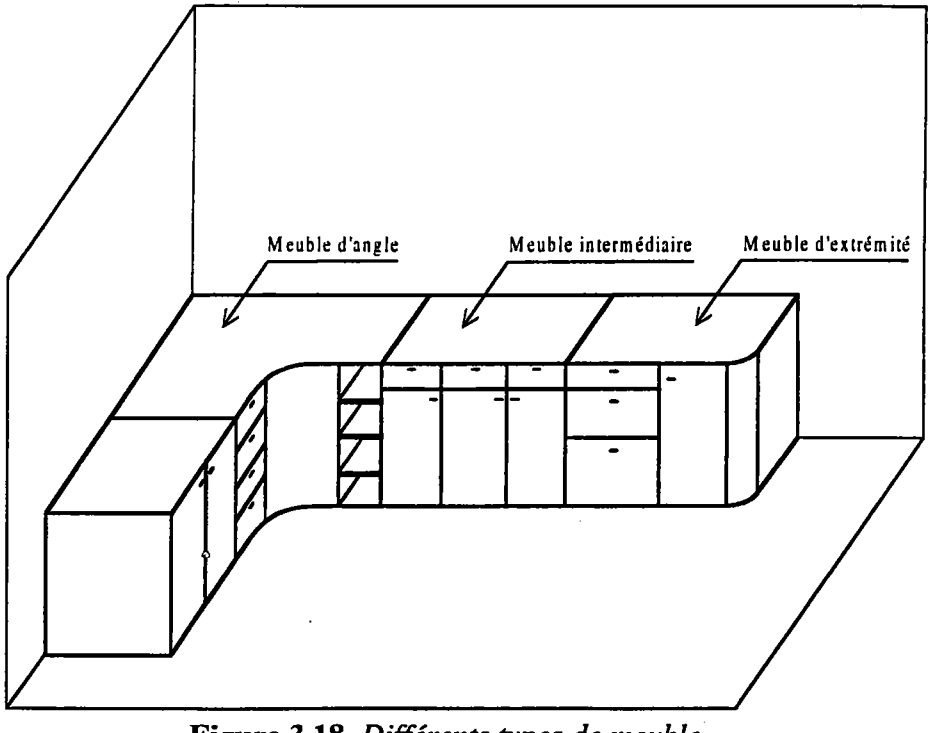


Figure 3.18. *Différents types de meuble*

Pour un appareil électroménager, il est intéressant de faire un rapprochement entre les besoins de l'appareil et les possibilités de l'environnement. Ainsi, il est judicieux de placer un appareil :

- évacuant de l'eau près d'une évacuation d'eau ;
- utilisant de l'eau près d'une arrivée d'eau ;
- à gaz près d'une arrivée de gaz ;
- électrique près d'une prise ;
- ...

Il est important de connaître les caractéristiques de ces appareils car on ne pourra pas mettre un appareil utilisant de l'eau dans une pièce qui ne dispose pas d'une arrivée d'eau et de même pour les autres types d'alimentation ou d'évacuation. Par contre, d'après les spécialistes en aménagement de cuisine, la proximité d'une alimentation ou d'une évacuation n'est pas un critère de décision fondamentale car des déplacements peuvent être réalisés. Cependant si des travaux supplémentaires peuvent être évités sans nuire à la fonctionnalité de la cuisine finale ou sans empêcher la vérification d'autres contraintes, la position des différentes alimentations et évacuations sera prise en compte.

4.2. Modélisation détaillée de l'espace

La modélisation de l'espace de placement peut également être réalisée suivant les deux mêmes phases. La décomposition de l'espace en une union de rectangles (éventuellement petits dans les endroits délicats) peut constituer une modélisation de l'enveloppe. En choisissant cette représentation, il existe inévitablement des zones qui ne peuvent pas être modélisées (arrondies, triangulaires, ...) ce qui n'est pas gênant car il est peu probable qu'elles correspondent à un placement correct.

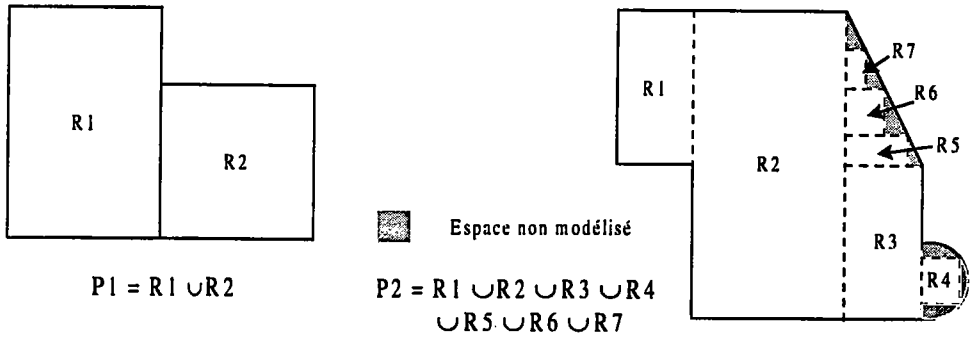


Figure 3.19. *Modélisation de l'espace*

Parmi ces rectangles modélisant l'espace de placement, on distingue trois types de zones :

- les zones libres qui peuvent être utilisées pour placer de nouveaux meubles ;

- les zones occupées qui contiennent un meuble et qui ne peuvent plus être utilisées ;
- les zones d'accès qui peuvent éventuellement être réutilisées pour l'ouverture d'autres meubles. Le chevauchement de zones d'accès est donc autorisé. Par contre tous les autres types de chevauchement (accès-occupée / occupée-occupée) sont strictement interdits.

La figure 3.20 montre les différents types de zones pour un aménagement de cuisine.

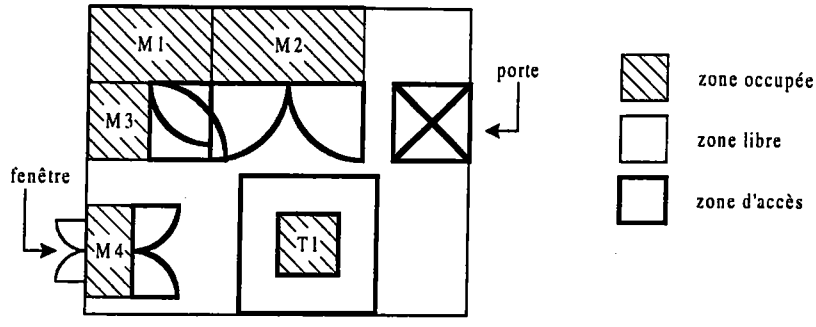


Figure 3.20. *Exemple d'aménagement de cuisine*

Comme pour l'ensemble des objets, on peut formaliser la définition de l'espace de placement. On définit ainsi un ensemble Σ_e représentant l'espace de placement qui est constitué de l'union des différents types de zones distinguées. Dans notre cas particulier, trois types de zones sont différenciés auxquels sont associés trois ensembles :

E_1 = L'ensemble des espaces correspondant à des zones occupées

E_2 = L'ensemble des espaces correspondant à des zones libres

E_3 = L'ensemble des espaces correspondant à des zones d'accès

de sorte que $\Sigma_e = E_1 \cup E_2 \cup E_3$.

Une application g mettant en relation chaque sous-espace de l'espace de placement avec un ou plusieurs rectangles en 2D peut être définie de la façon suivante :

$$g: \quad \Sigma_e \rightarrow R \times R \times \ldots \times R$$
$$e \rightarrow r \cup r \cup \ldots \cup r$$

La modélisation des ouvertures de l'espace de placement, notamment celle des portes, va impliquer la définition de zones d'accès correspondant soit à l'ouverture de la porte si on dispose d'une porte à ouverture vers l'intérieur de la pièce, soit à une zone d'accessibilité si la porte est coulissante ou à ouverture vers l'extérieur de la pièce. Il est nécessaire également de modéliser la fenêtre et surtout la hauteur sous-fenêtre de façon à pouvoir placer éventuellement un meuble en dessous d'une fenêtre. On pourra modéliser comme caractéristiques supplémentaires d'un espace de placement, les emplacements des différentes évacuations, alimentations, prises même si d'après les experts du domaine, ce ne sont pas des informations primordiales car elles peuvent être déplacées plus ou moins facilement.

4.3. Contraintes à respecter

La modélisation des objets d'une part et celle de l'espace d'autre part vont fournir des informations nécessaires au placement des objets dans l'espace considéré. Pour que ce placement soit fonctionnel, un certain nombre de contraintes vont être définies et il sera impératif que le placement final, obtenu grâce au système défini, en respecte un maximum. Ces contraintes (indispensables ou facultatives) peuvent être imposées par différentes personnes, ce qui constitue différents niveaux de prise en compte.

Les contraintes du ***niveau logique*** sont des "évidences" pour la plupart, elles sont propres à tout aménagement et souvent obligatoires. Elles constituent un premier ensemble de règles à respecter qui peuvent être :

- ne pas gêner l'ouverture d'une porte ou d'un tiroir ;
- mettre un meuble devant une fenêtre seulement si sa hauteur est inférieure à la hauteur sous la fenêtre ;
- mettre la face arrière (le fond) des meubles contre un mur.

Le ***niveau fabricant*** est constitué essentiellement de consignes de sécurité et d'utilisation des appareils données par le manuel d'utilisation. Elles peuvent être impératives lorsqu'elles sont liées à la sécurité (ne rien poser directement sur un appareil de chauffage) et conseillées si elles concernent une meilleure utilisation des appareils (éviter de placer un appareil chauffant à côté d'un appareil réfrigérant).

Au ***niveau concepteur***, on trouve des contraintes facultatives, relatives à l'aménagement de l'espace et fournies par un professionnel du domaine. Elles ne sont pas obligatoirement toutes respectées (impossible la plupart du temps) mais plus la cuisine créée respecte de contraintes de ce type, plus elle est fonctionnelle. Ces contraintes fonctionnelles définies par l'expert sont issues de l'expérience et ont des priorités différentes :

- les "incontournables"
  - l'évier devant une fenêtre ou à défaut devant une forte source lumineuse ;
  - le lave-vaisselle à côté de l'évier ;
  - le radiateur sur un mur extérieur ou/et sous une fenêtre.

- les "pratiques"
  - la règle du triangle d'activité : même nombre de pas entre l'évier, le réfrigérateur et les plaques de cuisson ;
  - un minimum de trois tiroirs.

- les "agréables" : celles qui permettent une adaptation au client
  - tenir compte pour l'ouverture des portes du fait que l'utilisateur de la cuisine soit gaucher ou droitier ;
  - adapter la hauteur de la table de travail, des meubles, ... hauts à la taille du client;
  - satisfaire au mieux toutes les contraintes en respectant un certain budget.

Les contraintes du ***niveau utilisateur*** correspondent à des préférences du client, des choix et des envies personnels. Elles sont prioritaires par rapport à celles de l'installateur.

- contraintes de proximité entre certains meubles ou appareils ;
- types de meubles ou d'appareils utilisés ;
- contraintes de placement de certains meubles ou appareils.

A priori les contraintes du niveau concepteur comme celles du niveau utilisateur recherche pour la cuisine conçue une fonctionnalité maximale. Cependant, le concepteur qui travaille étroitement avec l'installateur, a tendance à favoriser en priorité les notions de faisabilité et de facilité de mise en œuvre des contraintes considérées pour la réalisation d'un aménagement donné.

5. RÉSOLUTION DU PROBLÈME D'AMÉNAGEMENT SPATIAL

5.1. Méthode algorithmique

Dans cette partie, nous détaillons les méthodes mises en œuvre pour trouver le meilleur placement d'un objet dans un espace donné. Nous distinguons deux types d'objets, les objets simples pouvant être modélisés par un rectangle et les objets complexes pour lesquels la modélisation à l'aide d'un unique rectangle est trop imprécise. Dans ce cas, l'utilisation d'une union de rectangles est nécessaire pour obtenir une modélisation satisfaisante.

5.1.1. Présentation générale

Le positionnement d'un élément dans un espace de placement est présenté en utilisant les modélisations données de ces deux entités, dans la partie précédente. Les contraintes énoncées antérieurement permettent de définir un placement optimal de tout élément dans un espace libre.

Tout élément est défini par un point de référence et deux directions et longueurs d'accroissement, tout positionnement sera donc également représenté par les mêmes informations (figure 3.21).

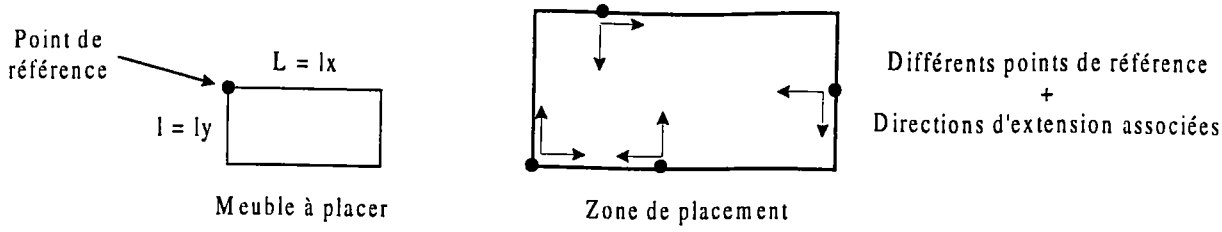


Figure 3.21. *Définition d'un placement*

La résolution du problème de positionnement contraint d'objets consiste donc, dans un premier temps, à déterminer certains points de référence possibles. Puis dans un deuxième temps, les extensions nécessaires à partir du point de référence, en longueur et en largeur, sont testées. Finalement, dans un troisième temps, les différents placements valides sont évalués et comparés de manière à retenir le positionnement optimal.

Les différents points de référence envisageables sont obtenus après vérification d'un ensemble de règles prédéfinies. Ces règles peuvent porter sur les caractéristiques de la cuisine ainsi que sur celles du meuble à placer mais également sur la configuration courante c'est-à-dire sur l'aménagement intermédiaire déjà obtenu. En effet, les meubles sont placés contre les murs, de préférence à côté de meubles déjà placés, l'évier sous une fenêtre, ...

Pour chaque point de référence, l'algorithme de positionnement vérifie s'il existe une direction d'extension en longueur et une direction en largeur permettant d'obtenir une zone libre suffisamment grande pour contenir le meuble à placer. L'espace de placement étant constitué d'une union de rectangles, l'extension à partir du point de référence se fait en recherchant des rectangles adjacents à celui contenant le point de référence s'il n'est pas initialement assez grand. Les rectangles nécessaires au positionnement du meuble sont découpés de manière à ce que leur union soit exactement aux dimensions du meuble à placer.

Une adaptation du concept de graphe d'adjacence présenté précédemment est utilisée comme amélioration de la méthode initiale. En effet, nous montrerons dans un paragraphe suivant que la création au fur et à mesure du processus de placement du graphe d'adjacence de l'espace considéré permet sans être trop coûteuse de retrouver rapidement des informations importantes. Aucun chevauchement n'est évidemment autorisé lors du placement d'un meuble ; par contre, les zones d'accès de chaque meuble peuvent éventuellement se superposer. Ces zones sont indispensables pour chaque meuble ou appareil et correspondent aux ouvertures des portes ou des tiroirs du meuble considéré.

À chaque placement valide correspond donc une nouvelle représentation de l'espace de placement contenant de nouveaux espaces occupés ou d'accès ainsi qu'un nouveau découpage des zones libres. Cette nouvelle configuration est rendue effective lorsqu'un placement est validé. Certaines contraintes facultatives non considérées jusqu'à présent peuvent intervenir et permettre de différencier les différentes situations satisfaisantes de façon à retenir la meilleure. La figure 3.22 représente les différentes étapes de résolution du problème d'aménagement spatial d'objets contraints que nous proposons et appliquons à l'aménagement des cuisines.

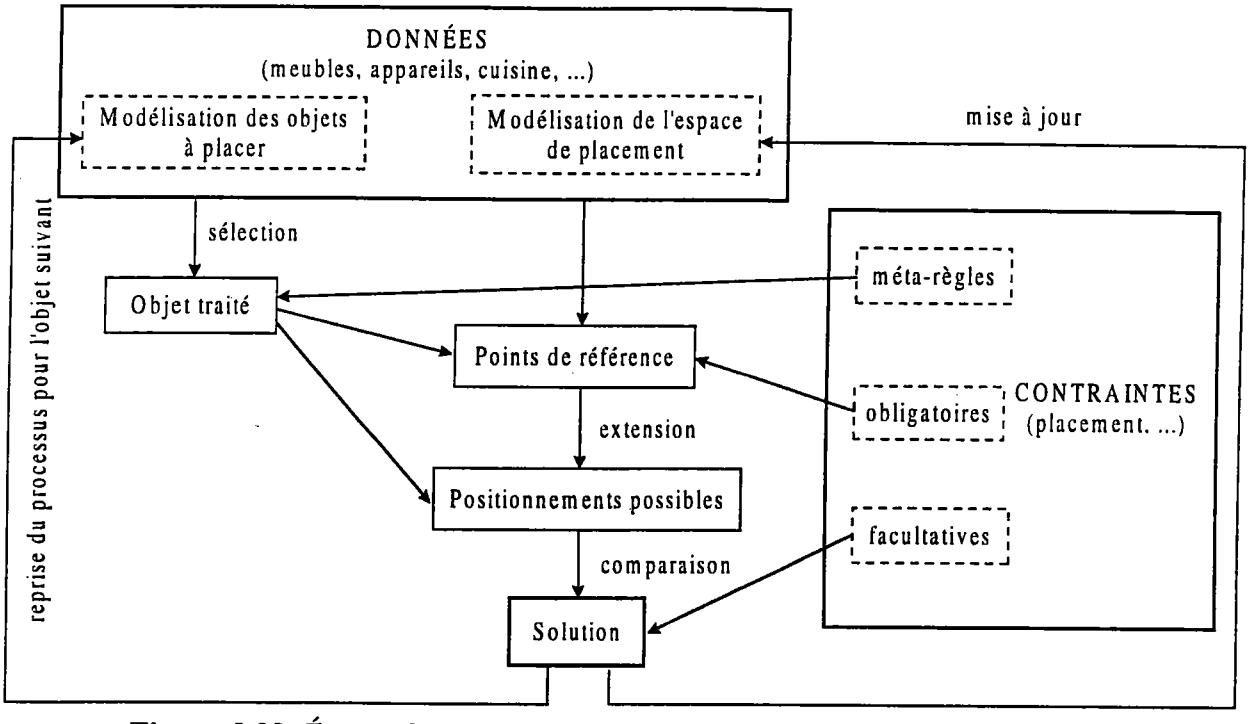


Figure 3.22. *Étapes de résolution du problème d'aménagement spatial*

On peut remarquer sur cette figure un type de contraintes non encore mentionné, il s'agit des méta-règles ou méta-contraintes. Elles interviennent dans une phase de prétraitements et permettent de définir un ordre de traitement des différents objets à placer. La prise en compte de ces règles en amont du processus de placement effectif, permet d'éviter un certain nombre de retours arrières c'est-à-dire d'essais de tentatives inutiles car infructueuses.

En effet, une étude préalable de l'ensemble des objets contraints à positionner peut permettre de détecter certains éléments prioritaires ; par exemple, si plusieurs objets sont placés en fonction d'un seul objet, ce dernier doit être placé assez tôt. De plus, on réserve les coins aux meubles d'angles, l'espace sous la fenêtre à l'évier, ... Si plusieurs objets sont fortement liés et doivent obligatoirement être positionnés les uns à côté des autres, on pourra considérer le regroupement de ces meubles ou appareils comme un seul et unique élément. Ces différentes considérations

conduisent généralement à traiter les entités les plus contraintes en priorité de façon à échouer le plus tôt possible et à réaliser ainsi le moins de retours arrières possibles.

L'optimalité de la méthode dépend essentiellement de trois facteurs :
- l'ordre de traitement des objets ;
- le choix du point de référence ;
- la méthode d'extension.

On peut espérer de bons résultats car :
- des méta-règles permettent de considérer en priorité les cas difficiles, ce qui permet de remettre en cause un faible nombre de résultats déjà obtenus ;
- les points de référence ne sont pas choisis aléatoirement. La vérification de contraintes permet d'éliminer a priori, une grande partie des points de référence ne convenant pas. De plus, si un point n'a pas été éliminé par le pré-traitement mais que son extension a échouée, on évitera de traiter à la suite des points de référence voisins de ce dernier.
- nous avons tenté de choisir le meilleur compromis entre, une méthode systématique et une discrétisation trop importante de l'espace. La méthode proposée impose des découpages de l'espace qui ne semblent pas a priori indispensables mais qui permettent d'automatiser la méthode. Le nombre de ces découpages est limité au minimum compte tenu de la méthode envisagée.

Après cette présentation générale de la méthode de placement que nous proposons, dans la partie suivante, nous présentons d'un point de vue plus technique un algorithme dédié au placement d'un objet simple (qui peut être représenté par un unique rectangle), suivie d'une méthode pouvant améliorer les performances de l'algorithme précédent. Finalement nous étudions le placement d'un objet complexe (union de plusieurs rectangles) toujours dans un environnement contraint.

5.1.2. Placement d'un objet simple

L'étape du processus de placement qui consiste à déterminer le placement exact de l'objet et à définir les nouveaux espaces occupés, libres et d'accès correspondants demande une méthode de recherche particulière (figure 3.23). On dispose d'un rectangle libre initial souvent plus petit que celui représentant le meuble à placer et que l'on agrandit suivant les deux axes x et y jusqu'à obtenir une union de rectangles ayant exactement la dimension du meuble. Plus précisément, on étend le rectangle initial suivant l'axe des x en considérant les rectangles qui lui sont adjacents verticalement, en les coupant en hauteur aux mêmes dimensions que le rectangle initial jusqu'à obtenir la longueur recherchée. Si la largeur du rectangle initial est inférieure à celle désirée, on

cherche le rectangle adjacent horizontalement au rectangle initial et on itère le processus décrit précédemment. Tous les rectangles constituant une zone aux dimensions du meuble à placer passent du statut de "libres" à "occupés" et les rectangles appartenant à la zone d'accès prennent (s'ils étaient libres) ou conservent (s'ils l'étaient déjà) le statut de zone d'accès.

Cette méthode permet de trouver de manière récursive tous les rectangles nécessaires au placement d'un meuble, en ayant choisi préalablement, un rectangle initial et en ayant fait correspondre le coin haut gauche de ce dernier au coin haut gauche du meuble. Les rectangles peuvent être découpés (création de nouveaux rectangles) afin de délimiter exactement l'espace de placement : la zone hachurée en final correspond exactement aux dimensions de l'élément. Il faut remarquer que cette méthode recherche assez fréquemment les rectangles adjacents à un rectangle de référence. Cette recherche peut être relativement coûteuse car elle consiste à parcourir l'ensemble des rectangles existants et à vérifier s'il existe une coordonnée commune entre le rectangle de référence et le rectangle considéré.

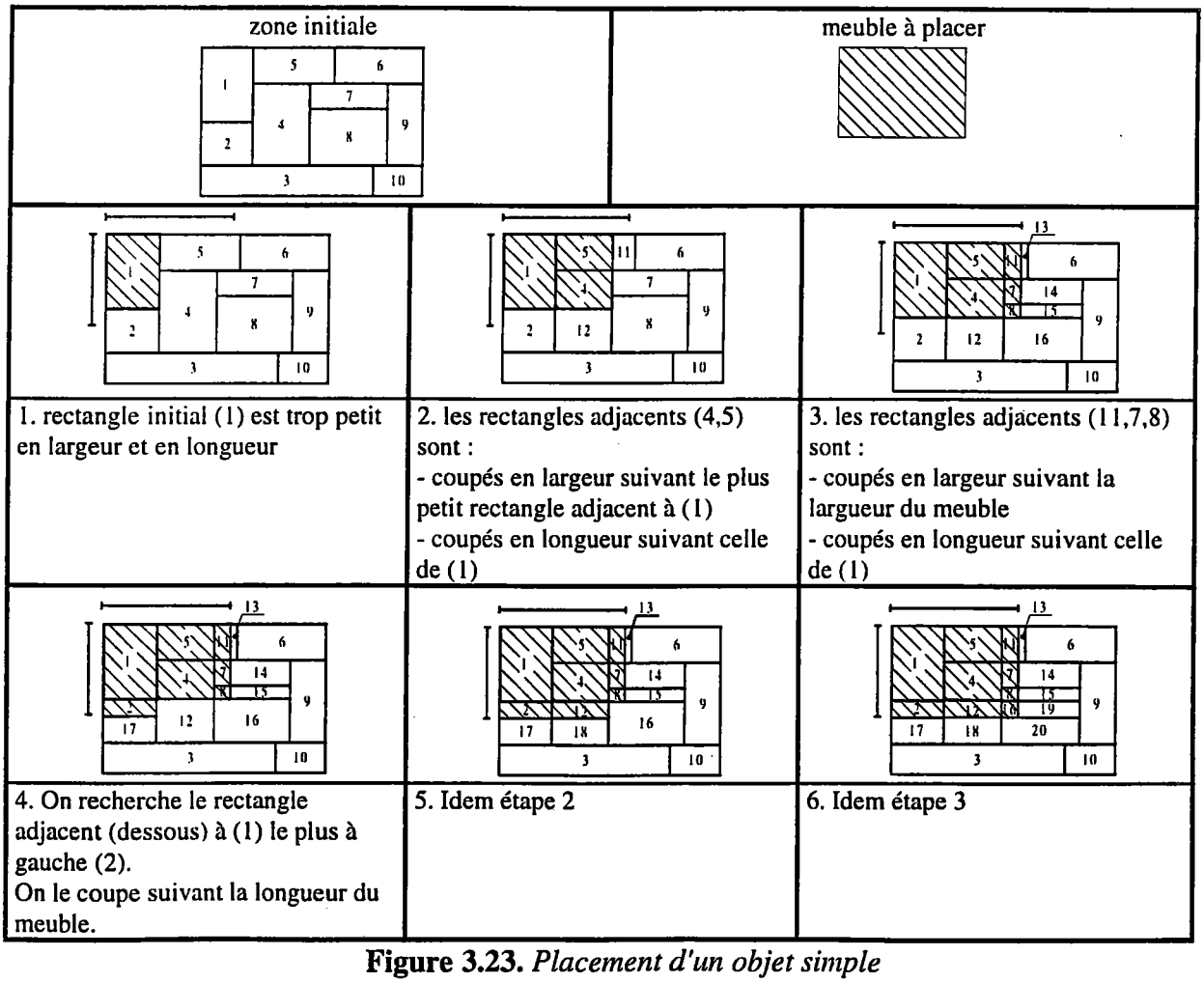


Figure 3.23. *Placement d'un objet simple*

Dans le paragraphe suivant, nous proposons d'éviter, dans certains cas, de rechercher parmi tous les rectangles possibles, les rectangles adjacents à un rectangle donné en conservant au cours de l'exécution du processus, les informations d'adjacence rencontrées.

5.1.3. Optimisation de la méthode de placement d'un objet simple

La méthode présentée au paragraphe précédent recherche régulièrement les rectangles adjacents (vers la droite et vers le bas) au rectangle considéré. Or, certains traitements nous permettent de connaître momentanément de telles contraintes. Nous proposons de conserver, dans un graphe, toutes les contraintes d'adjacence (droite et basse) dès qu'elles sont connues. Ce stockage évitera, dans certains cas, de parcourir l'ensemble de tous les rectangles existant pour retrouver des relations d'adjacence.

De manière formelle, le graphe d'adjacence G est défini par le triplet (N, A, E) tel que :

$N = \{n_i \in N / n_i$ est l'identifiant d'un rectangle$\}$ = Nœuds du graphe

$A = \{a_{ij} / a_{ij}$ est un arc liant les nœuds i et j$\}$ = Arcs du graphe

$E = \{e \in \{D,B\} /$ D représente une contrainte d'adjacence droite et

B représente une contrainte d'adjacence basse$\}$ = Étiquettes des arcs du graphe

Il existe ainsi une application bijective h transformant l'espace de placement E élément de l'ensemble des espaces $\mathcal{E}$ en un graphe d'adjacence G élément de l'ensemble des graphes Γ.

$$h : \mathcal{E} \rightarrow \Gamma$$
$$E \rightarrow G$$

Deux traitements mettent en évidence des relations d'adjacence et sont :

- la recherche des rectangles adjacents à un rectangle considéré. Ceci est nécessaire lorsque le rectangle initial est trop petit en largeur ; exemple figure 3.23 étape 1 : le rectangle 1 n'est pas assez large, le programme recherche alors ses adjacents vers la droite et trouve les deux rectangles 5 et 4. Ces deux relations d'adjacence (1 et 5 ainsi que 1 et 4) sont stockées dans le graphe d'adjacence (figure 3.24).

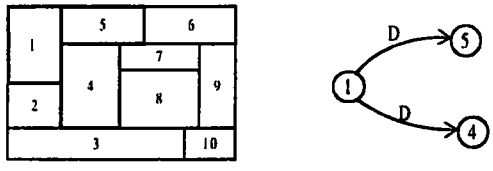


Figure 3.24. *Représentation dans un graphe des rectangles adjacents*

- lorsque l'on découpe un rectangle trop grand on peut déduire soit une contrainte d'adjacence droite, soit une contrainte d'adjacence basse suivant le type de découpage (figure 3.25).

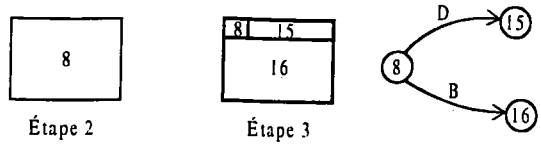


Figure 3.25. *Déduction d'adjacence*

Pendant l'exécution de l'algorithme, certaines informations pourront se trouver dans le graphe et d'autres devront être recherchées dans le modèle. La figure 3.26 correspond à l'exécution du placement d'un objet simple (exemple de la figure 3.23) ; le numéro entre parenthèses indique le numéro de l'étape ayant :

- soit demandé la recherche d'adjacence ;
- soit imposé le découpage, les relations engendrées sont stockées ;
- soit permis la déduction.

On peut constater sur l'exemple ci-dessous que les recherches d'adjacences dans le modèle se font surtout en début de traitement, plus la résolution est avancée plus les relations utilisées sont soit déduites soit stockées. Intuitivement, on peut conclure que si l'espace est découpé relativement finement par rapport à la taille de l'objet à placer, la création du graphe d'adjacence augmente les performances de l'algorithme de placement.

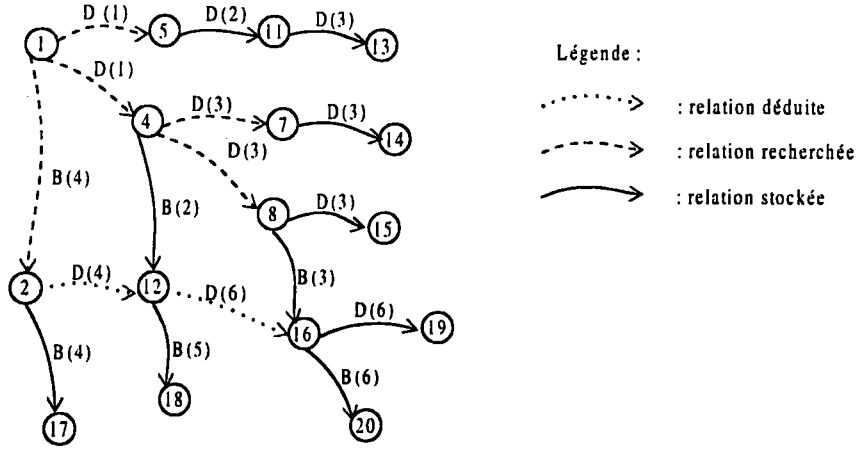


Figure 3.26. *Graphe d'adjacence*

Ce stockage partiel d'informations a différents avantages, il permet de :

- ne pas rechercher plusieurs fois les mêmes informations ;
- conserver uniquement des informations susceptibles d'être réutilisées par la suite, tout stockage est donc potentiellement utile ;
- déduire des informations à l'aide de règles triviales (figure 3.27) qui accélèrent la recherche. En effet, elles évitent le parcours complet du modèle représentant l'espace de placement pour retrouver une contrainte d'adjacence.

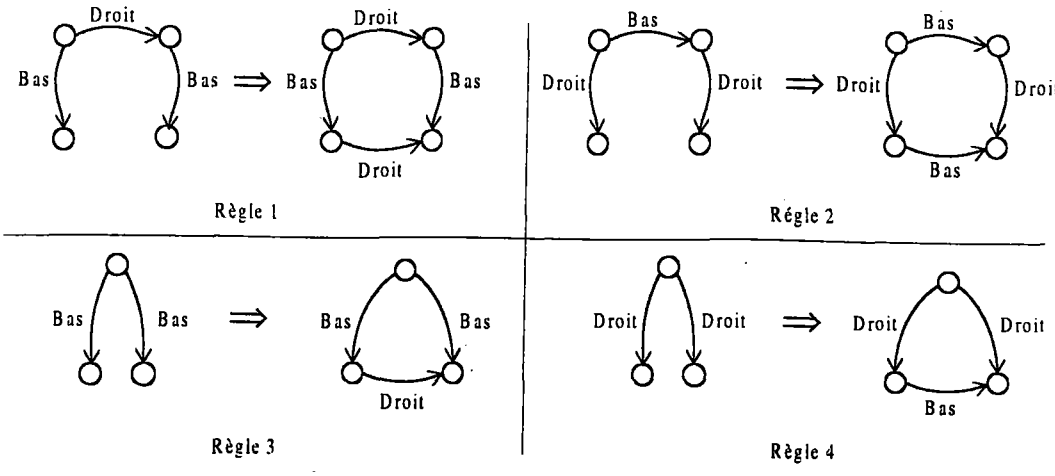


Figure 3.27. *Règles de déduction d'adjacence*

De plus ce stockage permet d'éviter un grand nombre de tests et le parcours du graphe construit n'est que local. Ce graphe augmente au fur et à mesure mais seules les nouvelles informations sont consultées.

Ces règles sont les pendants avec des rectangles du théorème qui énonce que lorsque deux droites sont parallèles, toute perpendiculaire à l'une est perpendiculaire à l'autre, sans les propriétés de commutativité et de transitivité. En effet, lorsque deux rectangles sont adjacences verticalement (resp. horizontalement), tous rectangles adjacents horizontalement (resp. verticalement) à ces rectangles sont susceptibles d'être adjacents verticalement (resp. horizontalement) entre eux (règle 1 et 3). De même, lorsque deux rectangles sont adjacents horizontalement (resp. verticalement), tous rectangles adjacents verticalement (resp. horizontalement) à ces rectangles sont susceptibles d'être adjacents horizontalement (resp. verticalement) entre eux (règle 2 et 4).

Ces déductions représentent donc des possibilités et non des certitudes compte-tenu de la non vérification des propriétés de transitivité et de commutativité. La figure 3.28 donne à partir d'un espace et d'une partie de informations connues, toutes les données qui peuvent être déduites en précisant si elles sont correctes ou non.

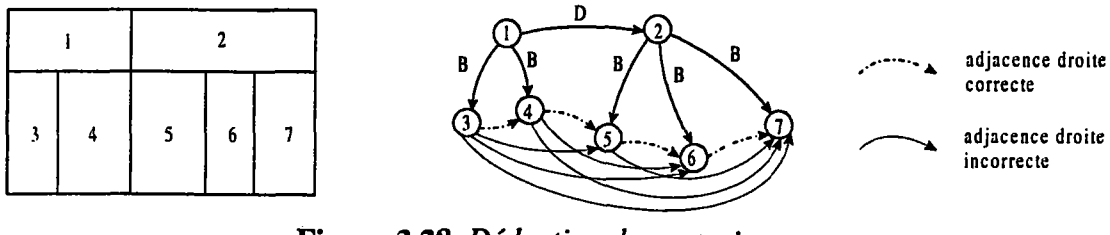


Figure 3.28. *Déduction de contraintes*

La validité des relations déduites est vérifiée en temps utile par une consultation ponctuelle du modèle. La pertinence du graphe d'adjacence n'est ainsi pas remise en cause. Le fait de stocker sur les arcs, les longueurs et largeurs des différents rectangles permet de tester en temps réel l'exactitude des déductions (figure 3.29).

Comme, à chaque étape, on découpe tous les rectangles impliqués dans la relation d'adjacence considérée suivant la plus petite largeur, on obtient des rectangles correspondant à des bandes verticales de même largeur. De même, horizontalement, on a des bandes de même largeur.

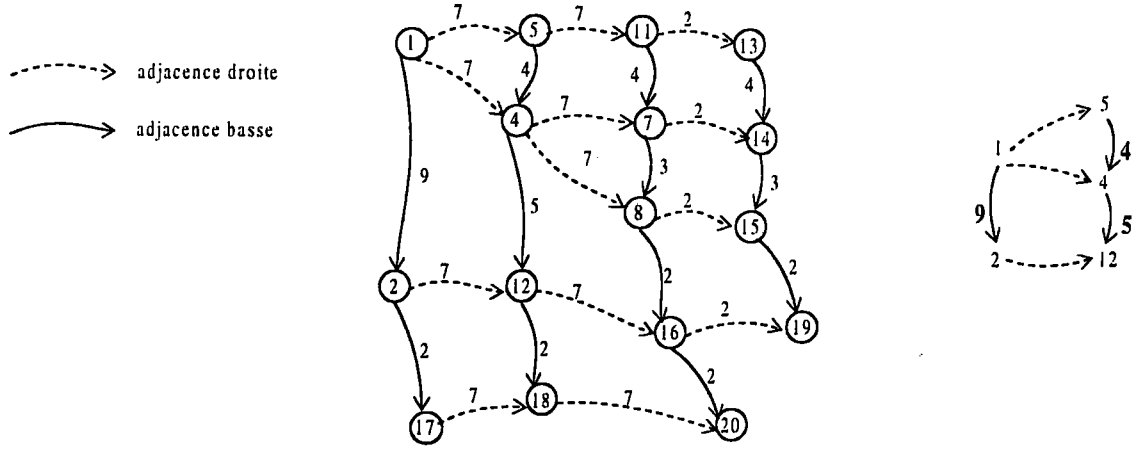
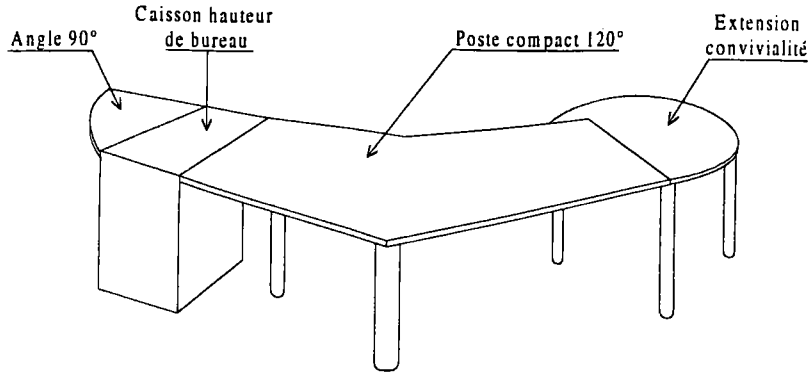


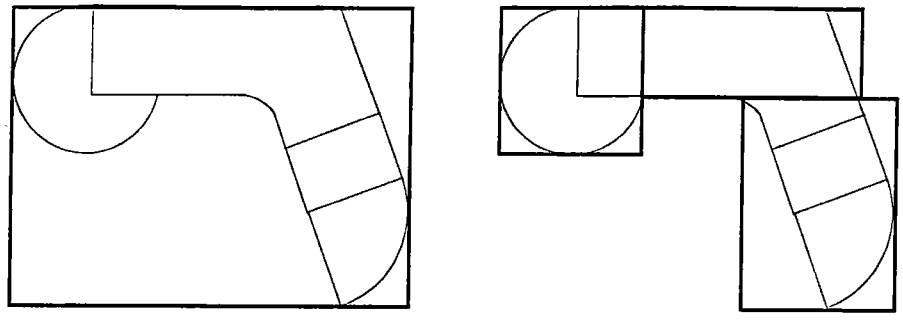
Figure 3.29. *Conserver les longueurs des rectangles*

5.1.4. Placement d'un objet complexe

La méthode définie précédemment permet le placement de la majorité des meubles ou appareils électroménagers d'une cuisine car dans la plupart des cas, ils peuvent être modélisés par un rectangle. Cependant, dans l'aménagement d'autres types d'espace de placement, l'aménagement spatial d'une usine ou d'un atelier, par exemple, le placement d'objets plus complexes (figure 3.30a) doit être réalisé. En effet, dans certains cas, la modélisation par un rectangle d'un objet n'est pas la mieux adaptée (figure 3.30b) car la différence entre la forme réelle de l'objet et son rectangle englobant est trop grande. Une modélisation possible de ces objets est alors obtenue par le découpage de l'objet initial en un ensemble de rectangles (figure 3.30c). L'union de rectangles pourra être utilisée également pour modéliser un regroupement de meubles simples mais indissociables. Les contraintes liant fortement ces meubles entre-eux pourront être définies par les spécifications des différents meubles ou appareils, ou encore, déduites par le système lors de l'application des règles correspondant aux différentes contraintes géométriques, topologiques et fonctionnelles liées à l'environnement.



(a) Objet complexe à modéliser



(b) Le rectangle englobant n'est pas la modélisation la mieux adaptée

(c) Modélisation de l'objet par une union de rectangles

Figure 3.30. *Modélisation d'un objet complexe par une union de rectangles*

Dans cette partie, nous allons donc nous intéresser au placement d'un objet complexe composé d'une union de rectangles. La méthode proposée est divisée en trois étapes successives :

étape 1 : modélisation de l'espace de placement ;

étape 2 : modélisation de l'objet complexe ;

étape 3 : réalisation du placement.

L'espace libre représenté dans le modèle par une union de rectangles dont on connaît les longueurs et largeurs (figure 3.31a) est traduit par un graphe d'adjacence défini de la même façon que celui du paragraphe précédent (figure 3.31b). Toutefois l'introduction d'un nœud supplémentaire, appelé F, est nécessaire de façon à représenter les longueurs et largeurs des rectangles se trouvant aux extrémités gauche ou basse de l'élément modélisé. Cette mise sous forme d'un graphe d'adjacence étiqueté par les longueurs et largeurs des rectangles constitue la première étape de la méthode. Le processus est identique pour la deuxième étape qui correspond à la création du même type de graphe mais cette fois représentant les caractéristiques des différentes parties de l'objet à placer (figure 3.31c).

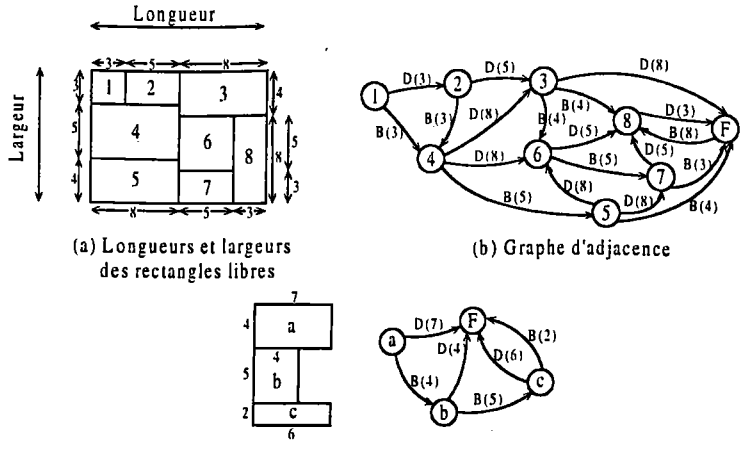
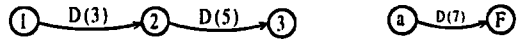


Figure 3.31. *Modélisation de l'espace de placement et de l'objet à placer*

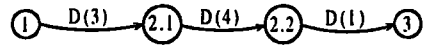
La troisième étape de la méthode réalise le placement effectif du meuble complexe ; la méthode réalise, en fait, le placement de tous les objets simples composant l'objet complexe. Certains rectangles de l'espace de placement peuvent comme précédemment être découpés de façon à correspondre exactement à l'espace nécessaire pour le placement de l'objet.

L'objectif de notre méthode est d'associer chaque nœud du graphe de l'objet à au moins un nœud (éventuellement plusieurs) du graphe de l'espace de placement, de façon à ce que la surface de l'union des rectangles sélectionnés corresponde à la surface du rectangle traité de l'objet à placer. Des nœuds supplémentaires correspondant à des découpages de rectangles pourront être créés de façon à couvrir exactement la largeur de l'objet.

Exemple : 1 est associé à *a* mais n'est pas suffisant ($D(3)_1<D(7)_a$). 2 est donc également associé à *a* mais dans ce cas, l'union des rectangles 1 et 2 est trop grande (en longueur) pour représenter le placement de *a* : $D(3)_1+D(5)_2 = D(8)>D(7)_a$.

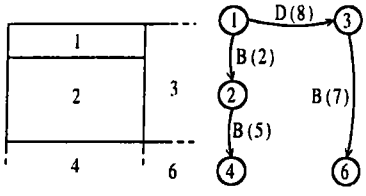


On découpe donc 2 en deux rectangles 2.1 de longueur 4 et 2.2 de longueur 1 et on associe alors 1 et 2.1 a *a* : $D(3)_1+D(4)_{2.1} = D(7)_a$. Le graphe est alors remis à jour :



On procède de même, en longueur et en largeur, pour le placement de tous les rectangles composant l'objet complexe. Les relations d'adjacence entre les anciens rectangles et le nouveau rectangle créé sont définies par quelques règles simples :

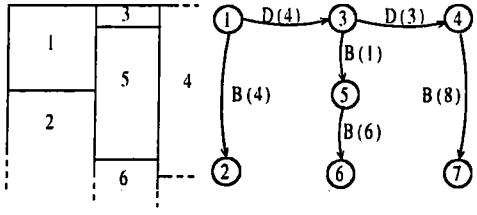
- un nouveau rectangle est au plus en relation avec exactement les mêmes rectangles que celui dont il est issu ;
- les règles de déduction d'adjacence de la figure 3.27 en tenant compte des distances, soit en les comparant directement, soit en les ajoutant :



On déduit les relations :

D(8) B(2)+B(5)=B(7)

D(8) B(2)<B(7)



On déduit les relations :

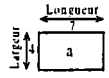
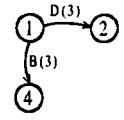
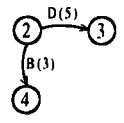
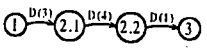
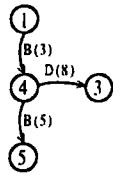
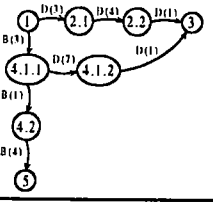
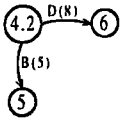
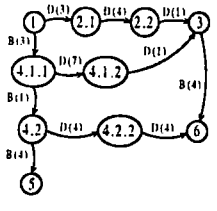
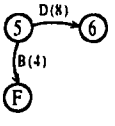
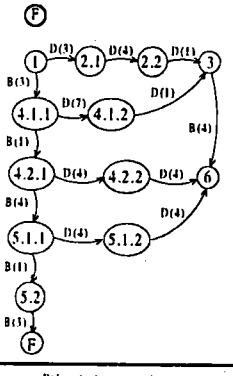
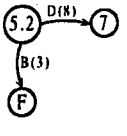
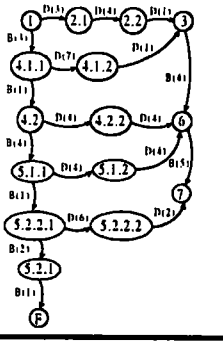
D(4) B(1)<B(4)

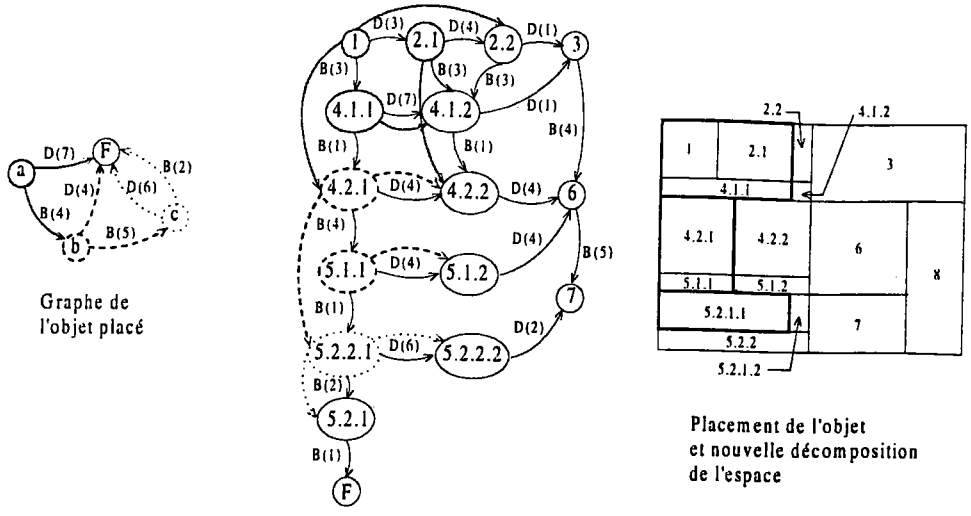
D(3) B(1)<B(8)

D(4) B(4)<B(1)+B(6)

D(3) B(1)+B(6)<B(8)

Nous allons présenter et justifier étape par étape, la réalisation du placement de l'objet multiple (c) dans l'espace de placement (a) de la figure 3.31.

Objet à placer	Explication	Informations consultées	Graphe auxiliaire créé	Associa -tion
	On choisit de commencer le placement à partir de 1. Largeur de 1 insuffisante (3<4) Longueur de 1 insuffisante (3<7)			a←1
	Extension vers la droite : 2 Largeur de 2 = Largeur de 1 Longueur trop grande (5+3>7) ⇒ découpage de 2 en 2.1 (4) et 2.2 (1)			a←1, 2.1
	Extension vers le bas : 4 Largeur trop grande (3+5>4) ⇒ découpage de 4 en 4.1 (1) et 4.2 (3) Longueur de 4.1 trop grande ⇒ découpage en 4.1.1 (7) et 4.1.2 (1)			a←1, 2.1, 4.1.1
	bas de a = haut de b or bas de a = haut de 4.2, on considère donc 4.2 pour le placement de b Longueur de 4.2 (8) trop grande ⇒ découpage de 4.2 en 4.2.1 (4) et 4.2.2 (4) Largeur de 4.2 (4) est trop petite pour b (5)			b←4.2.1
	Extension vers le bas : 5 Largeur trop grande (4+4>5) ⇒ découpage en 5.1 (1) et 5.2 (3) Longueur de 5.1 trop grande (8>4) ⇒ découpage en 5.1.1 (4) et 5.1.2 (4)			b←4.2.1, 5.1.1
	bas de b = haut de c or bas de b = haut de 5.2, on considère 5.2. Largeur trop grande (3>2) ⇒ découpage de 5.2 en 5.2.1 (2) et 5.2.2 (1) 5.2.1 trop long (8>6) ⇒ découpage en 5.2.1.1 (6) et 5.2.1.2 (2)			c← 5.2.2.1



Un même style de trait représente le lien entre chaque composant de l'objet complexe et son espace de placement dans respectivement le graphe de l'objet et le graphe de l'espace.

Figure 3.32. *Résultat du placement de l'objet complexe*

La figure 3.32 représente l'espace de placement ainsi que son graphe d'adjacence après le placement de l'objet complexe considéré. On peut ainsi vérifier que toutes les adjacences ainsi que toutes les longueurs et largeurs du problème sont respectées.

5.1.5. Conclusion

Dans cette partie, nous avons présenté une méthode algorithmique basée sur l'exploitation de graphes permettant de réaliser le positionnement d'un objet simple ou complexe dans un espace de placement défini. La méthode proposée traite l'étape centrale du problème de placement du objet qui se concentre uniquement sur la vérification des contraintes géométriques. En effet, nous avons précisé précédemment qu'il existait des étapes avant et après celle-ci permettant de prendre en compte d'autres types de contraintes comme les contraintes topologiques et fonctionnelles ou encore des contraintes géométriques de priorité différentes.

La méthode de base a fait l'objet d'une implémentation orientée objet en langage C++ qui permet d'obtenir le placement au sol d'un ensemble constitué d'une vingtaine d'objets contraints représentés par des rectangles de manière pratiquement instantanée. Des détails concernant cette implémentation sont fournis en annexe C de ce document. L'étude assez technique du placement d'un objet complexe réalisée au paragraphe précédent, n'est pas sans similitude avec les problèmes de recherche d'égalité de graphe réalisée au chapitre 2. C'est pourquoi, dans le paragraphe suivant, nous nous proposons d'étudier l'utilisation du formalisme CSP pour résoudre ce problème.

5.2. Application du concept de CSP au placement d'un objet complexe

Nous avons vu qu'un objet complexe est un objet représenté en 2D par une union de rectangles ; le placement d'un objet complexe consiste donc à positionner correctement cette union de rectangles dans un espace libre qui est lui-même modélisé par une union de rectangles. Ceci revient finalement à réaliser l'appariement de chaque rectangle modélisant un composant de l'objet à un ou plusieurs rectangles représentant l'espace de placement (figure 3.33).

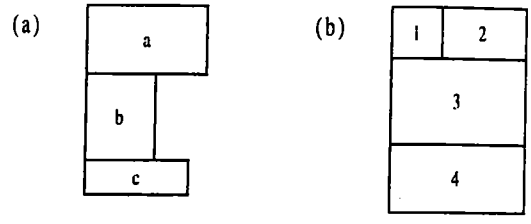


Figure 3.33. *(a) Objet complexe à placer - (b) Espace de placement*

Si le coin haut gauche de a est mis en relation avec le coin haut gauche de (1), les rectangles (1), (2) et (3) sont indispensables à la réalisation du placement de (a). Cependant, on peut constater que la surface représentée par l'union de ces trois rectangles est supérieure à la surface de (a). Il est donc nécessaire de réduire ces rectangles de manière à faire coïncider exactement les surfaces de (a) et de l'union des nouveaux rectangles obtenus après ajustement (figure 3.34).

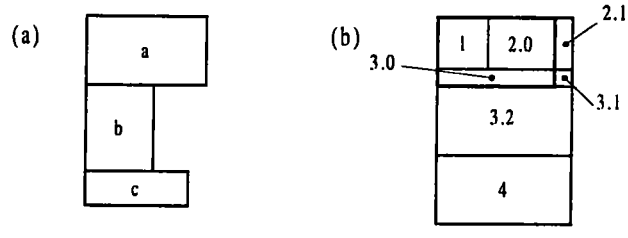


Figure 3.34. *Découpage nécessaire de l'espace de placement*

Le rectangle (2) est alors découpé en deux rectangles (2.0) et (2.1) et le rectangle (3) en trois rectangles (3.0), (3.1) et (3.2) de sorte que la surface de (a) soit égale à la surface de l'union des rectangles (1), (2.0) et (3.0) : (a) est donc apparié à ces trois rectangles.

Nous avons introduit au paragraphe précédent, la définition du concept de graphe d'adjacence permettant de modéliser à la fois les objets complexes et les espaces de placement. Nous rappelons que dans ce graphe, les nœuds représentent les composants de l'objet complexe ou de l'espace de placement modélisé en 2D à l'aide d'un rectangle et les arcs modélisent les relations d'adjacence entre les différents composants. Le placement de l'objet complexe consiste donc à apparier chaque nœud du graphe modélisant l'objet à un ou plusieurs nœuds du graphe représentant l'espace de placement.

Ce problème est similaire à celui de comparaison de pièces mécaniques que nous avons présenté au chapitre 2 de ce document et que nous avons résolu en modélisant la représentation sous forme de graphe des deux pièces à comparer à l'aide du formalisme CSP. Pour le problème qui nous intéresse, dans ce chapitre, le formalisme CSP est exploité en définissant de la façon suivante les données du problème :

- les variables sont les différents composants de l'objet complexe qui sont modélisés en 2D par des rectangles ;
- les contraintes sont les différentes relations d'adjacence liant les composants ;
- les domaines pour chaque variables sont les modélisations sous forme de rectangles des sous-espaces de l'espace de placement.

Le formalisme CSP est bien adapté aux problèmes dont les différentes données sont fixes c'est-à-dire lorsque les domaines de définitions des différentes variables ainsi que les contraintes les liant ne sont pas modifiés au cours du processus de placement. Cependant, nous avons pu constater que pour résoudre notre problème, il était nécessaire de modifier l'espace de placement ce qui implique une modification également des différentes contraintes d'adjacence liant les sous-espaces de l'espace de placement. Ces évolutions correspondent à des modifications des domaines des différentes variables et des étiquettes des arcs représentants les contraintes d'adjacence du problème.

Dans une première partie, nous montrons les adaptations nécessaires des données de notre problème de placement d'un objet complexe pour l'utilisation du formalisme CSP classique. Nous énonçons également les avantages et les inconvénients de cette méthode. Dans un second temps, nous proposons plutôt qu'une adaptation des données, une adaptation du formalisme en envisageant un concept de CSP dynamique. Nous évoquerons comme précédemment les qualités et défauts de cette méthode dans un cas général puis dans le cas de notre application.

5.2.1. Utilisation du formalisme CSP classique

Compte-tenu des observations déjà réalisées quant au problème de mise en correspondance exacte de tout composant de l'objet complexe avec un ou plusieurs rectangles représentant l'espace de placement, il est nécessaire de définir une unité de découpage des différentes représentations à l'aide de rectangles. Nous présentons ensuite, deux méthodes envisagées permettant de résoudre le problème considéré à l'aide du formalisme CSP classique.

5.2.1.1.Représentation des données

Une unité "raisonnable" de découpage est définie puis la décomposition de l'espace de placement et éventuellement de l'objet est réalisé suivant cette unité. On entend par unité "raisonnable" une unité permettant d'obtenir une mise en correspondance des composants de l'objet complexe et des représentants de l'espace de placement vérifiant une certaine tolérance définie préalablement. Cette unité ne doit ni être trop petite et entraîner ainsi une explosion combinatoire et une représentation à l'aide d'un graphe d'adjacence inutilisable de part sa taille et sa complexité, ni trop grande et engendrer en conséquent des solutions trop imprécises.

L'espace de placement peut être ainsi représenté par un ensemble de rectangles unitaires, on désignera chaque rectangle par son numéro de ligne et un numéro de colonne. La figure 3.35 montre l'application de cette stratégie de découpage à l'exemple considéré dans ce paragraphe.

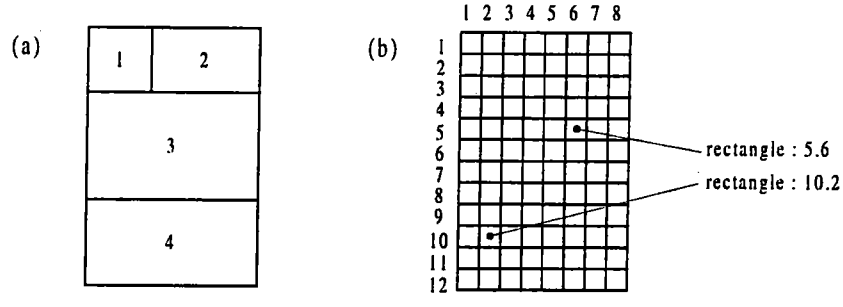


Figure 3.35. *(a) Représentation initiale - (b) Représentation unitaire*

L'objet complexe est représenté soit comme précédemment à l'aide d'un graphe d'adjacence (figure 3.36) soit de la même manière que l'espace de placement (figure 3.37).

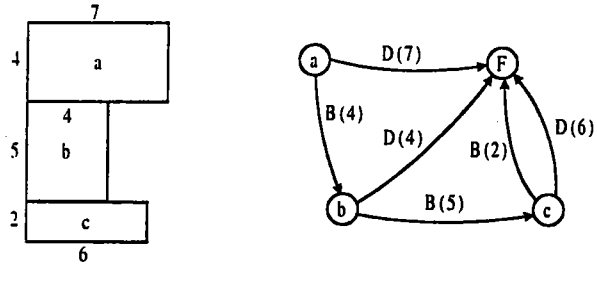


Figure 3.36. *Modélisation d'un objet complexe*

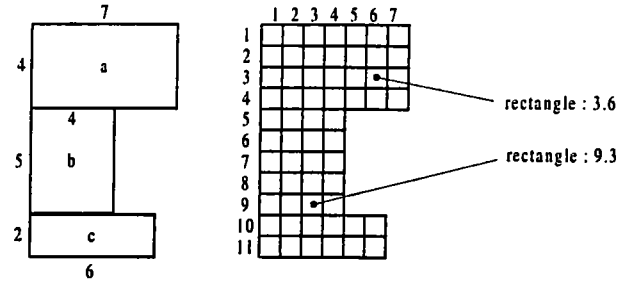


Figure 3.37. Représentation unitaire de l'objet complexe

Le graphe d'adjacence de l'objet complexe sert de support au graphe représentant le CSP correspondant au problème de placement d'un objet complexe dans un espace de placement défini. Le graphe représentant le CSP a donc les mêmes nœuds et les mêmes arcs que le graphe de l'objet. Le domaine initialement identique pour toutes les variables est constitué de tous les rectangles unitaires modélisant l'espace de placement et autorisé par les contraintes figurant sur

les arcs. Ces contraintes tiennent compte non seulement des adjacences mais également des longueurs et largeurs imposées.

Pour l'exemple considéré, on a par exemple sur l'arc liant les nœuds a et F les contraintes suivantes : (1.1,1.7) ; (1.1,1.8) ; (2.1,2.7) ; (2.1,2.8) ... (12.1,12.7) ; (12.2,12.8) correspondant à une adjacence droite de longueur 7. Tous les arcs du CSP sont étiquetés de cette manière, ainsi si a est associé à 1.1 alors alors lui associe également les rectangles 1.2, 1.3, 1.4, 1.5, 1.6 et 1.7 de manière à respecter une adjacence droite de longueur 7. Ce CSP ainsi construit peut être résolu en utilisant les différents algorithmes de résolution adaptés aux CSP notamment l'algorithme de Forward-Checking déjà utilisé au chapitre 2.

5.2.1.2.Définitions des CSP

Une première méthode utilisant seulement un découpage de l'espace de placement, permet de conserver toutes les informations utiles dans le graphe CSP, il n'est donc plus nécessaire de consulter le modèle de représentation des différentes entités après la création du graphe. Ce type de CSP est très contraint, en effet, même si le nombre de valeurs possibles pour chaque variable est initialement important (dans notre cas, nombre de variables=longueur×largeur = 8×12 = 96), il existe relativement peu de possibilités et après toute instanciation, les domaines de valeurs sont considérablement réduits. De plus, cette décomposition certes artificielle du problème, permet de rendre statique un problème a priori dynamique et d'exploiter ainsi le concept classique de CSP.

Cette méthode a toutefois deux inconvénients majeurs ; en effet, le choix de l'unité de découpage est relativement arbitraire alors qu'il détermine la complexité de la méthode et la création du CSP est assez coûteuse. À l'initialisation de nombreux tests sont réalisés de manière à définir les contraintes liant les différents éléments et ceci sur des domaines assez importants.

De manière à supprimer ces nombreux tests, nous présentons au paragraphe suivant, une méthode permettant d'automatiser la mise sous forme de graphe CSP du problème. En effet, cette méthode ne modélise pas en terme d'étiquettes les contraintes de longueurs et de largeurs à vérifier. Cette méthode est très proche de la méthode précédente, elle consiste à décomposer la représentation de l'objet complexe de la même manière que l'espace de placement (figure 3.37).

Dans cette méthode, l'ensemble des nœuds du CSP est constitué des rectangles unitaires représentant l'objet complexe. Les arcs liant les différents nœuds représentent les contraintes d'adjacence entre les différents rectangles unitaire. Contrairement à la méthode précédente, il

n'est pas nécessaire de prendre en compte les longueurs et largeurs des différents composants de l'objet complexe.

La résolution du problème consiste avec cette modélisation à prouver l'inclusion du graphe d'adjacence de l'objet complexe dans le graphe d'adjacence de l'espace de placement. Cette méthode a pratiquement les mêmes avantages et inconvénients que la méthode précédente. Cependant, le CSP peut être créé plus facilement car la méthode est systématique et ne nécessite donc pas la vérification d'un certain nombre de tests portant notamment sur la prise en compte des longueurs et largeurs des différents rectangles composants l'objet complexe. Par contre, la taille du CSP est beaucoup plus importante du fait de la décomposition en rectangles unitaires de l'objet complexe. Dans l'exemple considéré jusqu'à présent, le CSP de la première méthode était constitué de 4 nœuds alors que celui de la deuxième méthode en possède une soixantaine.

Les deux méthodes évoquées ci-dessus ont montré que le formalisme CSP classique n'est pas très bien adapté à un problème dont les données sont susceptibles de subir une évolution au cours du processus de résolution. De plus la transformation artificielle de ce problème dynamique en un problème statique complique considérablement les données du problème à modéliser. Ces méthodes ne permettent donc pas d'obtenir efficacement le placement exact d'un objet complexe dans un espace de placement. Par contre, elles peuvent fournir une esquisse de placement contenant certaines imprécisions (chevauchement, espaces entre meubles non souhaités, ...). Cette esquisse peut servir de donnée initiale à une méthode itérative de placement ; en effet, nous avons présenté au chapitre 1 plusieurs systèmes dont la stratégie consiste à améliorer de manière itérative une solution initiale.

Ces raisons nous amènent à proposer une méthode de résolution du problème en transformant les CSP classiques en CSP dynamiques.

5.2.2. CSP dynamique

Le CSP initial est créé de la façon suivante :

- les variables sont les différents constituants de l'objet complexe ;
- les domaines de définition des différentes variables sont constitués des sous-espaces de l'espace de placement ;
- les arcs représentent les différentes relations d'adjacence droites et basses liant les composants de l'objet ;

- les étiquettes sont les couples de numéro d'espace vérifiant soit une contrainte d'adjacence basse, soit une contrainte d'adjacence droite.

Pour l'exemple considéré, le CSP construit est celui de la figure 3.38.

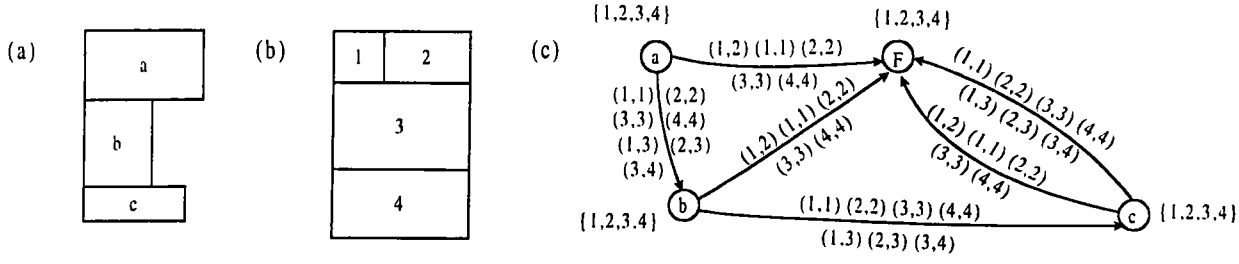


Figure 3.38. *(a) Objet à placer - (b) Espace de placement - (c) graphe CSP*

Ce CSP va évoluer au cours du processus de résolution en fonction des découpages nécessaires à la réalisation de la correspondance exacte entre chaque composant de l'objet complexe et un ou plusieurs rectangles de l'espace de placement (figure 3.34). Ces modifications doivent être répercutées non seulement au niveau des domaines de définition des variables mais aussi au niveau des étiquettes des différents arcs du CSP.

En considérant toujours l'exemple du placement de l'objet a (figure 3.34), le nouveau domaine de a est {1, 2.0, 3.0}, dans les domaines des autres variables 3 est remplacé par 3.1 et 3.2 et 2 par 2.1 et les ajouts et suppressions de contraintes suivants sont effectués :

- suppression :
  - adjacence basse : (2,2) (3,3) (1,3) (2,3) (3,4)
  - adjacence droite : (1,2) (2,2) (3,3)
- ajout :
  - adjacence basse : (2.1,2.1) (3.2,3.2) (3.1,3.1) (2.0,2.0) (1,3.0) (3.0, 3.2) (3.1, 3.2) (3.2, 4) (3.0,3.0) (2.1,3.1) (2.0,3.0)
  - adjacence droite : (1,2.0) (2.0,2.1) (2.1, 2.1) (3.2,3.2) (3.1,3.1) (2.0,2.0) (3.0,3.1) (3.0,3.0)

La mise à jour des contraintes peut être faite de deux manières :

- une méthode systématique ;
- une méthode avec vérifications.

La méthode systématique consiste à répercuter sur tous les rectangles issus d'un rectangle donné toutes les contraintes d'adjacence du rectangle initial. Cette méthode permet d'obtenir toutes les contraintes possibles parmi elles certaines n'existe pas. Il sera donc nécessaire de vérifier ultérieurement lors d'une utilisation éventuelle de la contrainte, sa validité à l'aide d'une

consultation des données du modèle. Le CSP ainsi représenté pourra être temporairement incorrect.

La mise à jour avec vérification n'autorise pas d'incorrection temporaire de la représentation ainsi toutes les contraintes déduites ne sont ajoutées que si elles sont validées par une consultation ponctuelle du modèle.

Certaines remarques peuvent être faites compte tenu de l'application considérée. En effet, dans notre application, seulement deux types de contraintes sont considérées, les adjacences basse et droite donc quel que soit le nombre d'arcs du CSP, il n'y a que deux listes de contraintes à mettre à jour. De plus, les retours arrières peuvent s'effectuer assez simplement si la numérotation des rectangles issus de découpages contient le numéro de l'étape ayant entraîné la modification. L'annulation d'une étape consistera ainsi à fusionner les rectangles ayant le même numéro d'étape (exemple figure 3.39). Annuler la troisième étape consiste à fusionner les trois rectangles 4.3.0, 4.3.1 et 4.3.2.

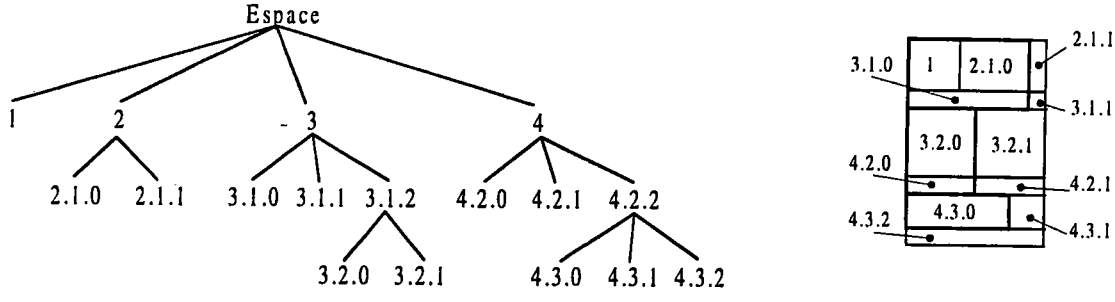


Figure 3.39. *Représentation arborescente de l'espace de placement*

Cette méthode permet de mettre en relation à l'aide d'une même structure toutes les informations du problème c'est-à-dire les objets, les espaces et les adjacences. Elle autorise également la modélisation et la résolution d'un problème dynamique à l'aide du formalisme CSP ce à quoi il n'est pas a priori destiné. Son inconvénient majeur pourrait être la mise à jour de toutes les contraintes à chaque étape mais il n'en est pas un réellement dans notre application car les contraintes à mettre à jour sont au nombre de deux.

5.2.3. Conclusion

Dans cette partie, nous avons envisagé la résolution du problème de placement d'un objet complexe en utilisant le formalisme CSP. De toute évidence, le concept classique de CSP n'est pas adapté à la résolution exacte de notre problème. Par contre, il peut fournir une solution

approximative satisfaisante et pouvant servir de point de départ à une méthode de recherche de placement exact.

Au contraire, le concept de CSP dynamique introduit, compte-tenu des particularités de l'application étudiée, permet de représenter efficacement et de manière formelle notre problème de sorte que des algorithmes robustes de résolution de CSP puissent être utilisés.

6. DES FACTEURS FONCTIONNELS IMPORTANTS : APPLICATION A LA CONCEPTION D'USINE

La conception d'une nouvelle usine est toujours une tâche très complexe ; les choix sont nombreux et difficiles, ils doivent souvent être faits dans des délais courts, par une équipe d'hommes ayant des préoccupations complémentaires mais contradictoires.

Cette conception consiste généralement en la définition d'un aménagement des différents organes composant l'usine. Tous les aspects, des plus abstraits aux plus spécifiques, doivent être définis précisément. En effet, l'empirisme et l'improvisation conduisent souvent à des erreurs de conception qui rendent l'usine construite moins efficace que prévue.

6.1. Présentation du concept d'usine

Le problème de conception d'usine peut se poser essentiellement dans deux situations : lors de la construction d'une nouvelle usine ou lors de la réhabilitation d'une usine ancienne. Dans le second cas, le plus important travail consiste à réorganiser l'atelier constitué de nombreuses machines. La disposition des machines est fonction de la série de pièces à construire : on suppose être dans le cas où la réorganisation de l'atelier est plus avantageuse qu'une utilisation avec perte de temps de l'atelier existant. Ceci étant vrai notamment lorsque :

- l'atelier existant est très mal adapté à la nouvelle fabrication ;
- une grande série d'objets doit être construite sur une durée assez longue.

Les objectifs souhaités pour toute usine actuellement, sont la qualité, la compétitivité ainsi qu'une bonne adaptation à l'évolution de plus en plus rapide du marché. En effet, pour une usine manufacturière, le but est de transformer le plus efficacement possible des matières premières de façon à obtenir des produits finis pour un coût minimum. Pour ce faire, le concepteur de l'usine doit donner la possibilité aux différents organes de cette dernière de coopérer de manière productive. Ces différents organes sont au nombre de quatre et sont représentés sur la figure 3.40.

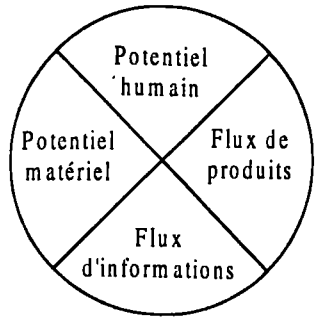


Figure 3.40. *Le concept d'usine*

Potentiel humain : toutes les personnes intervenant sur le site de l'usine.

Potentiel matériel : bâtiment / machines outils / informatique industrielle, c'est-à-dire tout ce qui est nécessaire à la production, la manutention et le stockage.

Flux de produits : tous les produits entrant dans l'usine, les matières premières évidemment mais également des produits semi-finis, des articles de conditionnement, des pièces de rechanges ou des fournitures diverses.

Flux d'information : ensemble de l'organisation de l'information et des circuits de décision de l'usine.

6.2. Paramètres caractérisant l'espace industriel

Les différentes entités définissant le concept d'usine font intervenir un certain nombre de paramètres à prendre en compte lors de la conception d'une usine. Dans [BEN 85], un grand nombre de ces paramètres est présenté de manière structurée. Nous présentons dans le tableau de la figure 3.41 cette hiérarchie établie en fonction des différents intervenants mais également des étapes de la conception.

1. Fonctions du bâtiment industriel

Le produit.

2. Localisation

•Choix du site (rôle des différents acteurs : entreprises, pouvoirs publics, ...) ;
•Circulation, accès ;
•Insertion dans le paysage, urbanisme.

3. Réalisation

3.1. Investisseur	3.2. Conception	3.3. Construction
•qui, comment ? •propriétaire - locataire, •hôtel industriel, usine relais.	•délais, équipe, qui, comment ? •quelle prévision de marché ? •quel plan d'occupation ? •optimisation de quoi ?	•délais, •qui, comment (financement) ? •usine clef en main.

4. Caractéristiques physiques de l'établissement

4.1. Enveloppe	4.2. Outil de production / manutention / stockage	4.3. Relations entre bâtiments et outil de production
•hauteur, largeur, profondeur, •niveaux, •volumétrie, •matériaux (démontable (?)), •portée, trame, •systèmes modulaires, •couleurs, éclairage, •architecture, taille.	•performance - capacité, •exigences (température, vibration), •qualités, économies, •âge - durée de vie - remplacement, •avance technologique, •productivité, •arrêts (coûts, flexibilité).	•dépendance, indépendance, •flexibilité.

5. Organisation

5.1. Volumétrique	5.2. Outil de production	5.3. Place et importance de :
•aménagement intérieur, •ateliers éclatés, •ensemble compact.	•chaîne compacte - éclatée, •centres d'usinages, •bouts de chaîne, •conditionnement, •stockage, •contrôle (norme), •évolution prévisible.	•bureau d'étude, administration, •production, maintenance, •contrôle, gestion, stocks, •méthodes, marketing, •service client - fournisseur, •laboratoire / recherche, •locaux sociaux, restaurant, •locaux syndicaux.

5.4. Des hommes	5.5. Préventive
•répartition des tâches, conditions de travail, •hiérarchie, services, locaux sociaux, •syndicats, comité d'entreprise, motivation.	•lutte contre l'incendie, •lutte contre la pollution, •sécurité des personnes.

6. Flux

6.1. D'information	6.2. De matières	6.3. Des hommes
•interne / externe, •décision d'investir, de produire.	•verticaux, horizontaux.	•longueurs des trajets,

7. Modèles

7.1. Architectural	7.2. De vie morale et humaniste.

8. Image de marque de l'entreprise

quelle est-elle ? qu'est-ce qui la représente ?

8.1. Pour le personnel	8.2. Pour le patronat	8.3. Pour le client
8.4. Pour toute personne extérieure		8.5. Publicité

9. Environnement

9.1. Économique :
•prévision, crédit, financement,
•fonction dans la chaîne de production.

Figure 3.41. *Paramètres caractérisant l'espace industriel*

Ces différents paramètres ne sont pas tous pris en compte simultanément, la conception d'une usine est obtenue grâce à différentes étapes successives mais non indépendantes et des remises en cause peuvent être faites. La figure 3.42 donne en exemple une étape de la conception d'une usine, cette figure permet de constater les dépendances entre chaque concept.

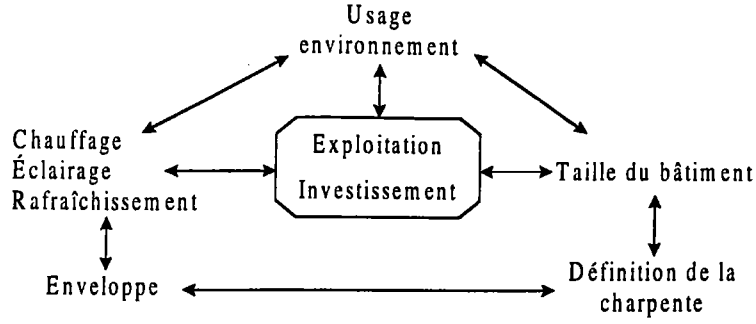


Figure 3.42. *Schéma d'analyse conceptuelle de l'enveloppe industrielle*

Ces différentes étapes font intervenir différents bureaux d'études :

- le bureau d'étude financière qui gère tous les problèmes de coût des travaux, prêts possibles (intérêts, échéances, ...) ;
- le bureau d'étude des questions d'environnement pour l'adaptation du sol, la proximité des clients (liaisons ferroviaire, maritime, aéroport, ...) ;
- le bureau d'étude BTP pour la construction de l'enveloppe ;
- le bureau d'étude de l'organisation interne qui doit simuler le fonctionnement de l'atelier de façon à déterminer l'agencement le mieux adapté.

C'est dans cette dernière étude que l'approche CAO intervient, elle peut permettre de trouver la meilleure structuration interne de l'atelier qui a déjà été placé dans l'usine grâce à des concepts amonts. Le problème qui nous préoccupe est la disposition des différentes machines dans l'atelier ainsi que la stratégie de placement mise en œuvre.

6.3. Ordonnancement d'atelier

Dans cette étude, nous avons limité les paramètres pris en compte à ceux qui nous semblent les plus fondamentaux ; les paramètres gérés sont donc liés :

- à une base de règles générales ;
- à un ensemble de contraintes mettant en relation les différentes entités manipulées ;
- aux flux de production ;
- aux machines.

Dans cette première partie, nous présentons brièvement ces différentes notions que nous développerons plus largement ultérieurement.

6.3.1. Contraintes d'ordre général

Ces contraintes sont issues d'un ensemble de règles constituant un fond commun à toutes les applications et dérivées de règles ou de consignes liées à des normes de sécurité, des recommandations de fabricants ou encore au bon sens.

Par exemple, toute machine libérant de l'eau ou de la vapeur d'eau, doit être éloignée d'une certaine distance de sécurité de toute prise électrique non isolée. Ou encore, si la production d'une machine est l'entrée d'une autre machine, il doit exister un moyen de faire transiter la production de l'une à l'autre (tapis roulant pour un objet, câble pour de l'information, ...).

6.3.2. Contraintes liant différents objets

L'étude bibliographique réalisée et liée aux problèmes d'aménagement spatiaux a mis en évidence une répartition assez courante des contraintes en trois catégories qui sont, des plus abstraites aux plus concrètes, les contraintes fonctionnelles, les contraintes topologiques et les contraintes géométriques. Pour les trois types de contraintes, on peut distinguer des contraintes unaires caractérisant des propriétés de l'objet et des contraintes binaires caractérisant les liens éventuels de l'objet avec d'autres objets ou l'environnement. Les contraintes géométriques sont déduites des contraintes topologiques elles-mêmes dérivées des contraintes fonctionnelles. Toutes les contraintes fonctionnelles peuvent ainsi être exprimées à l'aide d'une ou plusieurs contraintes topologiques ou géométriques.

6.3.3. Flux de produits

Les flux d'informations ou d'éléments entre les différentes machines d'un atelier, par exemple, sont décrits dans le cahier des charges de manière informelle. Au niveau fonctionnel de l'étape de conception d'un atelier, ces flux sont modélisés formellement pour pouvoir être pris en compte lors de cette étude.

Ces flux peuvent être modélisés pour une machine comme :

- des entrées : des objets à un stade non terminal, des matières premières, ...
- des sorties : des objets à un état n+1 de leur processus de conception, des production, ...
- des conséquences : des déchets, des fumées, des vapeurs d'eau, des copeaux, ...

La figure 3.43 représente en (a) et (b) deux types généraux de machines possibles et les figures 3.43 (c) et 3.43(d) représentent respectivement des exemples concrets de telles machines.

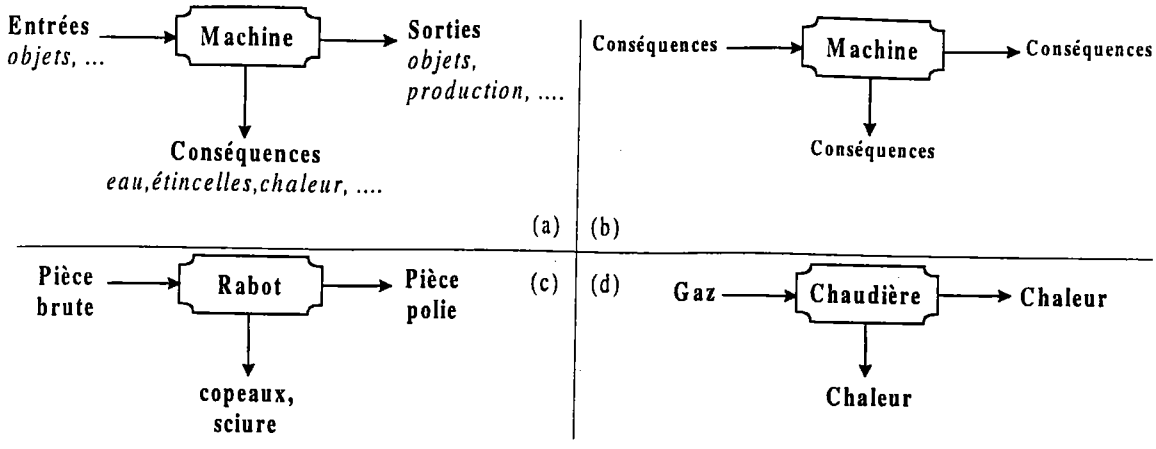


Figure 3.43. *Représentation de machines et de leurs flux*

Les flux d'éléments ou d'informations sont modélisés pour pouvoir être pris en compte lors de la conception de tout aménagement. Deux types de règles fonctionnelles sont définies à partir du cahier des charges : les interdictions et les obligations.

Dans le cas du placement spatial d'objets contraints, une règle d'interdiction correspond à l'impossibilité totale de mettre en présence deux objets. Par exemple, il est strictement interdit d'utiliser tout type d'objet en fer comme des clous, dans un atelier où sont manipulés des explosifs. Les règles d'obligation quant à elles sont essentiellement des règles de voisinage maximum ou minimum concernant le placement des machines les unes par rapport aux autres ou par rapport à l'environnement.

6.3.4. Objets

Nous avons vu précédemment que les flux entrant et sortant des machines sont représentés ; nous modélisons également toutes les caractéristiques et propriétés des différentes machines considérées. Ces caractéristiques peuvent être de base comme les dimensions, le poids ou la profondeur mais aussi plus complexes lorsqu'elles représentent des zones d'accessibilité, des parties qui peuvent se déplacer autour d'un axe de rotation, ... Un type d'objet particulier est également considéré, il s'agit des éléments de liaison comme les tapis roulants ou les câbles de connexion qui rendent possible l'échange de données entre machines.

Ces différentes notions font partie des informations permettant de définir l'organisation de l'atelier. Cependant après avoir déterminé le placement précis de chacune des machines, il peut être nécessaire de définir une stratégie de placement. En effet, le placement trop hâtif d'une machine dans un atelier peut empêcher la réalisation du placement même correct d'autres machines. Il peut donc être préférable de calculer a priori le chemin permettant de faire transiter

les différentes machines de l'atelier de manière à obtenir la configuration finale établie pour l'atelier. Différentes étapes pour le processus de placement pourront être définies certaines permettant d'obtenir une configuration locale d'autres une configuration globale.

6.4. Synthèse

Dans cette partie, nous avons abordé la création d'une usine d'un point de vue conceptuel en présentant les principaux axes d'études et en précisant celui qui intéresse plus particulièrement la CAO. Nous avons montré que les facteurs fonctionnels peuvent être très différents et même relativement éloignés des concepts simplement géométrique ou de positionnement.

Nous avons présentée brièvement les données manipulées et les problèmes à envisager car dans les parties précédentes, nous avons exposé plus précisément une méthode de modélisation et de détermination d'aménagement spatial sous contraintes.

L'aménagement de cuisine intégrée et l'aménagement d'atelier sont des problèmes similaires qui font intervenir les mêmes types de données. Cependant, nous avons préféré illustrer notre propos grâce à l'exemple du problème d'aménagement de cuisine intégrée. Cette préférence est liée à nos connaissances respectives des deux problèmes envisagées ; en effet, le problème d'aménagement de cuisine intégrée nous étant plus familier, il nous était plus commode de définir un cahier des charges cohérents. De plus dans ce domaine, il nous a été possible de rencontrer des professionnels et de connaître ainsi précisément les problèmes importants.

7. CONCLUSION

Nous avons présenté dans ce chapitre, deux méthodes, l'une algorithmique et l'autre utilisant le concept de CSP, permettant de placer, dans un espace donné, des objets simples modélisés par un rectangle ou des objets complexes modélisés par une union de rectangles. Les meubles sont disposés a priori horizontalement sur la surface de placement. Pour une orientation différente, il est éventuellement possible de considérer l'englobant de l'élément. L'espace de placement est modélisé par une union de rectangles, on détermine la précision de la représentation en fonction du pas de discrétisation.

La méthode algorithmique permettant de réaliser les placements des différents éléments dans l'espace de placement est précédée d'une étude heuristique permettant d'éliminer un certain

nombre de mauvaises solutions relativement tôt dans le processus de recherche. Cette méthode est ensuite améliorée par une utilisation d'informations complémentaires modélisées à l'aide d'une structure de graphe.

Dans les perspectives d'amélioration de notre travail, on peut envisager l'extension du modèle de base utilisé pour modéliser les meubles ainsi que l'espace, notamment en ajoutant les arcs de cercle, les triangles, ... aux primitives de base utiles à la modélisation.

La phase préliminaire de rejet de mauvaises solutions peut également être améliorée en ajoutant ou en précisant les heuristiques disponibles (plus spécifiques au domaine) de façon à étudier le plus rapidement possible en détail une configuration aboutissant à une solution optimale.

Finalement, l'inclusion d'un algorithme de type "ramasse miettes" pour regrouper périodiquement certaines zones de l'espace de placement peut être envisagée de manière à diminuer le nombre d'éléments (rectangles ou nœuds du graphe) à manipuler.

CONCLUSION

Les travaux que nous avons présentés dans ce document contribuent à la prise en compte plus importante des contraintes par les logiciels de CAO. D'une part, nous nous sommes intéressés à la normalisation des contraintes géométriques 2D en conception de pièces. Cette étude nous a permis de développer différentes méthodes de comparaison de modèles par normalisation de graphes. D'autre part, nous avons étudié l'introduction des contraintes fonctionnelles dans les systèmes de CAO et en particulier dans les problèmes de placement d'objets contraints dans un espace défini.

L'étude réalisée fournit certaines solutions ou pistes de réflexion aux problèmes soulevés par l'étude bibliographique du premier chapitre. Cette étude bibliographique soulève un premier problème qui est l'accroissement des fonctionnalités et des capacités requises pour les systèmes de CAO. Nous en avons présenté quelques unes ainsi que des méthodes permettant de mettre en œuvre leur réalisation.

Parmi les concepts novateurs recherchés, nous nous sommes intéressés à la notion de contraintes fonctionnelles pour une prise en compte d'un ensemble de plus en plus large de contraintes. L'étude des différents systèmes appartenant à des domaines variés met en évidence une prise en compte non encore systématique de ce type de contraintes. De plus, lorsque les contraintes fonctionnelles sont prises en compte, elles sont généralement transformées en contraintes géométriques ou topologiques.

Le deuxième chapitre est consacré à la présentation d'une méthode, de sa validité à sa mise en œuvre. Cette méthode facilite la réutilisation, concept fondamental dans tous les domaines et en particulier en CAO. En effet, notre objectif est de fournir à l'utilisateur d'un système de conception de pièces mécaniques, une assistance réelle en phase de création. Cette aide apportée par le système, est envisagée comme des suggestions de réutilisations totales ou partielles de pièces précédemment construites. Ce principe est basé sur le parcours d'une bibliothèque de pièces référencées et sur une comparaison de ces différentes pièces avec la pièce en cours de conception.

Nous avons montré que, pour des raisons d'historique de construction et de représentations multiples des combinaisons de contraintes, la comparaison de modèles ne permet pas dans tous

les cas de détecter une égalité ou une inclusion entre deux pièces. Nous avons ainsi proposé dans un cadre bien précis un ensemble de règles permettant d'obtenir une représentation normalisée de toute pièce, c'est-à-dire une représentation qui soit unique et minimale. La terminaison et l'unicité du processus de normalisation permettant de transformer le graphe utilisateur en un graphe normalisé unique, ont été prouvées dans notre contexte. Le nombre des contraintes même en deux dimensions pourraient être augmenté ce qui implique une mise à jour de l'ensemble des règles et une vérification des propriétés précédemment prouvées sur cet ensemble. Nous utilisons ensuite ces représentations normalisées pour réaliser la comparaison de pièces et détecter leur égalité ou inclusion. Nous avons développé deux méthodes de comparaison de modèles : une méthode algorithmique et une méthode utilisant le formalisme CSP. Finalement, nous avons évalué ces deux méthodes et comparé leurs performances.

L'étude de ces deux méthodes pourrait être approfondie. La méthode algorithmique dispose actuellement de certains filtres dont le nombre pourrait être augmenté et l'utilisation améliorée. En effet, une utilisation contextuelle d'une partie des filtres disponibles pourrait être bien plus efficace qu'une utilisation systématique de tous les filtres possibles. De même, la décomposition suivant un niveau fixé a priori est moins performante qu'une décomposition dynamique des classes d'équivalence en fonction du nombre d'éléments concernés.

Dans le cas de la seconde méthode basée sur le formalisme CSP, nous nous sommes appuyés sur une étude comparative extérieure pour le choix de l'algorithme utilisé, en l'occurrence celui de Forward-checking. Ce choix et par-là même l'efficacité de la méthode CSP pourraient être au moins partiellement remis en cause après une étude des différents algorithmes de résolution de CSP directement appliqués à notre problème. L'introduction de filtres adaptés renforce cette nécessité d'évaluer à nouveau les méthodes.

Le troisième chapitre aborde la modélisation et la gestion des contraintes fonctionnelles dans un domaine particulier qui est l'aménagement spatial d'objets contraints. Nous avons précisé le domaine de notre étude en choisissant comme application l'aménagement d'une cuisine intégrée. Nous avons proposé une modélisation pour les données du problème : les meubles, les appareils ménagers et l'espace de placement. Après consultation d'un professionnel en agencement de cuisines intégrées et l'utilisation de logiciels existants, des contraintes supplémentaires ont également été proposées.

À nouveau, deux méthodes de résolution ont été envisagées : l'une algorithmique disposant de plusieurs optimisations et l'autre utilisant le formalisme CSP qui a été envisagé sous différentes approches. La modélisation des différents constituants a été réalisée de manière satisfaisante à l'aide de rectangles ou des unions de rectangles. Nous pourrions éventuellement affiner cette modélisation en considérant d'autres éléments de base (arcs de cercle, triangles, ...) et en faisant intervenir des compositions d'éléments. Une modélisation plus fine des objets pourraient ainsi permettre de gagner de la place en encastrant certains meubles ou appareils. On pourrait également après avoir prévu le placement de chaque objet, ajouter une étape de création d'un planning au processus de conception d'une cuisine ou d'un atelier. Ce planning définirait le placement des machines dans l'atelier (ordre d'entrée et d'installation) tout en respectant l'organisation définie précédemment. De plus, les contraintes fonctionnelles considérées jusqu'à présent sont traduites en contraintes géométriques ou dimensionnelles. On pourrait envisager la prise en compte réelle (sans traduction) de contraintes fonctionnelles comme, par exemple, certaines contraintes temporelles. En effet, il pourrait être intéressant de prendre en compte la durée de vie d'une machine qui pourrait être remplacée avant, en même temps ou après une machine empêchant sa sortie de l'atelier. Lors de la fabrication, l'état d'un produit peut également influencer l'agencement d'un atelier. Par exemple, si un produit sort d'une machine à une température donnée et doit être traité par une autre machine à cette même température, la durée de la transition devra être minimale. Finalement, l'approche CSP est suggérée pour ce problème, mais mériterait d'être approfondie de manière à comparer son efficacité avec celle de la méthode algorithmique implémentée.

Les méthodes de comparaison de pièces présentées permettent d'une part, de réduire la taille des bibliothèques de pièces en les regroupant par familles et d'autre part, de réutiliser au maximum des travaux précédents. Ces méthodes ont été envisagées, dans un premier temps, uniquement de manière itérative. Cependant, ce problème peut être naturellement parallélisé puisque les comparaisons indépendantes du graphe courant avec différents éléments de bibliothèques peuvent être réparties sur différents processeurs. Les problèmes soulevés sont principalement dus au nombre de solutions à comparer dans un système réel. Les principales solutions envisagées à ces problèmes sont liées aux notions de parallélisme ainsi qu'à la classification des solutions par l'intermédiaire d'un processus de normalisation. L'introduction du parallélisme peut permettre d'augmenter la fréquence des comparaisons réalisées par le système. En effet, les comparaisons entre la pièce en construction et les éléments de

bibliothèques doivent être faites pratiquement en temps réel car elles sont censées faciliter le processus de conception et non le ralentir. Ainsi, il est nécessaire de définir une étape de conception à partir de laquelle il devient significatif de rechercher des égalités ou des inclusions entre pièces. Une fréquence de comparaison doit également être établie si les moyens matériels ne permettent pas de réaliser systématiquement des comparaisons à chaque modification de la pièce.

La prise en compte de la troisième dimension reste un problème toujours très délicat quelle que soit l'étude menée. Au deuxième chapitre, le degré de complexité s'accroît considérablement si on envisage une gestion d'éléments et de contraintes en trois dimensions. En effet, l'ensemble des règles est délicat à écrire et la complétude de la méthode difficile à prouver. En conséquence, nous envisageons de décomposer le problème en plusieurs étapes. Une première étape consisterait à extraire les différentes caractéristiques de la pièce à partir d'une représentation géométrique normalisée. Une modélisation topologique normalisée des différentes pièces serait ainsi déduite et pourrait servir de support à la comparaison des éléments. Des règles permettant de normaliser toutes les situations doivent être définies. On définira ainsi des règles de décomposition ou de recomposition. Par exemple, plaçons-nous dans le cadre de la modélisation par les caractéristiques de formes, plusieurs représentations normalisées de l'intersection de deux rainures peuvent ainsi être envisagées :

- en considérant intégralement les deux rainures ;
- en considérant quatre rainures et un trou carré ;
- ...

Pour notre étude portant sur l'aménagement spatial, la prise en compte de la troisième dimension n'est que partielle et apparaît en tant que vérification de la contrainte "au-dessus de" ou d'incompatibilités (chevauchement, obstruction d'ouverture, ...). On peut se demander si le bénéfice apporté par cette prise en compte plus précoce dans le processus de placement justifie l'augmentation de complexité inévitable des algorithmes existants.

RÉFÉRENCES BIBLIOGRAPHIQUES

[ALD 88] ALDEFELD B., Variation of geometries on a geometric-reasining method, *Computer-Aided Design*, Vol.20, n°3, Avril 1988, p.117-126.

[AND 86] ANDRE J.M., Vers un système d'aide intelligent pour l'aménagement spatial : CADOO, *CIIAM 86 Intelligence Artificielle, Actes du 2ème Colloque International d'Intelligence Artificielle*, 1-5 décembre 1986, Marseille, p.31-47.

[BEN 85] BENCHIMOL G. (coordinateur), La conception des usines de demain, *Hermès*, 1985.

[BRO 88] BROWN A.D.,Automated placement and routing, *Computer-aided Design*, Vol.20, n°1, 1988, p.39-44.

[CAO 90] CAO X., HE Z., PAN Y.,Automated design of house-floor layout with distributed planning, *Computer-aided Design*, Vol.22, n°4, 1990, p.213-222.

[CAR 94] CARDEIRA C., MAMMERI Z., Ordonnancement de tâches dans les systèmes temps réel et répartis, *APII*, Vol.28, n°4, 1994, p.353-384.

[CAU 95] CAUX C., PIERREVAL H., PORTMANN M.C., Les algorithmes génétiques et leur application aux problèmes d'ordonnancement, *APPI*, Vol.29, n°4-5, 1995, p.409-443.

[CHU 90] CHUNG J.C.H, SCHUSSEL M.D., Technical evaluation of variational and parametric design, *Computers in Engineerig 1990*, p.289-298.

[CLE 87] CLERC T., SADGAL M., ORCHAMPT P., Prototype d'un système de placement intelligent dans le plan, *Actes de MICAD 87*, p.491-503.

[ERQ 95] ESQUIROL P., LOPEZ P., Programmation logique avec contraintes et ordonnancement, *APII*, Vol.29, n°4-5, 1995, p.379-407.

[GAR 83] GARDAN Y., LUCAS M., Techniques graphiques interactives et CAO, *Hermes Publishing*, 1993.

[GAS 77] GASCHNIG J., A general backtracking algorithm that eliminates most redundant tests, *Proceedings IJCAI-77*, Vol.1, 1977, p.457.

[GAS 79] GASCHNIG J., Performance measurement and analysis of certain search algorithms, *Technical Report CMU-CS-79-124*, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[GOS 88] GOSSARD D.C., ZUFFANTE R.P., SAKURAI H., Representing dimensions, tolerances, and features in MCAE Systems, *IEEE Computer Graphics & Applications*, mars 88, p.51-59.

[GRA 93] GRABOT B., GENESTE L., DUPEUX A., Ordonnancement réactif par utilisation des règles floues paramétrables, *Revue d'automatique et de productique appliquées*, Vol.6, n°3, 1993, p.273-290.

[HAN 90] HANSER J.M., CFAO et propagation de contraintes, technologie, enjeux et étapes industrielles, *Actes de MICAD 90*, p.202-231.

[HAR 79] HARALICK R.M., SHAPIRO L.G., The consistent labeling problem : part I, *IEEE Transactions on Pattern Analysis & Machine Intelligence*, n°3, 1980, p.193-203.

[HAR 80] HARALICK R.M., ELLIOTT G.L., Increasing tree search efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, n°3, 1980, p.263-313.

[HER 95] HERTZ A., WIDMER M., La méthode TABOU appliquée aux problèmes d'ordonnancement, *APII*, Vol.29, n°4-5, 1995, p.353-378.

[KAM 92] KAMEYAMA K.I., Real-Time Constraint Cheching in Design Process, *Concurrent Engineering, automation, tools and techniques*, Andrew Kusiak, 1992, p.111-130.

[KWA 97] KWAITER G., GAILDRAT V., CAUBET V., Oranos : un solveur dynamique de contraintes hiérarchiques et géométriques, *Revue de CFAO et d'informatique graphique*, Vol.12, n°1-2, 1997, p.139-152.

[KYU 92] KYUNG C.M., WIDDER J., MLYNSKI D.A., Adaptive cluster growth : a new algorithm for circuit placement in rectilinear regions, *Computer-aided Design*, Vol.24, n°1, january 1992, p.27-35.

[LIG 82] LIGHT R.A., GOSSARD D.C., Modification of geometric models through variational geometry, *Computer-Aided Design*, Vol.14, n°4, Juillet 1982, p.209-214.

[LEI 94] LEINEN S., Une nouvelle approche pour la modélisation et la propagation des contraintes en CAO, *Mémoire de DEA*, Centre de Recherche en Informatique de Nancy, 9 septembre 1994.

[LEI 95] LEINEN S., JUNG J-P., GARDAN Y., Une nouvelle approche pour la comparaison de modèles en CAO, *Rapport de Recherche*, Laboratoire de recherche en informatique de Metz, n°95/10, octobre 1995.

[LEI 97] LEINEN S., JUNG J-P., GARDAN Y., Comparaison de modèles de CAO par normalisation de graphes, *Revue internationale de CFAO et d'informatique graphique*, Vol.12, n°1-2, 1997, p.153-167.

[MED 96] MEDJDOUB B., YANNOU B., ARCHiPLAN : A new tool for conceptual design in architecture, *First International Conference : IDMME'96*, 15-17 april 1996, Nantes, p.205-214.

[MIS 96] MISUND G, JOHANSEN B.S., HASLE G., HAUKLAND J., Integration of Geographical Information Technology and Constraint Reasoning - A Promising Approach to Forest Management, *Rapport de recherche interne*, 1996.

[PIP 95] PIPIERI T., Implémentation d'un nouveau dialogue pour le système de CAO SACADO, *Mémoire d'ingénieur CNAM*, février 1995.

[PIQ 89] PIQUET F., POITOU J.P., TASSE J.C., La CFAO Concevoir et produire autrement, *Cisigraph/Nathan*, 1989.

[PRO 93] PROSSER P., Hybrid algorithms for the constraint satisfaction problem, Computational Intelligence, Vol.9, 1993, p.268-299.

[PRO 95] PROSSER P., Forward checking with backtracking, *Constraint Processing, LNCS 923*, 1995, p.185-204.

[ROL 91] ROLLER D, An approach to computer-aided parametric design, *Computer-aided Design*, Vol.23, n°5, june 1991, p.385-391.

[SAM 89] SAMET H., The design and analysis of spatial data structures, *Addison-Wesley Publishing compagny*, inc., 1989.

[SHA 95] SHAH J.J., MANTYLA M., Parametric and feature-based CAD/CAM -Concepts, Techniques, and Applications-, *Wiley & Sons*, inc., 1995.

[SCH 94] SCHWARZ A., BERRY D.M., SHAVIV E., On the use of the automated building design system, *Computer-aided Design*, Vol.26, n°10, october 1994, p.747-762.

[STA 77] STALLMAN R.M., SUSSMAN G.J., Forward reasonning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence*, Vol.9-2, 1977, p.135-196.

[TAR 72] TARJAN R., Depth-first search and linear graph algoritms, *SIAM J. Computer.*, Vol.1, n°2, june 1972, p.146-160.

[TIL 87] TILOVE R.B., A null-object detection algorithm for constructive solid geometry, *Communication of the ACM*, july 1984, Vol.27, n°7

[VER 90] ROCHON DU VERDIER F., Placement de rectangles par répartition dans une surface bornée, *Actes de MICAD 90*, p.634-648.

[VER 92] ROCHON DU VERDIER F., Résolution de problèmes d'aménagement spatial fondée sur la satisfaction des contraintes -Validation sur l'implantation d'équipements électroniques hyperfréquences-, *Thèse de Doctorat*, Université Claude Bernard, Lyon I, 7 juillet 1992.

[VER 95] VERFAILLIE G., SCHIEX T., Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches, *Revue d'intelligence artificielle*, Vol.9, n°3, 1995, p.269-309.

[WAL 60] WALKER R.L., An enumerative technique for a class of combinatorial problems, *Proceedings of the Symposium on Applied Mathematics*, Vol.10, Providence, RI, 1960, p.91-94.

ANNEXES

1. ANNEXE A : FORMES CLASSIQUES DE MODÈLES GÉOMÉTRIQUES

1.1. Modélisation par arbre de construction

Pour réaliser tout traitement sur un objet, il est nécessaire de disposer d'un certain nombre d'informations (positions, dimensions, etc.) ; ces données sont modélisées et stockées, en fonction de leur nature, dans différents modèles. En effet, on peut distinguer plusieurs types de modèles ([GAR 83], [SHA 95]) : le modèle descriptif, les modèles de fonctionnement ou les modèles de communication.

Le modèle descriptif, appelé également modèle géométrique, contient la description des objets, en particulier les notions géométriques et technologiques. Dans les systèmes de CAO, la modélisation géométrique est la base de nombreux traitements tant au niveau conception que fabrication. Le modèle choisi doit donc contenir toutes les informations nécessaires à la description de la forme des objets à représenter.

Il existe différents modèles géométriques dont les plus courants sont : le modèle par les limites, le modèle par historique (ou arbre de construction ou CSG : *Constructive Solid Geometry*), les modèles mathématiques ou les modèles paramétrés. Les modèles CSG sont représentés par un arbre. Les nœuds de l'arbre correspondent soit à une opération booléenne (union, intersection, différence) soit à une transformation géométrique (translation, rotation, homothétie) tandis que les feuilles de l'arbre sont soit des volumes primitifs, soit les paramètres des transformations géométriques.

Ainsi, à chaque nœud correspond un objet. Même si celui-ci n'est pas réellement calculé, il représente une étape intermédiaire dans la construction de l'objet final (figure A.2). Le modèle CSG est souvent vu comme l'historique de construction de la pièce modélisée. Dans les cas les plus simples, la représentation d'un modèle CSG peut être réalisée par un arbre mais souvent plusieurs copies du même sous-arbre peuvent apparaître. Dans ce cas, l'arbre actuel devient un graphe acyclique orienté.

Dans un contexte mécanique, l'ensemble minimal des primitives est composé du cube, du cylindre, du cône, de la sphère et du tore (figure A.1). Il est souvent intéressant de compléter cet ensemble par d'autres primitives même redondantes. Dans certains cas, il est préférable, par exemple, de considérer une pyramide à base carrée comme une primitive au lieu d'une différence entre un cube et quatre autres cubes.

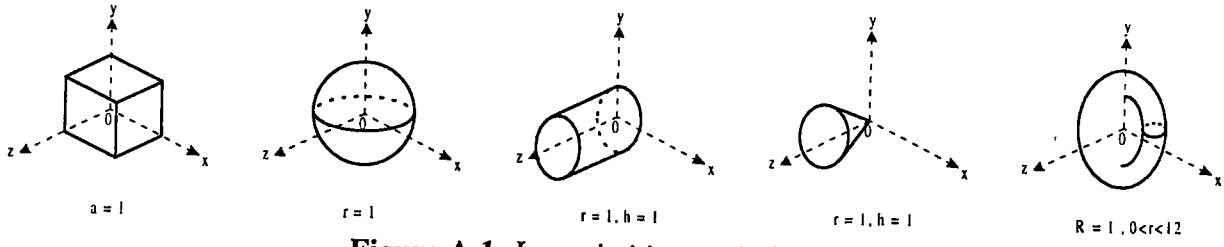


Figure A.1. *Les primitives unitaires*

Le modèle CSG est valide car à tout objet représenté correspond un objet réel et il est relativement ouvert puisqu'il existe de nombreux algorithmes pour réaliser différents traitements. L'arbre CSG est une description déclarative et implicite de la géométrie du solide.

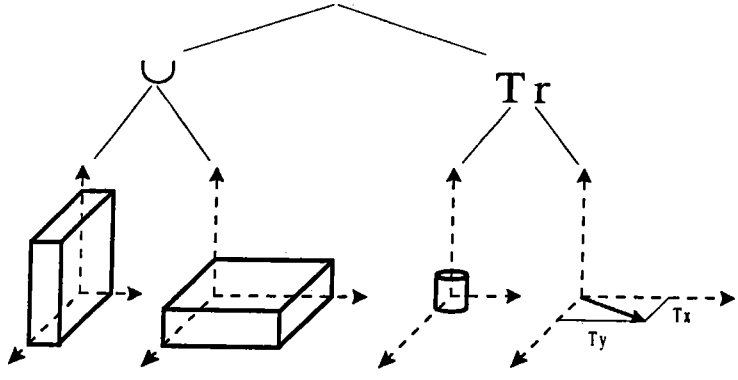


Figure A.2. *Un arbre de construction CSG*

1.2. Modélisation par les limites

Dans le modèle par les limites (B_Rep) ([SAM 89], [PIQ 89]), les solides sont définis par leurs surfaces délimitantes. Le modèle par les limites d'un objet est une description géométrique et topologique de ces frontières. Un objet est segmenté en un nombre fini de sous-ensembles limités appelés des faces. Chaque face est à son tour représentée par ses sommets et arêtes frontières. Un modèle par les frontières est donc constitué de trois entités primitives topologiques : les faces, les arêtes et les sommets. Les faces sont les portions continues de la surface du volume. Les arêtes sont les éléments dont la jonction forme une face frontière close. Les sommets sont les points de la surface correspondant à des intersections d'arêtes.

Les informations géométriques et topologiques doivent être associées aux entités topologiques individuelles. De part leur utilisation et leurs caractéristiques différentes, une séparation nette peut être faite dans le modèle par les frontières entre les descriptions topologiques et géométriques de l'objet. La description topologique est constituée par les relations d'adjacence entre les paires d'entités topologiques individuelles. La description géométrique correspond à la

forme et la localisation dans l'espace de chaque entité primitive topologique. Cependant les descriptions topologique et géométrique d'un objet ne sont pas indépendantes. Par exemple, un changement dans la description géométrique d'une entité primitive topologique telle qu'une arête passant de la description d'un segment rectiligne en une approximation linéaire par morceaux introduira de nouvelles entités topologiques.

Finalement, contrairement au modèle CSG, le modèle B-Rep mémorise les faces qui constituent la peau de l'objet (figure A.3). Chaque face est orientée pour pouvoir déterminer le côté où se situe la matière.

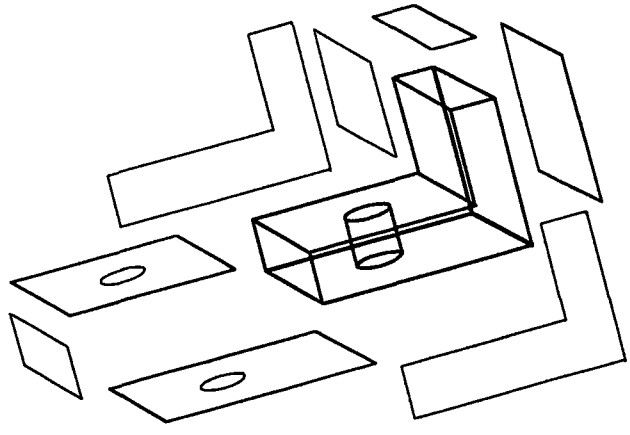


Figure A.3. Modèle B_Rep

Les deux principaux modèles de représentation des objets ont été présentés séparément dans cette partie mais nous verrons ultérieurement que le modèle idéal pour représenter un élément est souvent une modélisation hybride. En effet, un modèle combinant à la fois un arbre de construction ainsi qu'une représentation par les limites permettra de choisir en fonction du problème à résoudre, la représentation la plus efficace.

2. ANNEXE B : COMPARAISON DE MODÈLES CAO PAR NORMALISATION DE GRAPHES

2.1. Structure de données

2.1.1. Présentation générale

Une structure de graphe est utilisée pour représenter les différents objets contraints ainsi que les différentes contraintes du problème quelles soient propres à l'objet lui-même ou liant deux éléments.

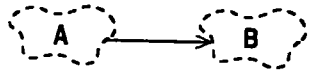
Différentes structures permettent donc de manipuler des graphes dans cette maquette. En effet, lors d'une session de trravail avec la maquette développée, plusieurs instances de graphes sont utilisées simultanément :

- un graphe courant (utilisateur) ;
- un graphe chargé (résultat d'une précédente utilisation) ;
- un graphe normalisé (résultat du processus de normalisation).

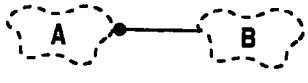
Un graphe est constitué de nœuds représentant les objets et d'arêtes permettant de modéliser les contraintes binaires. Le processus de création d'un objet dans le modèle affecte à tout nouvel objet un identifiant (un numéro d'objet) or un nœud est inséré dans le graphe courant pour le nouvel objet seulement si celui-ci est contraint. Il est donc nécessaire de gérer une table de correspondances entre les différents numéros de nœuds et les identifiants des objets de façon à pouvoir consulter des informations conservées dans le modèle car toute redondance entre ces deux représentations sont évitées.

La figure B.1 représente le diagramme des classes de notre maquette ; il est réalisé en utilisant les formalisme OOAD (Object-oriented Analysis and Design) de Grady Booch. Brièvement, ce formalisme représente par des nuages pointillés les différentes classes et trois relations d'association peuvent lier les relations entre classes :

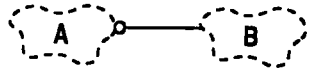
- héritage : A hérite de B ;



- composition : A est composée de B ;



- utilisation : A utilise B.



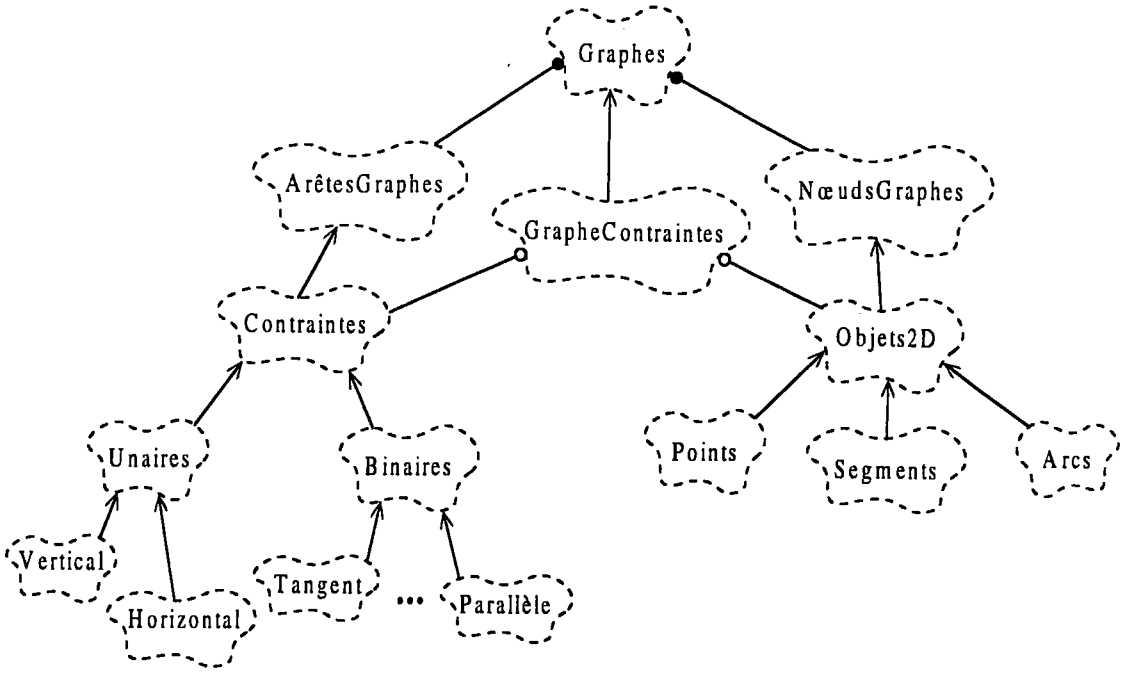


Figure B.1. *Diagramme des classes*

La classe modèle est composée et utilise d'autres classes que nous ne détaillerons pas ici car elles appartiennent aux noyaux du système SACADO ; par contre, nous décrivons les structures spécifiques à la réalisation de la comparaison de modèles de CAO par normalisation de graphes que nous avons développées.

2.1.2. Graphe

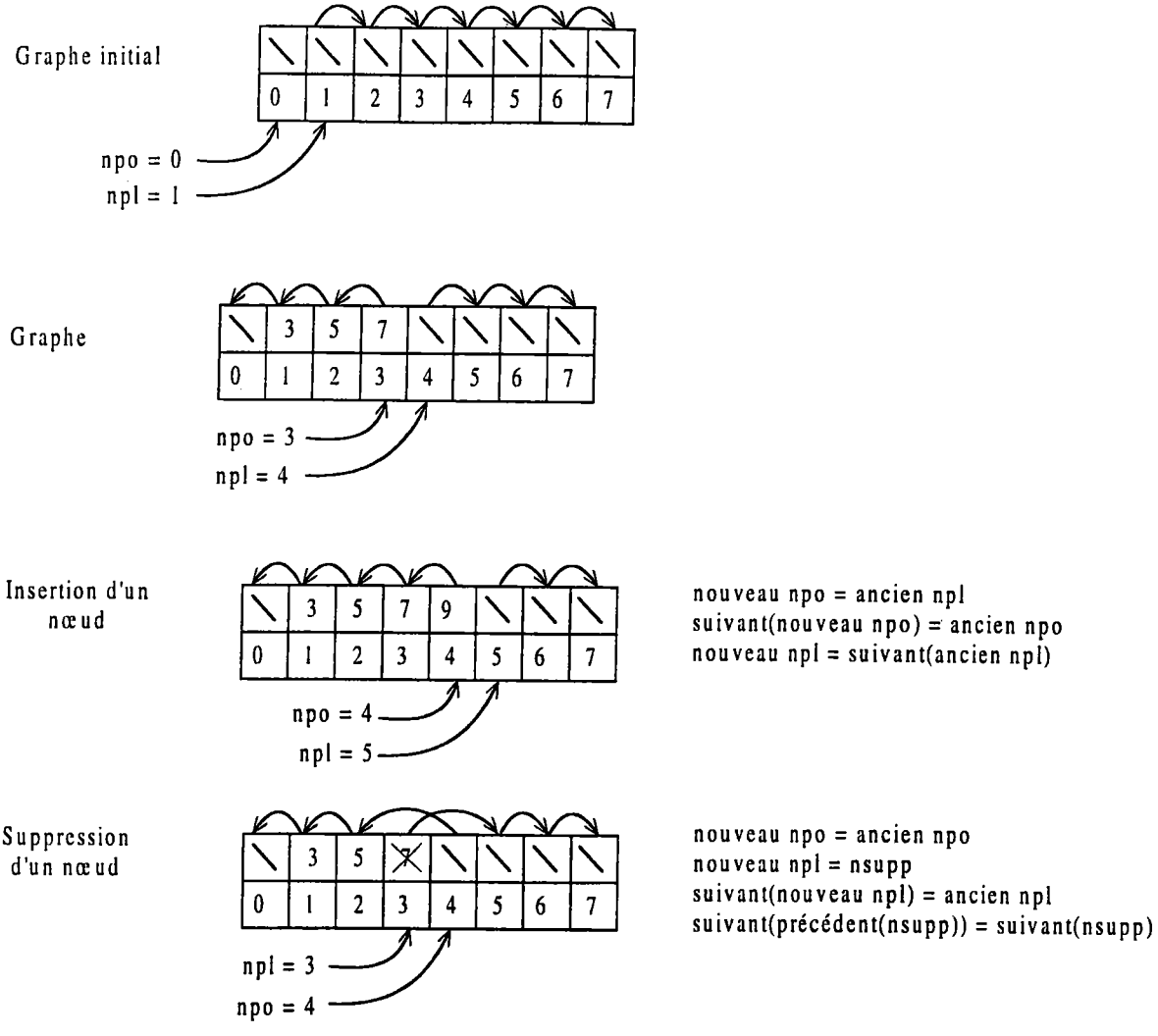
Nous avons déjà décrit précédemment de manière formelle la structure de graphe utilisée. Pratiquement, un graphe est représenté par un tableau ce qui permet d'éviter une gestion complexe de listes et de pointeurs. Les inconvénients de l'utilisation de tableaux c'est-à-dire les décalages en cas d'ajout ou de suppression d'un élément ou encore les cases libres non réutilisées, sont évités grâce à un double chaînage des données ; le chaînage d'une part des cases libres et d'autres part celui des cases occupées.

Deux variables sont donc associées à tout graphe, ces variables représentent le point de départ des deux chaînages et sont :

- le numéro de la première case libre (npl) ;
- le numéro de la première case occupée (npo).

Chaque case est ainsi chaînée à une case suivante, si la case est de type occupé (respectivement libre), elle est chaînée à une case occupée (respectivement libre). Le numéro représentant la première case libre permet d'insérer un nouveau nœud dans le graphe, le nœud est

inséré dans la case indiquée par la variable npl et cette variable est mise à jour avec la valeur indiquée par le suivant. La seconde variable représentant la première case occupée permet de parcourir tout le graphe en passant d'une case occupée à la case occupée suivante.



Dans notre maquette, les variables chaînées ne sont pas simplement des entiers comme dans les exemples ci-dessus mais ce sont des nœuds représentant les différents éléments contraints du modèles. De plus, à chaque nœud est associé également, en utilisant un certain chaînage les différentes arêtes du nœud considéré.

2.1.3. Nœud

En effet, il existe une structure appelée TabArête contenant une représentation de toutes les arêtes du graphe. Chaque arête étant décrite par son nœud origine et son nœud extrémité, il est associé à ces deux nœuds le numéro de l'arête suivante ayant soit comme origine soit comme extrémité le nœud considéré. Chaque nœud contenant lui-même dans sa description le numéro de sa première arête ainsi que celui de sa dernière.

	N° première arête	N° dernière arête
Nœud 5	5	5
Nœud 4	3	4
Nœud 3	2	4
Nœud 2	1	3
Nœud 1	1	5

	N° nœud origine	N° nœud extrémité	Arête suivante origine	Arête suivante extrémité
Arête 5	1	5	\	\
Arête 4	3	4	\	\
Arête 3	2	4	\	4
Arête 2	1	3	5	4
Arête 1	1	2	2	3

Figure B.2. *Représentation simplifiée du chaînage des arêtes*

De manière plus détaillée, un nœud est une structure comportant six paramètres :

- le ***numéro de la première arête*** ayant comme extrémité le nœud considéré, il permet de parcourir toutes les arêtes ayant un nœud en commun car un chaînage est également défini sur la table des arêtes ;
- le ***numéro de la dernière arête*** de façon à savoir rapidement si le nœud a une seule arête d'où l'arrêt de recherche de suivant ;
- le ***numéro de nœud suivant*** qui correspond soit au numéro du prochain libre si le nœud considéré est libre, soit au numéro du prochain occupé si le nœud considéré est occupé ;
- le ***numéro d'objet*** car à chaque nœud du graphe correspond un objet du modèle contenant tous les paramètres géométriques (coordonnées, rayon, ...) ;
- le ***nombre d'arêtes***, cette variable est utile pour la répartition des nœuds en classes d'équivalence en fonction du nombre d'arêtes. En effet, il est ainsi plus rapide et moins coûteux de réaliser la répartition car il n'est plus nécessaire de parcourir le chaînage de toutes les arêtes liées au nœud traité de façon à calculer le nombre d'arêtes ;
- à chaque nœud, une ***contrainte unaire*** peut être associée. Cette variable est nécessaire du fait du type de représentation choisi pour modéliser cette information.

2.1.4. Arête

De manière plus détaillée, une arête est une structure comportant six paramètres :

- le ***numéro du nœud origine*** de l'arête ;
- le ***numéro du nœud extrémité*** de l'arête ;
- le ***numéro de l'arête suivante pour le nœud origine*** qui permet de retrouver toutes les arêtes ayant comme origine ou extrémité le nœud considéré ;
- le ***numéro de l'arête suivante pour le nœud extrémité*** qui permet de retrouver toutes les arêtes ayant comme origine ou extrémité le nœud considéré ;
- une ***marque*** (un booléen) permettant de marquer l'arête lors d'un parcours de la table des arêtes ;
- une variable indiquant le ***type de l'arête*** (explicite, implicite) ; cette variable est nécessaire du fait du type de représentation choisi pour modéliser l'information.

2.2. Égalité et inclusion de graphes

2.2.1. La maquette : présentation générale

La figure B.3 présente l'écran initial obtenu par l'utilisateur du système SACADO. On distingue de haut en bas, une zone de menus, des fenêtres graphiques et des zones de dialogue.

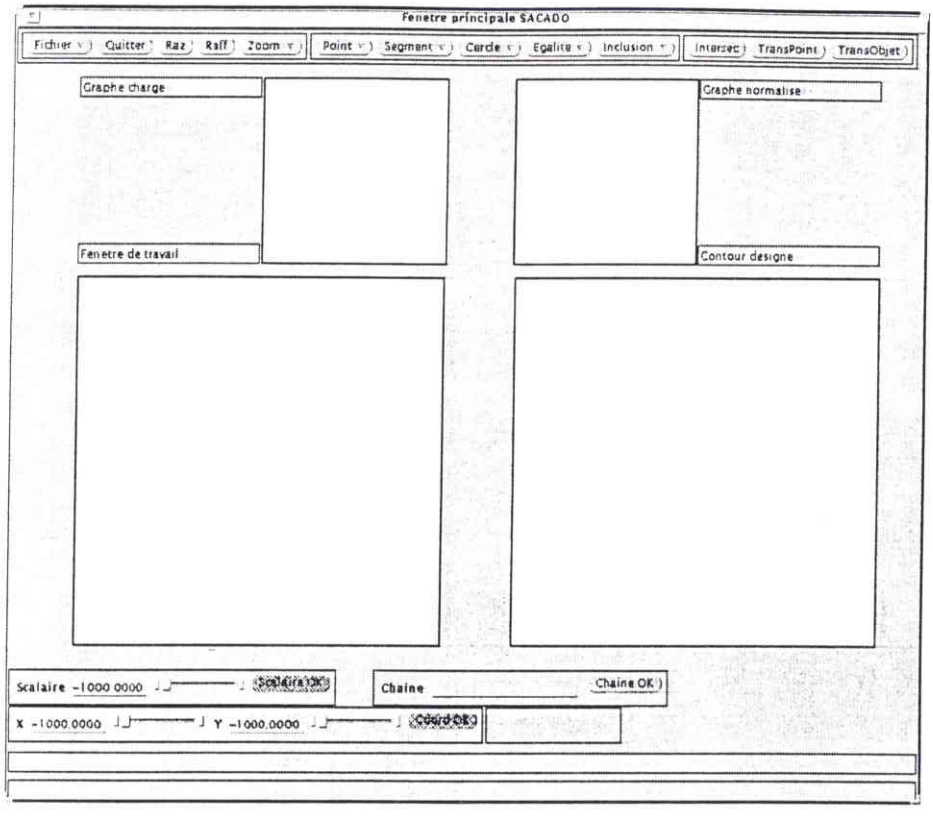
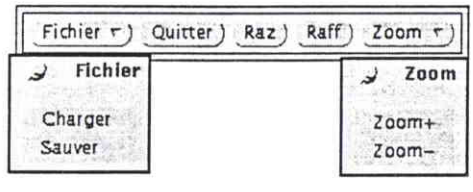


Figure B.3. *Écran initial de SACADO*

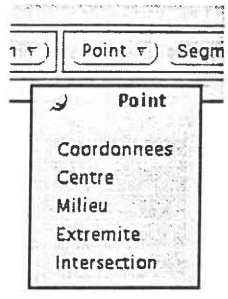
2.2.1.1.La zone de menus

Cette zone permet de sélectionner les différentes fonctionnalités de notre logiciel, nous les présentons brièvement ci-dessous :

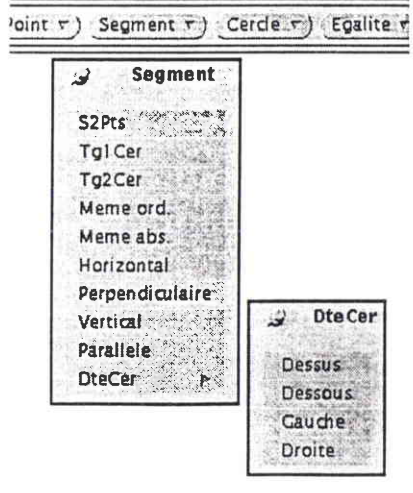


- l'option ***Charger*** du menu ***Fichier*** permet de restaurer une scène précédemment sauvées en créant le modèle associé aux données mémorisées dans un fichier texte ;
- l'option ***Sauver*** du menu ***Fichier*** permet de sauvegarder la scène courante sous forme d'un fichier texte contenant toutes les informations importantes ;

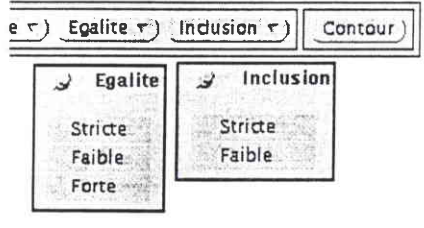
- le menu ***Quitter*** permet d'interrompre soit l'interaction en cours (création d'un segment, ...), soit la session de travail avec SACADO ;
- le menu ***Raz*** réinitialise le modèle c'est-à-dire qu'il détruit les données relatives à la construction en cours ;
- le menu ***Raff*** rafraîchit les différentes graphiques en réaffichant les différentes scènes ;
- le menu ***Zoom+*** du menu ***Zoom*** permet de visualiser un agrandissement de la zone sélectionnée ;
- le menu ***Zoom-*** du menu ***Zoom*** permet au contraire d'avoir une vue d'ensemble de la scène.



- le menu ***Point*** se décompose en différents sous-menus qui correspondent à diverses méthodes de création d'un point :
    - le menu ***Coordonnées*** définit un point en précisant ces coordonnées (x,y) soit à la souris soit au clavier ;
    - le menu ***Centre*** crée un point qui soit le centre du cercle désigné ;
    - le menu ***Milieu*** crée un point qui soit le milieu du segment désigné ;
    - le menu ***Extrémité*** crée le point extrémité d'un segment ;
    - le menu ***Intersection*** crée le point correspondant à l'intersection des deux objets désignés.



- Le menu ***Segment*** construit un segment en utilisant différentes méthodes de définition :
    - le menu ***S2Pts*** définit un segment quelconque en précisant simplement son origine et son extrémité ;
    - le menu ***Tg1Cer*** crée un segment tangent à un cercle ;
    - le menu ***Tg2Cer*** crée un segment tangent simultanément à deux cercles ;
    - le menu ***Même ord.*** définit un segment horizontal ayant la même ordonnée qu'un point désigné ;
    - le menu ***Même abs.*** définit un segment vertical ayant la même abscisse qu'un point désigné ;
    - le menu ***Horizontal*** crée un segment horizontal par désignation d'un point de passage ;
    - le menu ***Perpendiculaire*** crée un segment perpendiculaire au segment crée et passant par un point désigné ;
    - le menu ***Vertical*** permet la création d'un segment vertical par désignation d'un point de passage ;
    - le menu ***Parallèle*** crée un segment passant par le point désigné et parallèle à un segment donné ;
    - le menu ***DteCercle*** a différents sous-menus : ***Dessus***, ***Dessous***, ***Gauche*** et ***Droite*** qui permettent respectivement de créer un segment tangent au cercle désigné et horizontal d'ordonnée maximale, horizontal d'ordonnée minimale, vertical d'abscisse minimale et vertical d'abscisse maximale.



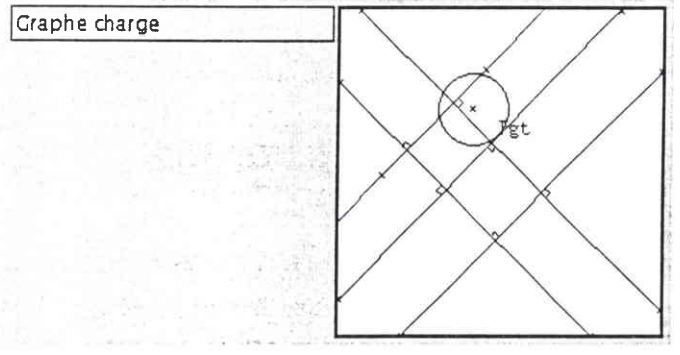
- Le menu ***Égalité*** permet de vérifier différents types d'égalité entre les graphes de la scène chargée et de la scène courante :
    - l'égalité ***Stricte*** est obtenue entre deux graphes lorsqu'ils ont exactement les mêmes éléments et les mêmes contraintes de même nature (les mêmes contraintes explicites ainsi que les mêmes contraintes implicites) ;
    - l'égalité ***Faible*** est réalisée lorsque les deux graphes considérés ont les mêmes objets et les mêmes types de contraintes (perpendiculaire, parallèle, ...) sans vérification de nature ;
    - l'égalité ***Forte*** consiste en une égalité stricte complétée par une relation dimensionnelle entre les deux figures modélisées ; il doit exister une composition de transformations affines permettant de passer de l'une des scènes à l'autre.

- Comme pour l'égalité on distingue plusieurs types d'inclusion. Le menu ***Inclusion*** a ainsi deux sous-menus :
    - l'inclusion ***Stricte*** obtenue lorsque un certain nombre d'objets et de contraintes sont présentes à la fois dans les deux graphes ;
    - l'inclusion ***Faible*** obtenue dans les mêmes conditions que l'égalité faible.

- Le menu ***Contour*** permet d'extraire un contour d'une scène créée d'un ensemble de segments et de cercles contraints ; le contour est obtenue par désignation successives des différents points d'intersection.

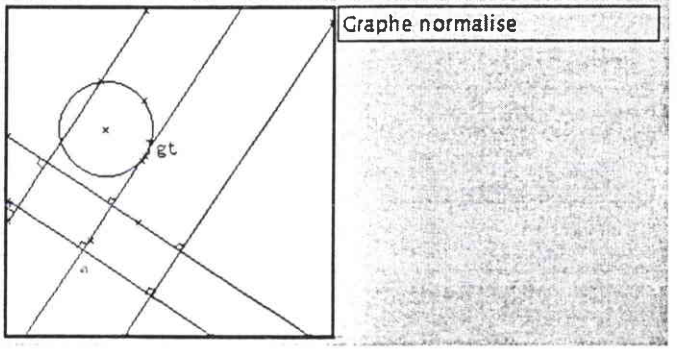
2.2.1.2.Les fenêtres graphiques

- La fenêtre ***Graphe chargé*** :



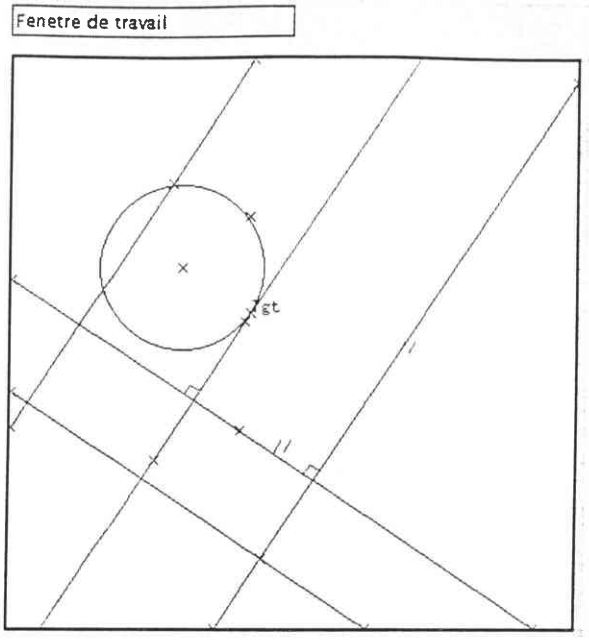
Cette fenêtre contient la représentation de la scène chargée, elle permet également de visualiser toutes les contraintes liant les différents éléments de la scène. Les contraintes affichées résultent du parcours du graphe normalisé de la scène précédemment sauvée.

- La fenêtre ***Graphe normalisé*** :



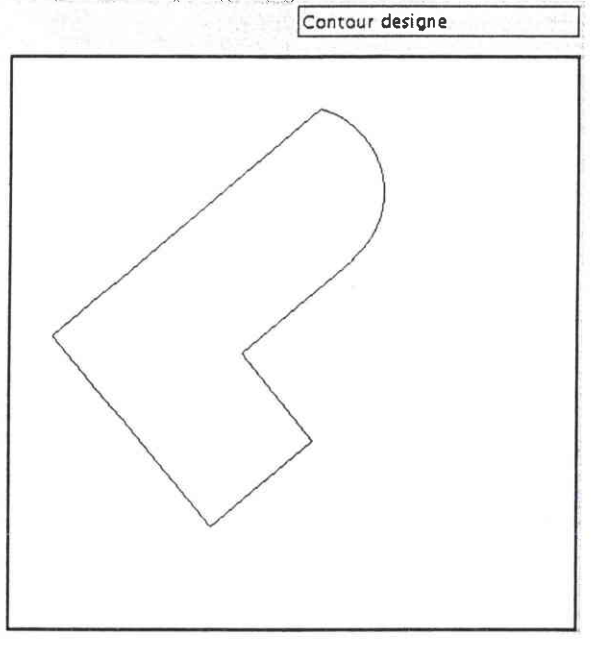
Cette fenêtre représente la scène en cours de réalisation, sur cette représentation figure toutes les contraintes issues du graphe normalisé de la scène. En cas de sauvegarde de la scène, c'est cette représentation qui est conservée.

- La ***fenêtre de travail*** :



C'est dans cette fenêtre que sont réalisées toutes les constructions, elle contient donc une représentation de la scène courante ainsi que des différentes contraintes définies par l'utilisateur du logiciel ou encore déduites par le système.

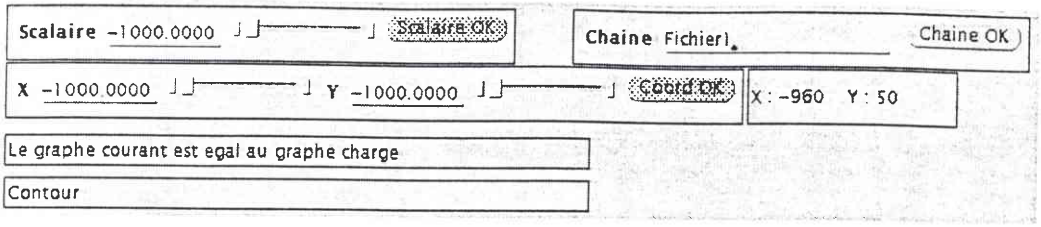
- La fenêtre ***Contour désigné*** :



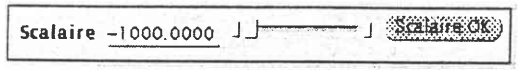
Cette fenêtre contient la représentation du contour définit à partir de la scène courante ; les éléments de ce contour sont donc désignés dans la fenêtre de travail.

2.2.1.3.Les zones de dialogue

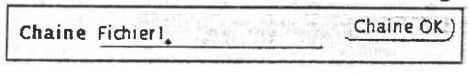
Différentes zones permettent à l'utilisateur de fournir des données au système ou encore donnent la possibilités au système d'informer l'utilisateur sur la situation actuelle.



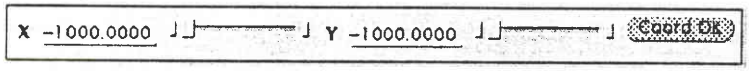
- la zone ***Scalaire*** permet de définir un scalaire qui peut représenter, par exemple, la valeur du rayon d'un cercle, cette valeur peut être donnée soit en utilisant le clavier soit à l'aide d'une jauge ;



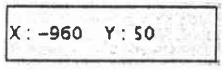
- la zone ***Chaîne*** précise une chaîne de caractères, elle permet notamment de définir le nom du fichier de sauvegarde ou encore celui du fichier à charger ;



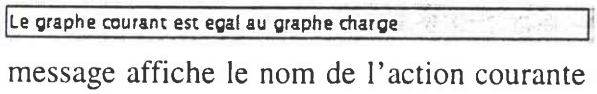
- la zone ***XY*** permet de définir soit au clavier soit à l'aide de la jauge les coordonnées précises d'un point donné ;



- la zone ci-dessous affiche en temps réel les coordonnées du point désigné par le curseur de la souris ;



- la première zone de message affiche les résultats de certains traitement notamment celui de la vérification de l'égalité ou de l'inclusion de deux graphes ;



- la deuxième zone de message affiche le nom de l'action courante c'est-à-dire le nom du dernier menu sélectionné.

2.2.2. *La comparaison de modèles*

2.2.2.1.L'égalité

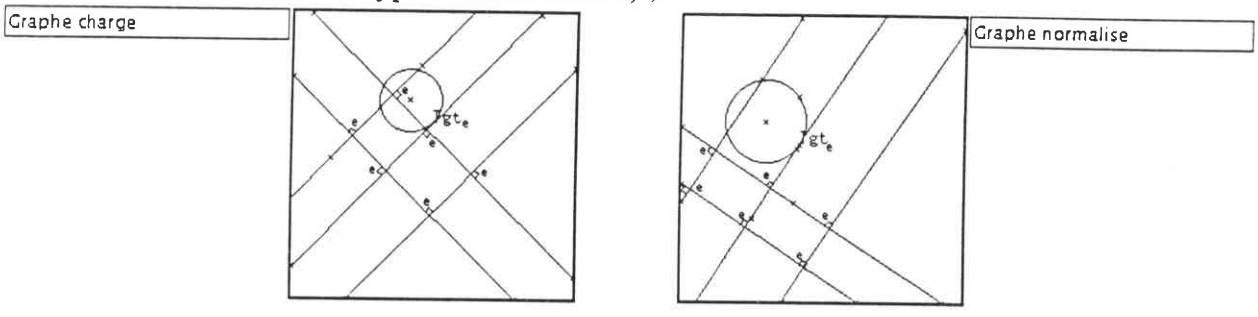
L'égalité de deux pièces est recherchée dans cette maquette entre le modèle normalisé de la pièce en cours de création et celui de la pièce chargée par l'utilisateur. Cette pièce peut être issue,

par exemple, d'une bibliothèque prédéfinie de pièces mécaniques ou encore avoir été conçue précédemment par un utilisateur du système.

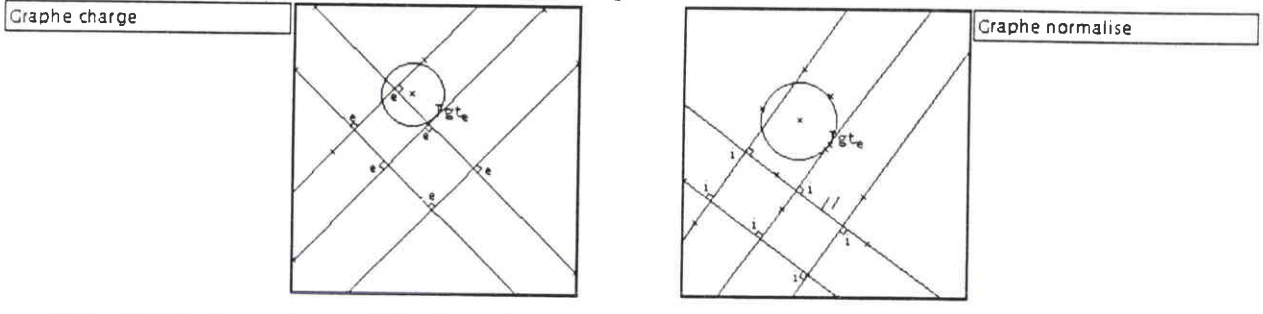
Les deux pièces comparées sont celles représentées dans les deux fenêtres graphiques appelées "Graphe chargé" et "Graphe normalisé". Dans ces deux fenêtres figurent non seulement les différents éléments (segments, cercles) composant les pièces mais aussi les différentes contraintes liant ces entités. On distingue deux types de contraintes, les contraintes explicites et les contraintes implicites, cette information est représentée respectivement par la lettre "e" et la lettre "i" à côté de la description classique de la contrainte (// pour parallèle, Tg pour tangent, H pour Horizontal, V pour Vertical, ⌐ pour perpendiculaire).

Ainsi les différentes égalités possibles pourront être vérifiées visuellement lorsque les pièces sont constituées d'un nombre raisonnable de contraintes et d'éléments :

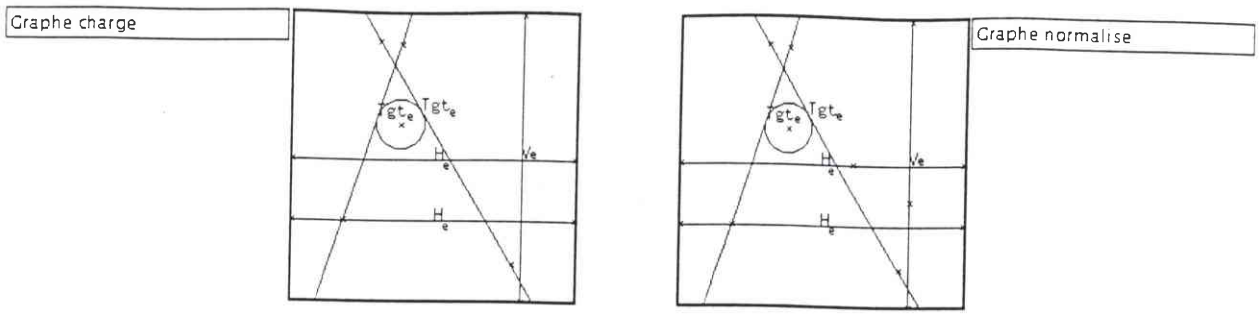
- l'***égalité stricte*** est obtenue lorsque exactement les mêmes contraintes se retrouvent dans les deux figures (même type et même nature) ;



- l'***égalité faible*** est obtenue lorsque les mêmes contraintes sont présentes dans les deux représentations des pièces (même type uniquement) ;



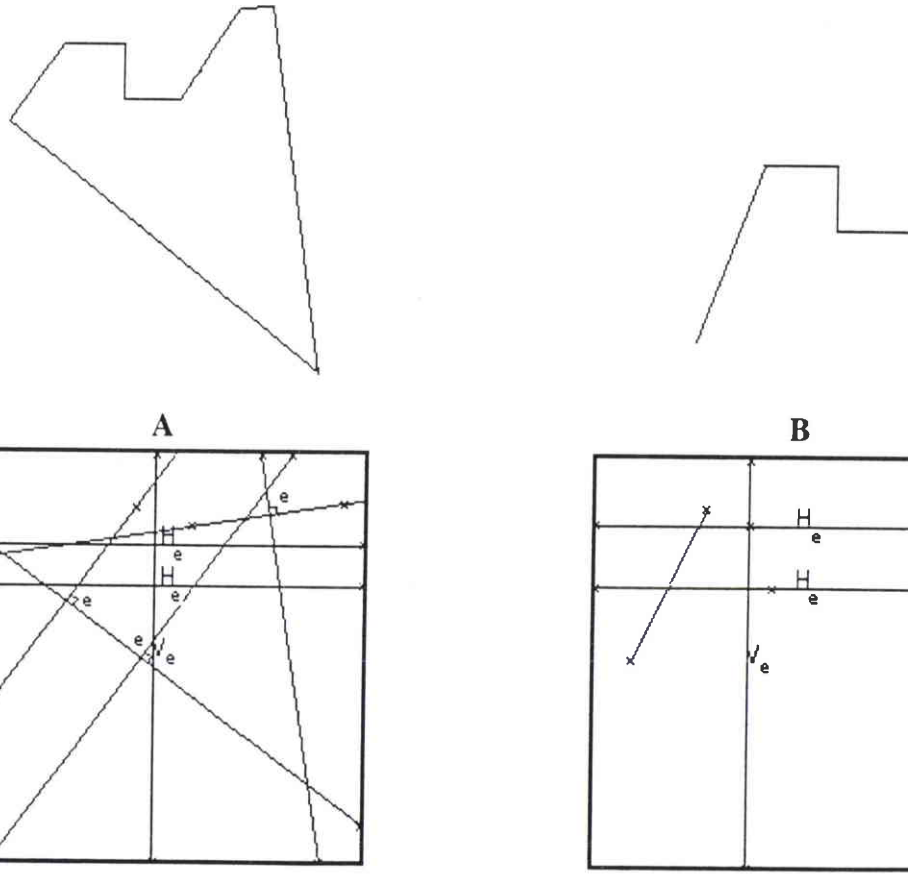
- l'***égalité forte*** est la combinaison d'une égalité stricte et de l'existence d'une composition de transformations affines permettant de transformer une pièce en l'autre.



2.2.2.2.L'inclusion

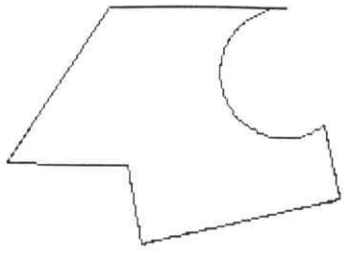
Les mêmes notations sont utilisées pour l'inclusion, on distingue seulement deux types d'inclusion, l'inclusion stricte et l'inclusion faible :

- ***inclusion stricte*** : la figure B est incluse dans la pièce A, en effet la représentation de B est incluse dans la représentation de A ou encore, le graphe de B est un sous-graphe de A ;

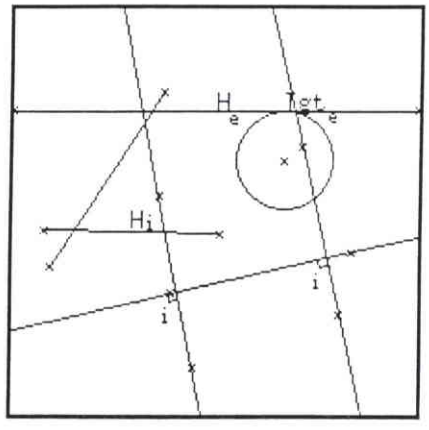


Représentation de A **Représentation de B**

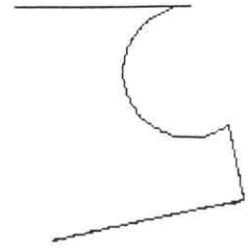
- ***inclusion faible***



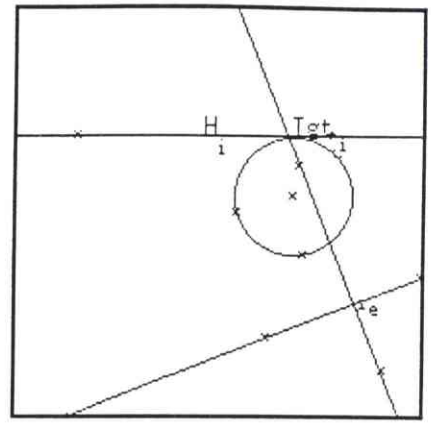
A



Représentation de A



B



Représentation de B

2.3. Normalisation de graphes

Le chapitre 2 a présenté les différentes règles permettant d'obtenir une représentation normalisée sous forme de graphes de toutes pièces. Nous présentons, dans cette annexe, pour différentes pièces (représentation dans la fenêtre Contour désigné), la représentation utilisateur (dans la fenêtre de travail) et la représentation normalisée (dans la fenêtre Graphe normalisé), les différences entre les représentations sont des conséquences du processus de normalisation.

Nous rappelons ci-dessous les différentes règles de normalisation de notre application ($\succ$ signifie "est remplacé par") :

R_1 $(D_1 // D_2) \wedge D_1$ Horizontal $\Rightarrow$ $(D_1 // D_2) \succ D_2$ Horizontal

R_2 $(D_1 // D_2) \wedge D_1$ Vertical $\Rightarrow$ $(D_1 // D_2) \succ D_2$ Vertical

R_3 $(D_1 \perp D_2) \wedge D_1$ Vertical $\Rightarrow$ $(D_1 \perp D_2) \succ D_2$ Horizontal

R_4 $(D_1 \perp D_2) \wedge D_1$ Horizontal $\Rightarrow$ $(D_1 \perp D_2) \succ D_2$ Vertical

R_5 $(D_1 // D_2) \wedge (D_1 \perp D_3)$ $\Rightarrow$ $(D_1 // D_2) \succ D_2 \perp D_3$

R_6 $(D_1 \perp D_2) \wedge (D_2 \perp D_3) \wedge (D_1 \perp D_4)$ $\Rightarrow$ $D_3 \perp D_4$

R_7 $(D_1 // D_2) \wedge (D_1 // D_3)$ $\Rightarrow$ $D_2 // D_3$

R_8 D_1 Dessus $\Rightarrow$ D_1 Tangent

R_9 D_1 Dessous $\Rightarrow$ D_1 Tangent

R_{10} D_1 Gauche $\Rightarrow$ D_1 Tangent

R_{11} D_1 Droite $\Rightarrow$ D_1 Tangent

R_{12} D_1 MêmeAbs $\Rightarrow$ D_1 Vertical

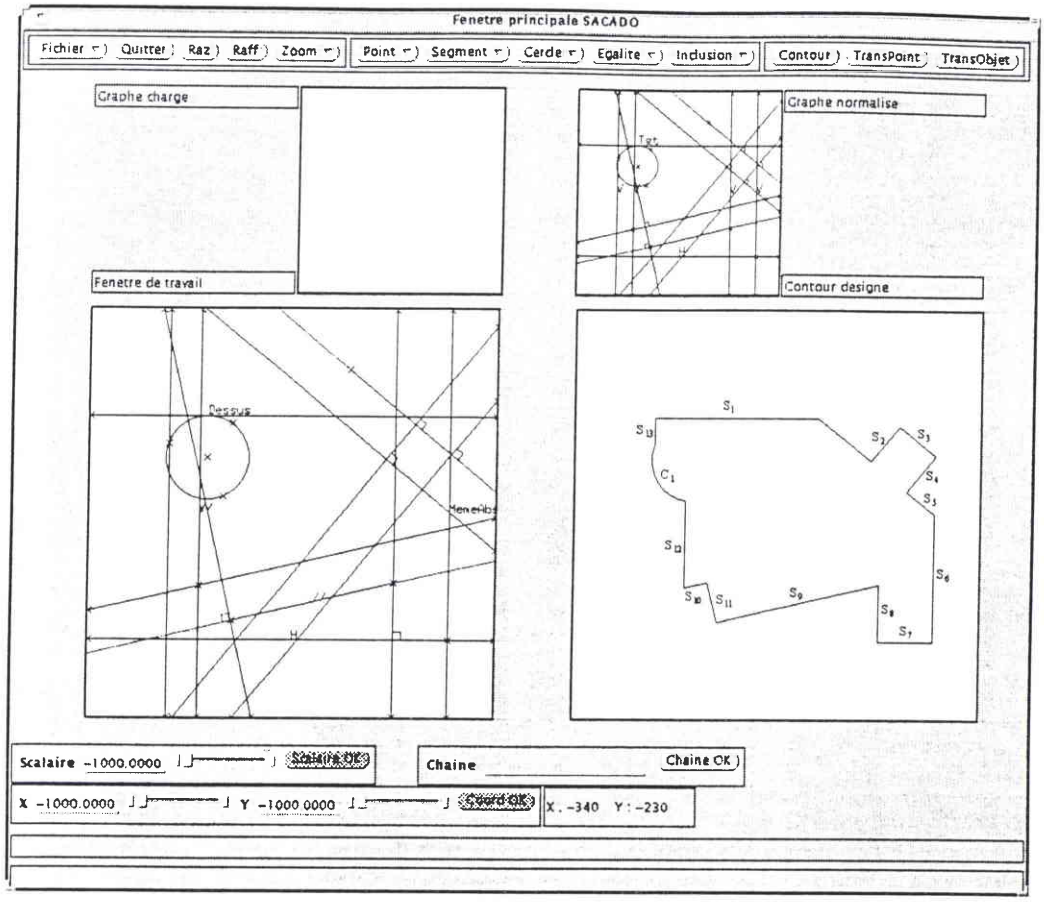
R_{13} D_1 MêmeOrd $\Rightarrow$ D_1 Vertical

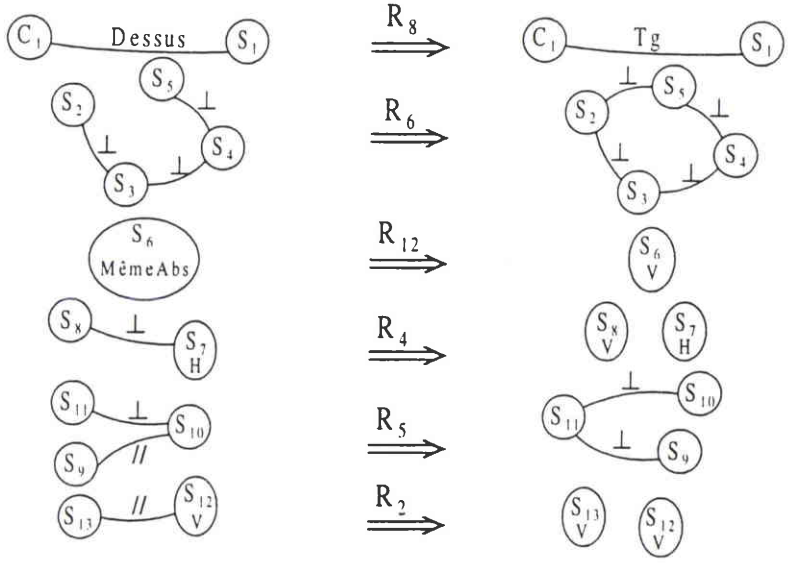
R_{14} $(D_1 //_i D_2) \wedge (D_1 \perp_e D_3)$ $\Rightarrow$ $(D_1 //_i D_2) \succ (D_3 \perp_i D_2)$

R_{15} $(D_1 //_e D_2) \wedge (D_1 \perp_i D_3)$ $\Rightarrow$ $(D_3 \perp_i D_2)$

2.3.1. Exemple 1 :

La pièce est construite en utilisant un seul type de contraintes (uniquement des contraintes explicites dans ce cas précis)



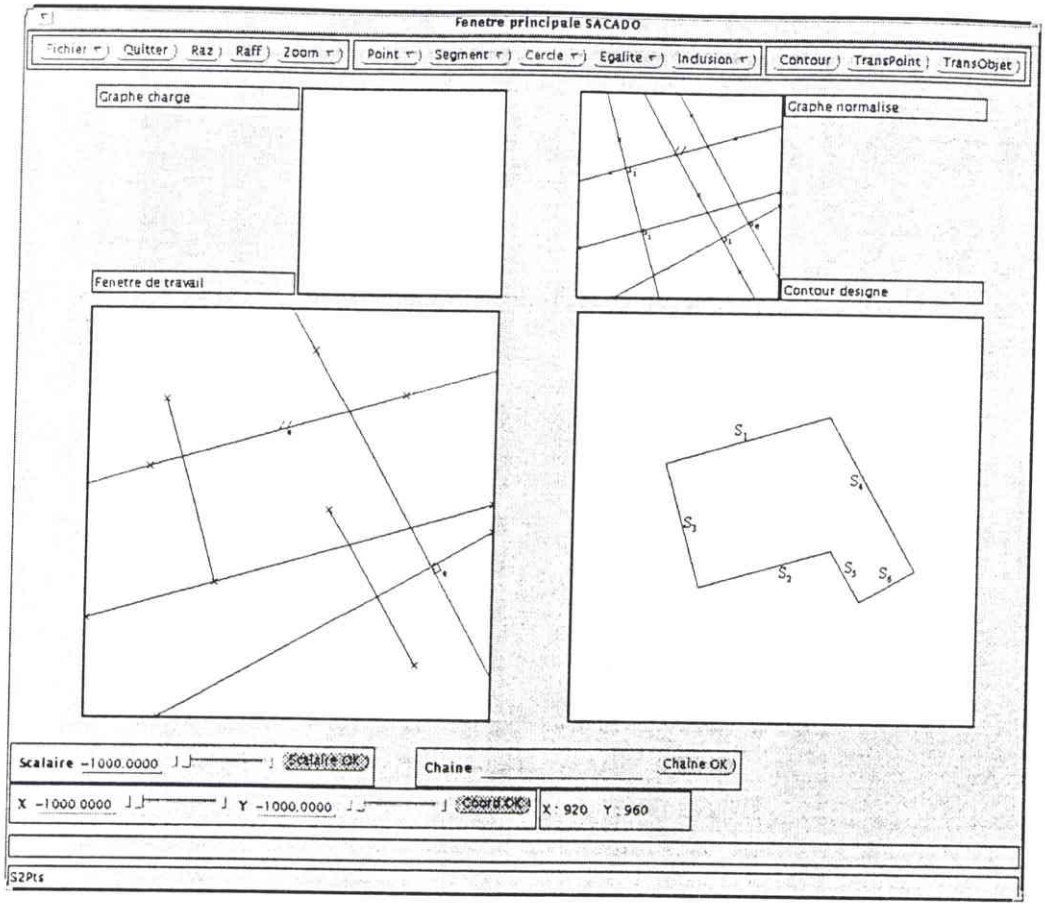


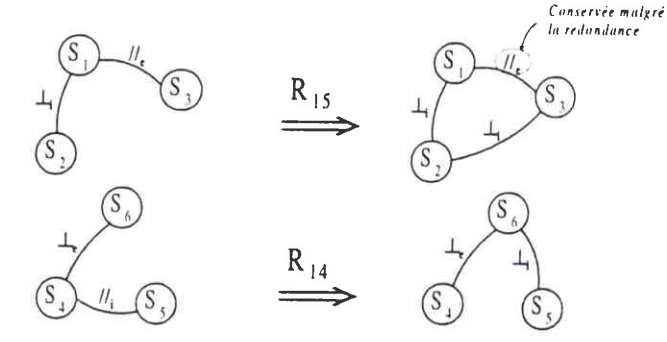
Graphe utilisateur

Graphe normalisé

2.3.2. Exemple 2 :

La pièce est créée en combinant des contraintes explicites et des contraintes implicites, les règles mixtes de normalisation sont utilisées.





Graphe utilisateur Graphe normalisé

3. ANNEXE C : MODÉLISATION ET GESTION DES CONTRAINTES FONCTIONNELLES PAR LES SYSTÈMES DE CAO

3.1. Présentation de la maquette

3.1.1. Présentation générale de la maquette

Lors de la réalisation de cette maquette, nous nous sommes surtout consacrés à l'implémentation des différents algorithmes et données permettant de résoudre le problème d'aménagement spatial fonctionnel d'une cuisine.

Les différentes classes mises en œuvre sont représentées ainsi que les relations pouvant les lier dans le diagramme des classes ci-dessous (le même formalisme est utilisé que dans les annexes précédentes).

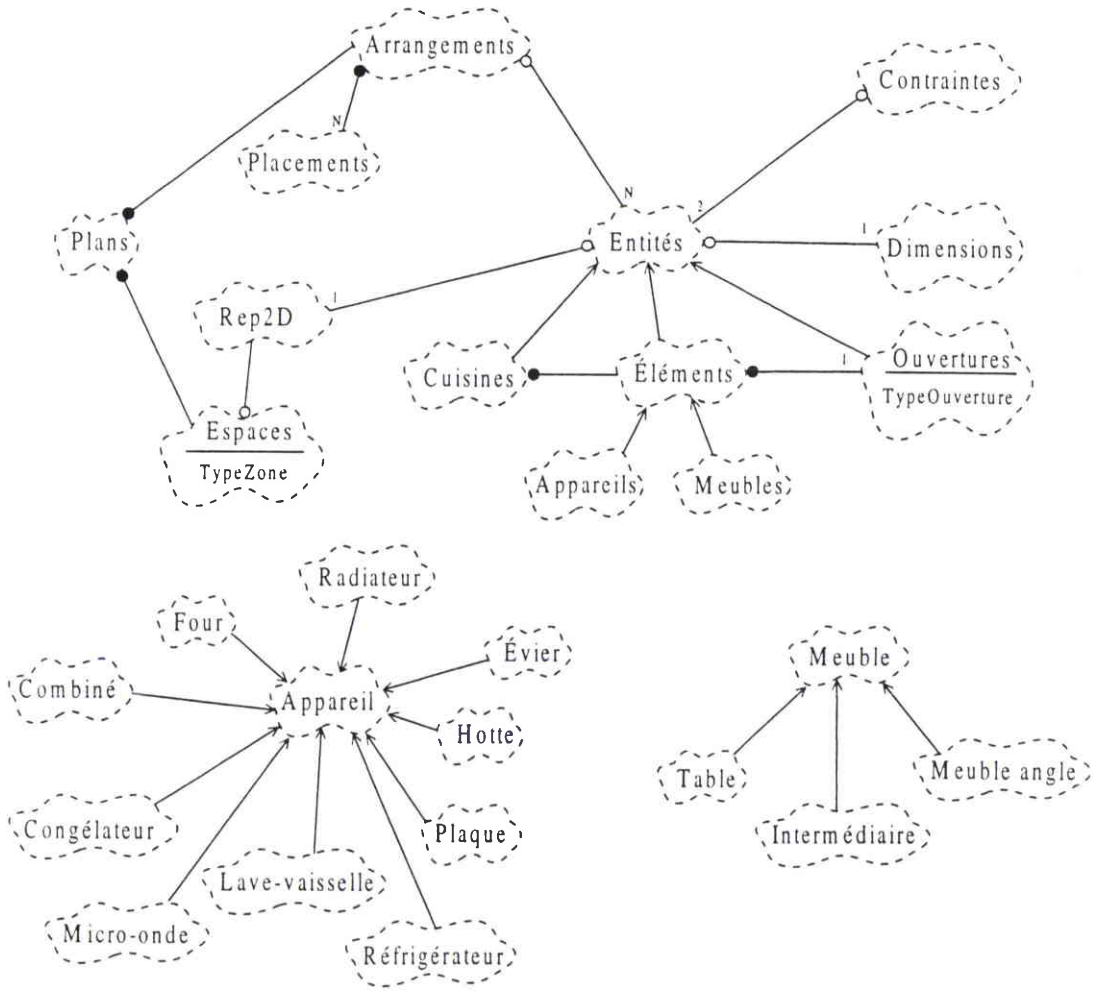


Figure C.1. *Diagramme des classes*

Nous précisons ci-dessous les paramètres et l'utilisation de certaines classes :
classe ***Entités*** : c'est la classe parent des classes ***Cuisines***, ***Éléments*** et ***Ouvertures***, elle regroupe les caractéristiques communes des ces entités c'est-à-dire les dimensions (longueur, largeur et éventuellement profondeur) et la représentation en deux dimensions (Rep2D) qui est composée d'une union de rectangles.

classe ***Éléments*** : elle centralise les informations partagées par les différents éléments comme par exemples les ouvertures qui peuvent avoir différents types (Dessus, Face simple, Face double, ...). Cette classe a deux classes dérivées : ***Appareils*** et ***Meubles***.

classe ***Appareils*** : elle possède de nombreuses classes dérivées correspondant aux différents types d'appareils.

classe ***Meubles*** : elle a plusieurs classes filles représentant les différents types de meubles.

Un plan représentant un aménagement complet de cuisine est composé de deux données :
- un arrangement regroupant tous les placements de meuble représentés par les coordonnées du point de référence de chaque entité dans le repère associé à la cuisine ;
- une représentation de l'espace sousforme d'une union de rectangles de différents types correspondant à des zones libres, occupées ou de passage.

3.1.2. Présentation de l'algorithme de placement

Ce processus peut être décomposé grossièrement en trois phases :
- le placement des meubles d'angles ;
- le placement des meubles intermédiaires ;
- le placement de la table.

Nous ne détaillons pas ici ces différents algorithmes, nous présentons simplement un exemple de diagramme d'interaction des différentes classes sollicitées s'appuyant sur le sous-programme réalisant le placement des meubles d'angles. Sur la même ligne, figure d'une part sous forme d'algorithme les différentes instructions permettant de résoudre le problème et d'autre par les différentes requêtes liant les différents classes concernées.

Algorithme réalisant le placement des meubles d'angle

Début

Isoler tous les meubles d'angles de la cuisine

Pour tous les meubles d'angles faire
Consulter la longueur et la largeur du meuble considéré

Consulter les Rep2D correspondants à des coins libres de l'espace de placement

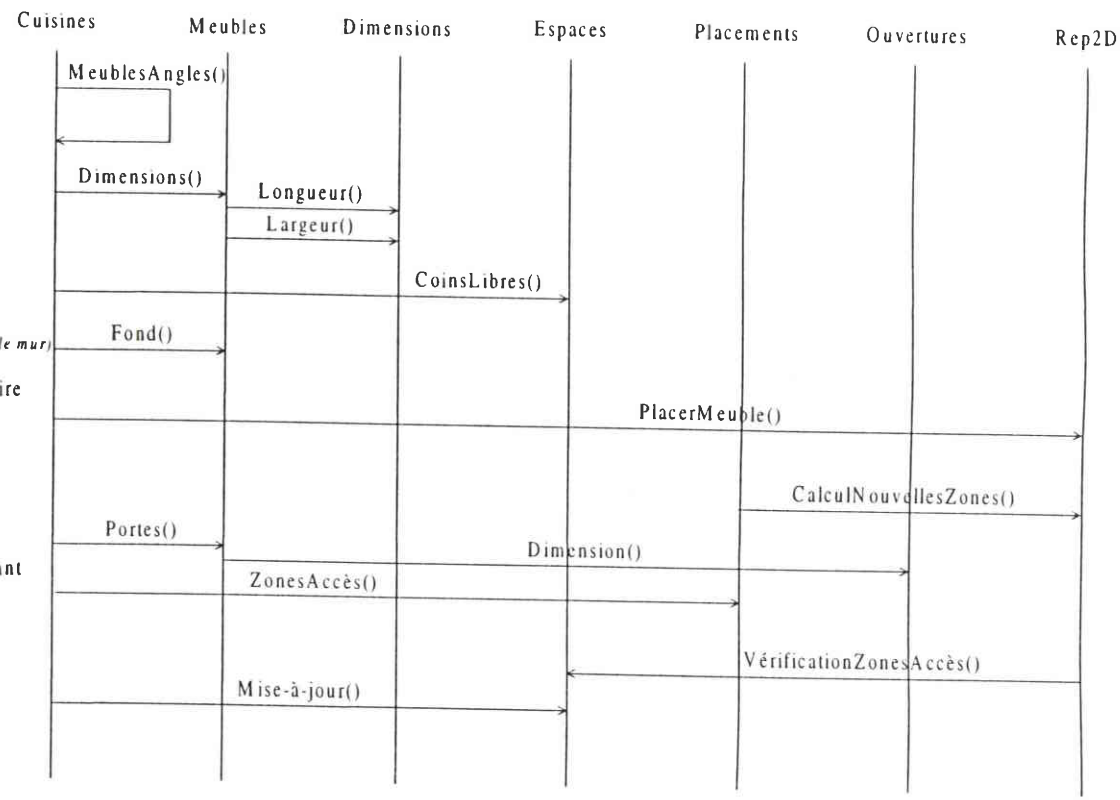
Consulter le Fond du meuble *(en largeur ou en longueur contre le mur)*

Pour toutes les Rep2D correspondant à des coins libres faire
Essayer de définir un placement de l'objet

Si un placement est possible alors
Calculer les nouvelles zones (libres, occupées)
FinSi
FinPour
À partir du placement trouvé et des informations concernant le meuble, calcul de la zone d'accès

Vérification de la zone d'accès par rapport à l'espace

Mise à jour de la Rep2D de l'espace
FinPour
Fin

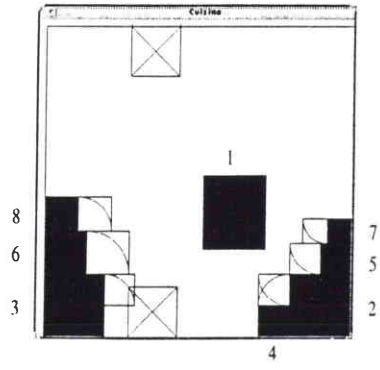


3.2. Exemples de résultats obtenus

Dans cette partie, nous présentons différents exemples de cuisine obtenus avec la maquette réalisée et implantant les différents algorithmes et classes présentés soit au chapitre 3 de ce document soit en annexe A.

3.2.1. Exemple 1

Dans cet exemple, les différents meubles ne sont pas contraints et deux ouvertures (portes d'accès) sont définies. Parmi les huit meubles à placer, il y a deux meubles d'angles et une table, les autres sont des meubles intermédiaires sans particularité.

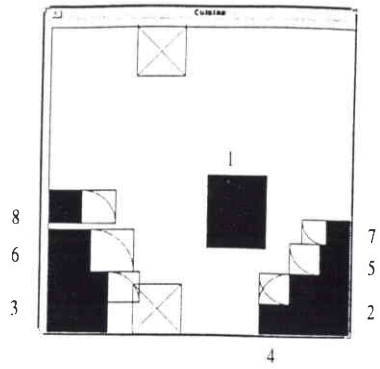


3.2.2. Exemple 2

Dans cet exemple, les meubles à placer sont les mêmes que ceux de l'exemple ci-dessus mais des contraintes de positionnement précisent les positions de certains meubles par rapport à d'autres.

Les contraintes imposées sont les suivantes :

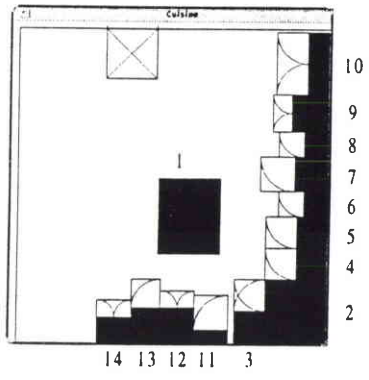
- l'élément 4 doit être à une distance de 0 de l'élément 2 ;
- l'élément 5 doit être à une distance de 0 de l'élément 2 ;
- l'élément 6 doit être à une distance de 0 de l'élément 3 ;
- l'élément 7 doit être à une distance de 0 de l'élément 5 ;
- l'élément 8 doit être à une distance de 10 de l'élément 6 ;



Remarque : On peut constater sur les exemples 1 et 2 que les résultats obtenus sont identiques lorsque le système décide du placement de tous les meubles (exemple 1) ou lorsque l'utilisateur contraint tous les meubles de sorte qu'il définit précisément la position de chacun d'eux (exemple 2).

3.2.3. Exemple 3

Dans cette configuration, il y a une ouverture vers l'extérieur, une table et quatorze éléments à positionner dont seulement un avec un placement contraint (11) et un meuble d'angle (2).



Remarque : Notre algorithme de placement complète en priorité un pan de mur puis le suivant. En effet, il place tout nouveau meuble, de préférence à côté d'un meuble déjà placé.

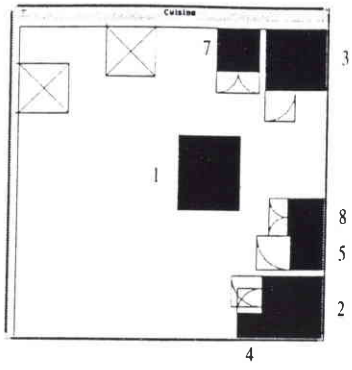
3.2.4. Exemple 4

Dans cette configuration, nous montrons que notre système détecte les problèmes sur-contraint et il signale ainsi à l'utilisateur tout placement impossible d'élément.

Configuration de cet exemple :

- 2 ouvertures ;
- 1 table ;
- 2 meubles d'angles (2,3) ;

- 6 meubles intermédiaires :
  - l'élément 4 doit être à une distance de 0 de l'élément 2 ;
  - l'élément 5 doit être à une distance de 10 de l'élément 2 ;
  - l'élément 6 doit être à une distance de 30 de l'élément 2 ;
  - l'élément 7 doit être à une distance de 10 de l'élément 3 ;

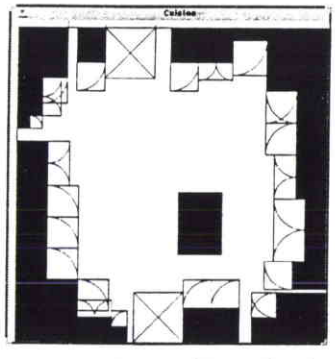


Remarque : L'élément 6 ne peut pas être placé à côté de l'élément 2 car deux éléments sont déjà placés de part et d'autre de l'élément 2.

3.2.5. Exemple 5

Cet exemple montre un encombrement maximal de la cuisine, la configuration est la suivante :

- 2 ouvertures ;
- 1 table ;
- 4 meubles d'angle ;
- 21 meubles intermédiaires (non contraint) ;



Remarque : L'avant-dernier meuble ne peut pas être placé car il n'existe pas d'espace non occupé, sur un mur suffisamment grand pour placer ce meuble. Pae contre, le dernier meuble, plus petit que l'avant-dernier peut être placé.

Dans tous ces exemples, on remarque que dans la mesure du possible, la table est placée à la fois, près du centre de la pièce mais aussi près des meubles déjà positionnés.

UNE NOUVELLE APPROCHE POUR LA MODÉLISATION ET LA GESTION DES CONTRAINTES EN CAO

Le savoir-faire des logiciels de CAO (Conception Assistée par Ordinateur) ainsi que les techniques d'IA (Intelligence Artificielle) sont souvent associés à la résolution de problèmes dans des domaines nombreux et variés. De nos jours, il est demandé à tout logiciel de CAO de fournir à l'utilisateur une réelle assistance et d'être capable de réaliser des raisonnements.

Nous présentons les principales fonctionnalités requises par les systèmes de CAO actuels ainsi que les méthodes de mise en œuvre possibles. La présentation de problèmes contraints dans des domaines variés montre que leur résolution peut se faire par des logiciels de CAO. On évoquera en particulier les problèmes de conception de pièces mécaniques, d'ordonnancement de tâches ou encore d'aménagement spatial.

Tout d'abord, nous nous intéressons à la gestion des contraintes géométriques et topologiques lors de la conception de pièces mécaniques et plus particulièrement à l'étude de la comparaison de modèles de CAO par normalisation de graphes. Cette comparaison permet lors de la conception, la réutilisation en totalité ou en partie de pièces existantes et issues de bibliothèques. Deux méthodes (l'une algorithmique, l'autre utilisant le formalisme CSP) sont présentées et permettent la comparaison de modèles de CAO par normalisation de graphes.

Nous nous intéressons ensuite à un type plus particulier de contraintes : les contraintes fonctionnelles. Ces dernières sont abordées dans un cadre très vaste : la conception d'une usine. L'étude conceptuelle de ce problème permet ainsi de montrer la grande expressivité de ces contraintes fonctionnelles. Un problème particulier d'aménagement spatial est considéré : la conception de cuisines intégrées. Les méthodes de modélisation mises en œuvre sont ensuite présentées ainsi que deux méthodes de résolution.

Mots-clés : *normalisation de graphes, modélisation, aménagement spatial, CAO, modélisation, CSP.*

UNE NOUVELLE APPROCHE POUR LA MODÉLISATION ET LA GESTION DES CONTRAINTES EN CAO

Le savoir-faire des logiciels de CAO (Conception Assistée par Ordinateur) ainsi que les techniques d'IA (Intelligence Artificielle) sont souvent associés à la résolution de problèmes dans des domaines nombreux et variés. De nos jours, il est demandé à tout logiciel de CAO de fournir à l'utilisateur une réelle assistance et d'être capable de réaliser des raisonnements.

Nous présentons les principales fonctionnalités requises par les systèmes de CAO actuels ainsi que les méthodes de mise en œuvre possibles. La présentation de problèmes contraints dans des domaines variés montre que leur résolution peut se faire par des logiciels de CAO. On évoquera en particulier les problèmes de conception de pièces mécaniques, d'ordonnancement de tâches ou encore d'aménagement spatial.

Tout d'abord, nous nous intéressons à la gestion des contraintes géométriques et topologiques lors de la conception de pièces mécaniques et plus particulièrement à l'étude de la comparaison de modèles de CAO par normalisation de graphes. Cette comparaison permet lors de la conception, la réutilisation en totalité ou en partie de pièces existantes et issues de bibliothèques. Deux méthodes (l'une algorithmique, l'autre utilisant le formalisme CSP) sont présentées et permettent la comparaison de modèles de CAO par normalisation de graphes.

Nous nous intéressons ensuite à un type plus particulier de contraintes : les contraintes fonctionnelles. Ces dernières sont abordées dans un cadre très vaste : la conception d'une usine. L'étude conceptuelle de ce problème permet ainsi de montrer la grande expressivité de ces contraintes fonctionnelles. Un problème particulier d'aménagement spatial est considéré : la conception de cuisines intégrées. Les méthodes de modélisation mises en œuvre sont ensuite présentées ainsi que deux méthodes de résolution.

Mots-clés : *normalisation de graphes, modélisation, aménagement spatial, CAO, modélisation, CSP.*