Titre: Title:	Programmation logique d'un algorithme de routage par hiérarchie d'abstraction et raffinements successifs s'appliquant aux circuits imprimés
Auteur: Author:	Christian Langheit
Date:	1989
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Langheit, C. (1989). Programmation logique d'un algorithme de routage par hiérarchie d'abstraction et raffinements successifs s'appliquant aux circuits imprimés [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/58245/

UNIVERSITÉ DE MONTRÉAL

PROGRAMMATION LOGIQUE D'UN ALGORITHME DE ROUTAGE PAR HIÉRARCHIE D'ABSTRACTION ET RAFFINEMENTS SUCCESSIFS S'APPLIQUANT AUX CIRCUITS IMPRIMÉS

par

Christian LANGHEIT
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU GRADE DE MAITRE ES SCIENCES APPLIQUÉES (M.Sc.A.)
Septembre 1989

ISBN 0-315-58184-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Ce mémoire intitulé:

PROGRAMMATION LOGIQUE D'UN ALGORITHME DE ROUTAGE
PAR HIÉRARCHIE D'ABSTRACTION ET RAFFINEMENTS SUCCESSIFS
S'APPLIQUANT AUX CIRCUITS IMPRIMÉS

présenté par: Christian Langheit

en vue de l'obtention du grade de: Maître es sciences appliquées

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

M. Bernard Lanctôt, B.Sc.A. et Ing., président

Mme Bozena Kaminska, Ph.D.

M. Jean-Charles Bernard, Ph.D.

à Dominique

et mes parents

SOMMAIRE

Ce projet s'inscrit dans le cadre de recherches visant l'amélioration d'outils de conception de circuits imprimés assistée par ordinateur. On propose un environnement de routage idéal qui posséderait une flexibilité et des capacités semblables à celles d'un expert humain. Cet environnement est caractérisé par une structure hiérarchique avec une hybridation de structures itératives. Plusieurs algorithmes de routage existants pourront et devraient se greffer à la structure hiérarchique. Un module de conception physique, s'appuyant sur les bases d'un système expert, contrôle le cheminement des tâches entre les différents niveaux de la hiérarchie.

Un prototype de la hiérarchisation ne comportant que deux niveaux est construit. Il vise principalement la mise en évidence des difficultés de hiérarchisation d'un algorithme de routage existant. L'algorithme de Lee [1] est modifié et adapté à la hiérarchisation. Cela permet de conclure que cette étape est réalisable.

Cette implantation permet d'identifier les problèmes de hiérarchisation et propose des voies de solutions sans toutefois déterminer la solution idéale. Elle permet également d'apprécier les avantages d'un langage de programmation logique, comme Prolog qui offre un excellent support au prototypage rapide.

ABSTRACT

This research is related to computer assisted design tools for printed circuits. We describe and propose the architecture of an ideal routing system with properties, in terms of flexibility and capability, quite similar to those usually expected from a human expert. The system is caracterized by a hierarchical structure based on multiple recursivity levels. Many existing routing algorithms can be locked on this hierarchy. A physical design module, using expert system techniques, controls the progression of the tasks at each different level of the hierarchy.

A first prototype of the hierarchical structure, restricted to two levels, is developped. Its principal objective is to show the inherent difficulties of adapting an existing routing algorithm to this approach. The Lee algorithm is modified and used in the proposed system. For this purpose, a graphical alphabet has been developped and is presented. Basic graphic objects allows a 3-dimensional search for path on a given coarse grid. The feasability of this approach is thus demonstrated.

Implementation of our restricted prototype has permitted the identification of problems related to the use of an hierarchical approach. Avenues of solution are generated without, however, finding the optimal solution. Merit is also given to Prolog as an excellent rapid prototyping language.

REMERCIEMENTS

J'aimerais remercier mon directeur, M. Jean-Charles Bernard, pour son appui qu'il a manifesté envers mon projet et son encouragement continuel.

Je remercie sincèrement les membres du laboratoire SCRIBENS qui m'ont motivé et stimulé constamment jusqu'aux dernières heures de ce projet.

Enfin, je remercie également Mme Martine Pinsonneault, sans qui ce rapport n'aurait pu prendre une si belle forme et ce, en si peu de temps.

TABLE DES MATIÈRES

SOMMAIRE v

ABSTRACT vii

REMERCIEMENTS ix

INTRODUCTION 1

LE RÉALISATION DE CIRCUITS IMPRIMÉS 2

OBJECTIFS SPÉCIFIQUES DU PROJET 5

CHAPITRE 1 PROBLÉMATIQUE 7

1.1 LE PROBLÈME 7

1.2 LES ROUTEURS 12

CHAPITRE 2 ARCHITECTURE HIÉRARCHIQUE 16

2.1 TYPES DE ROUTEURS 16

2.1 ALGORITHMES DE LA FAMILLE DE LEE 20

2.3 ROUTEUR SANS GRILLE 22

2.4	ROUTEUR PAR CANAL	23
2.5	ROUTEUR PAR ÉCHANGEUR	25
2.6	ROUTEUR DE LIGNES	26
2.7	ROUTEUR PAR SYSTÈME EXPERT	28
2.8	ROUTEUR SÉQUENTIEL	30
2.9	ROUTEUR ITÉRATIF	30
2.10	ROUTEUR HIÉRARCHIQUE	31
2.11	ROUTEUR PAR HIÉRARCHIE D'ABSTRACTIONS SUCCESSIVES	33
2.11.1	HIÉRARCHIE	35
2.11.2	SYSTÈME EXPERT	37
2.11.3	BASE DE CONNAISSANCE	38
2.11.4	NIVEAU PLACEMENT	44
2.11.5	NIVEAU ROUTAGE GLOBAL À GROS GRAIN	47
2.11.6	NIVEAU ROUTAGE GLOBALI-LOCAL	48
2.11.7	NIVEAU ROUTAGE LOCAL	49
2.11.8	NIVEAU ÉLIMINATION DE BLOCAGE LOCAL	50
2.12	IMPLANTATION	50
2.13	MOTIVATION DE LA HIÉRARCHISATION	51
CHAPITRE 3	ENVIRONNEMENT DE TRAVAIL	56
3.1	CHOIX DE L'ENVIRONNEMENT DE TRAVAIL	56
3.2	COMPILATION DE PROGRAMME PROLOG	59
3.3	ÉDITEUR SCHÉMATIQUE	61
3.4	HIÉRARCHIE DE RECHERCHE	63

CHAPITRE 4 ROUTEUR GLOBAL À GRILLE LARGE 65

4.1 ROUTEUR DE LEE 65
4.2 LE NOUVEAU ROUTEUR GLOBAL 67
4.3 GRAPHÈMES DE RECHERCHE 69
4.4 STRUCTURE DE DÉPART 73
4.5 PRÉPARATION DES DONNÉES DE ROUTAGE 75
4.6 MODE D'EXPANSION 79
4.7 CONTRAINTES ADDITIONNELLES 83
4.8 DÉCISION DU CHEMIN GROSSIER 83
4.9 FORME DES RÉSULTATS 85
4.10 AVANTAGES ET INCONVÉNIENTS DE PROLOG 87

CHAPITRE 5 PROCESSUS DE MÛRISSEMENT 90

5.1 DESCRIPTION DES STRUCTURES DE DÉPART 91
5.2 RECHERCHE DES RÉGIONS DE BLOCAGE 92
5.3 DÉCISION DES CHEMINS À ENLEVER 95
5.4 FORME DES RÉSULTATS ET DE LA COMMANDE GÉNÉRÉE 99

CHAPITRE 6 ÉVALUATION DES RÉSULTATS 101

6.1 EXEMPLE 1 102
6.2 EXEMPLE 2 104

6.3 EXEMPLE 3 105

6.4 EXEMPLE 4 106

6.5 EXEMPLE 5 108

6.6 EXEMPLE 6 109

6.7 EXEMPLE 7 110

6.8 EXEMPLE 8 111

6.9 EXEMPLE 9 115

CHAPITRE 7 CONCLUSION ET VOIES DE RECHERCHE FUTURES 119

7.1 PROBLÈMES RELATIFS AU ROUTEUR GLOBAL À GROSSES MAILLES 119

7.2 PROBLÈMES RELATIFS AU ROUTEUR LOCAL 122

7.3 PROBLÈMES LIÉs À L'ENVIRONNEMENT 123

7.4 VOIES DE RECHERCHE FUTURES 124

BIBLIOGRAPHIE 131

ANNEXE 1 135

A1.1 FAITS DE L'EXEMPLE 1 135

A1.2 FAITS DE L'EXEMPLE 2 136

A1.3 FAITS DE L'EXEMPLE 3 137

A1.4 FAITS DE L'EXEMPLE 4 138

A1.5 FAITS DE L'EXEMPLE 5 140

A1.6 FAITS DE L'EXEMPLE 6 141

A1.7 FAITS DE L'EXEMPLE 7 141

A1.8 FAITS DE L'EXEMPLE 8 143

A1.9 FAITS DE L'EXEMPLE 9 144

ANNEXE 2 148

LEXIQUE 152

LISTE DES FIGURES

Figure		
1.1	Éléments de base d'un circuit imprimé	9
1.2	Distance de Manhattan	15
2.1	Modes de fonctionnement des routeurs	17
2.2	Algorithmes de routage	19
2.3	Fonctionnement d'un algorithme de la famille de Lee	21
2.4	Fonctionnement d'un algorithme par expansion de régions	23
2.5	Résultat d'un routeur par canal	24
2.6	Résultat partiel d'un routeur par échangeur	26
2.7	Résultat d'un routeur de lignes	27
2.8	Représentation de certaines règles de simplification d'un routeur par système expert	29
2.9	Hiérarchie du système	36
2.10	Schéma global du système	38
2.11	Séparation de la base de connaissance	40
2.12	Connaissances des familles de composantes sous forme d'objets	41
2.13	Aire d'une région d'expansion	52

2.14	Courbes du nombre de voisins visités pendant une expansion	54
4.1	Fonctionnement d'un algorithme de la famille de Lee	66
4.2	Graphèmes de base et chemin représenté par une suite de graphèmes	70
4.3	Espace occupée par le graphème COIN	71
4.4	Masque d'illégalité	77
4.5	Grosse case et masque d'illégalité	78
4.6	Visite de cases	82
4.7	Décision du chemin	85
5.1	Expansion après un blocage	93
6.1	Visualisation de l'exemple 1	102
6.2	Premier résultat du routeur hiérarchique pour l'exemple 1	102
6.3	Deuxième résultat du routeur hiérarchique pour l'exemple 1	103
6.4	Résultat de l'expert humain pour l'exemple 1	103
6.5	Visualisation de l'exemple 2	104
6.6	Premier résultat du routeur hiérarchique pour l'exemple 2	104
6.7	Deuxième résultat du routeur hiérarchique pour l'exemple 2	104

6.8	Résultat de l'expert humain pour l'exemple 2	104
6.9	Visualisation de l'exemple 3	105
6.10	Résultat du routeur hiérarchique pour l'exemple 3	105
6.11	Résultat de l'expert humain pour l'exemple 3	106
6.12	Visualisation de l'exemple 4	106
6.13	Premier résultat du routeur hiérarchique pour l'exemple 4	106
6.14	Deuxième résultat du routeur hiérarchique pour l'exemple 4	107
6.15	Résultat de l'expert humain pour l'exemple 4	107
6.16	Visualisation de l'exemple 5	108
6.17	Résultat du routeur hiérarchique pour l'exemple 5	108
6.18	Résultat de l'expert humain pour l'exemple 5	108
6.19	Visualisation de l'exemple 6	109
6.20	Résultat du routeur hiérarchique pour l'exemple 6	110
6.21	Résultat de l'expert humain pour l'exemple 6	110
6.22	Visualisation de l'exemple 7	110
6.23	Résultat du routeur hiérarchique pour l'exemple 7	111
6.24	Résultat de l'expert humain pour l'exemple 7	111
6.25	Visualisation de l'exemple 8	111
6.26	Premier résultat du routeur hiérarchique pour l'exemple 8	112

6.27	Deuxième résultat du routeur hiérarchique pour l'exemple 8	113
6.28	Arrangement idéale de traverses	113
6.29	Résultat de l'expert humain pour l'exemple 8	114
6.30	Visualisation de l'exemple 9	115
6.31	Résultat de l'expert humain pour l'exemple 9	116
6.32	Résultat du routeur commercial pour l'exemple 9	116
6.33	Résultat du routeur hiérarchique pour l'exemple 9	117
7.1	Incohérence de chemins	120
7.2	Résultat de l'algorithme du "Minimum Spanning Tree"	126
A.1	Routes de la figure 6.29 reposant sur la couche A	149
A.2	Routes de la figure 6.29 reposant sur la couche B	149
A.3	Routes de la figure 6.31 reposant sur la couche A	150
A.4	Routes de la figure 6.31 reposant sur la couche A	150
A.5	Routes de la figure 6.32 reposant sur la couche A	151
A.6	Routes de la figure 6.32 reposant sur la couche A	151

INTRODUCTION

Les développements accélérés dans les domaines de la conception et de la fabrication de produits électroniques engendrent une complexité croissante des tâches à exécuter. Les procédés de fabrication de composantes électroniques permettent des densités d'intégration qui augmentent constamment. Avec cette croissance continuelle de la densité, les composantes électroniques deviennent de plus en plus performantes, rapides mais surtout complexes. Cette complexité se réflète aussi à un niveau plus élevé. Les ordinateurs et les gadgets électroniques qui font un usage intensif de ces composantes électroniques proposent inlassablement des possibilités d'application plus phénoménales les unes que les autres.

Même si, de nos jours, chaque composante électronique permet de faire un grand nombre de tâches, il faudra toujours les interconnecter avec d'autres composantes afin d'obtenir un produit utile à un usager. Il existe principalement trois méthodes distinctes de connexion entre les composantes:

.le prototypage par insertion

.le prototypage par l'enroulement

.le circuit imprimé.

La méthode qui nous intéresse utilise le circuit imprimé. C'est la seule méthode qui permet la production de masse de circuits électroniques. Le premier circuit imprimé demande habituellement beaucoup de travail de conception. Néanmoins, une fois réalisé il est reproduisible automatiquement à une vitesse impressionnante et avec un minimum d'effort.

LA RÉALISATION DE CIRCUITS IMPRIMÉS

La réalisation de circuits imprimés se divise en trois étapes distinctes:

.la conception d'un circuit électronique

.la conception du circuit imprimé

.la fabrication du circuit imprimé.

L'étape dont fait l'objet notre recherche est la conception du circuit imprimé. Cette étape est habituellement réalisée par un expert humain à l'aide d'un programme de dessin spécialisé ou par un programme

automatique de conception de circuit imprimé. L'étape de conception est souvent appelée l'étape de placement et de routage. Pendant cette étape, se décide l'emplacement de chaque composante sur une plaque et se construisent des routes distinctes qui génèrent les connexions électriques entre les composantes.

Cette même étape de placement et de routage se retrouve aussi pendant la conception de circuits électronique intégrés. Depuis plusieurs années, les efforts se sont accrus dans la recherche visant le développement et l'amélioration des techniques de conception de circuits intégrés. De plus, des recherches importantes se sont attardées aux programmes automatiques de placement et de routage spécialisés pour les circuits intégrés en laissant de coté ceux spécialisés aux circuits imprimés. Serait-ce que les programmes de placement/routage de circuits imprimés sont assez matures pour résoudre tous les problèmes de conception de circuits imprimés d'une manière égale ou supérieure aux résultats d'un expert humain? Au contraire, l'expert humain est toujours supérieur par ses performances aux programmes automatiques.

Comment expliquer la diminution de l'intérêt pour ce domaine de recherche? Nous devons avouer que la conception et la réalisation de circuit à très grande échelle d'intégration (VLSI) sont des domaines à la "mode" et ce pour plusieurs raisons:

- C'est un domaine jeune et en évolution constante.
- Il y a un besoin marqué de l'industrie qui veut rester concurrentielle.
- Il se crée un intérêt grandissant au sein de la communauté scientifique.
- Il y a une injection massive de subventions et de capitaux dans ce domaine.

La conception des circuits imprimés se complexifie elle aussi. Avec les développements grandissants des circuits intégrés, il devient plus ardu de concevoir les circuits imprimés qui supportent ces composantes électroniques. Il est donc impératif de continuer les recherches dans le domaine de la conception des circuits imprimés si nous voulons avoir la capacité de mettre rapidement en marché les nouvelles composantes électroniques dans des produits utiles aux consommateurs.

OBJECTIFS SPÉCIFIQUES DU PROJET

Premièrement, par une recherche bibliographique, nous pourrons sonder ce qui a été réalisé et développé dans le domaine spécifique du routage des circuits imprimés. De plus, nous vérifierons l'état des recherches dans le domaine du routage des circuits intégrés puisque, comme mentionné à la section précédente, beaucoup de recherches ont été effectuées et pourraient possiblement être transposées aux circuits imprimés.

A partir de cette recherche, nous catégorisons les routeurs existants en distingant leur mode d'opération et leur algorithme de routage. En utilisant certaines des caractéristiques propres à ces routeurs, nous proposons une architecture pour un nouvel environnement de routage, lequel spécifiera le ou les modes d'opérations ainsi que les algorithmes idéaux pour le routage de circuits imprimés. Nous détaillerons donc notre environnement de routage qui est constitué d'une structure hiérarchique utilisant également un mode de fonctionnement itératif à chacun des niveaux de la hiérarchie.

Concrètement, nous démontrerons la possibilité de hiérarchiser un algorithme de routage comme celui de Lee [1].

D'autre part,nous préciserons les avantages et les inconvénients de l'utilisation du langage Prolog pour l'implantation d'un premier prototype de notre environnement de routage. Le choix d'utiliser l'environnement de travail d'Arity Prolog [24,25,26,30] pour le développement du prototype, nous a ammené à développer et à intégrer une bibliothèque de prédicats Prolog permettant l'usage du graphisme à haute résolution. Ces prédicats sont nécessaires puisque la validation des résultats du routeur hiérarchique se fera essentiellement par inspection visuelle.

Nous développons donc un routeur global à grosses mailles, sous produit des algorithmes de la famille de Lee, qui permet d'identifier les difficultés reliées à la hiérarchisation du routage. Nous pouvons alors comparer les résultats du routeur hiérarchique avec ceux d'un expert humain, afin d'identifier les problèmes reliés à une hiérarchisation lesquels nous permettrons de proposer des voies de recherche futures.

CHAPITRE 1

PROBLÉMATIQUE

Le routage, ou recherche de chemin est un problème dont les premiers algorithmes remontent déjà à un quart de siècle. L'un des premiers algorithmes de routage appliqué aux circuits imprimés a été développé par C.Y. Lee en 1961 [1]. Depuis ce jour, une quantité impressionnante de routeurs ont été développés; tous plus spécialisés ou rapides que les autres.

1.1 LE PROBLÈME

Prenons le temps de définir ce qu'est le routage. Quand on fait des prototypes de circuits électriques ou électroniques, on utilise habituellement soit une plaquette de prototypage, soit une plaque à wirewrap. Dans les deux cas, nous posons des fils conducteurs entre des objets ou composantes. Ces fils ont une gaine isolante qui permet de les mettre un par-dessus l'autre sans qu'il y ait contact électrique. Habituellement, quand le prototype est terminé, il y a sur la plaque une grosse quantité de fils qui se croisent et se torsadent.

Si nous voulons fabriquer un produit facilement manufacturable et en grande quantité, nous ne pouvons plus utiliser ces techniques de prototypage qui sont longues et fragiles. La technologie des circuits imprimés nous permet de reproduire facilement un même circuit plusieurs fois et rapidement. Un circuit imprimé est constitué de chemins de cuivre qui reposent sur une plaque de résine isolante; ce sont les chemins de cuivre qui remplacent maintenant les fils du prototype. Un circuit imprimé peut avoir plusieurs couches, chaque couche étant électriquement isolée des autres. Il est possible de faire un contact électrique entre deux chemins de couches différentes par une traverse. Les traverses sont des conduits de cuivre qui passent au travers de la résine isolante pour rejoindre les chemins sur deux couches distinctes. Les traverses sont toujours perpendiculaires au plan des couches. (Figure 1.1).

La difficulté est de transformer les fils de notre prototype en chemin sur une ou plusieurs couches grâce aux traverses. L'inconvénient avec les chemins de cuivre, c'est qu'ils n'ont que deux dimensions, ils sont contenus dans un plan ou couche ce qui empêche qu'ils ne se croisent ou qu'ils ne se touchent. Le routage est l'action de créer ces chemins de cuivre afin d'obtenir un circuit imprimé manufacturable et fonctionnel.

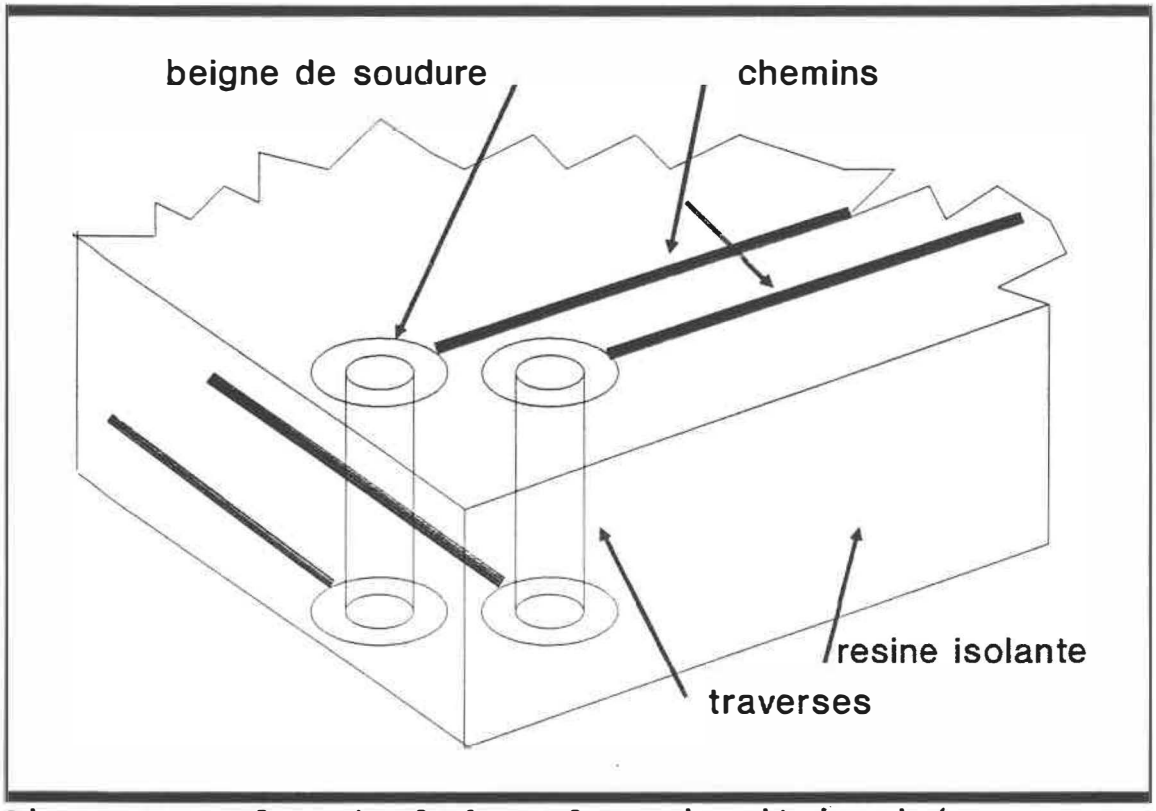


Figure 1.1 Elements de base d'un circuit imprimé

Donc par routage, nous voulons dire qu'étant donné des objets ou composantes dans un plan, il nous faut construire un ensemble de connexions entre ces objets afin de satisfaire un ensemble de critères de connectivité. Les deux considérations qui rendent le problème de routage difficile sont:

- que les chemins sont conducteurs et qu'ils ne doivent pas se toucher les uns les autres. Il faut que le placement d'un chemin tienne compte des autres chemins qui ont déjà été placés;

- chaque chemin requiert un espace physique sur la plaque ce qui réduit l'espace disponible pour les autres chemins [2].

C'est pourquoi habituellement 90% des chemins seront routés en 10% du temps et les derniers chemins prendront 90% du temps total pour router le circuit au complet. Ceci s'explique par le fait suivant : plus nous plaçons de chemins sur la plaque, plus il devient difficile de placer un nouveau chemin étant donné que nous avons plus de contraintes à respecter et moins d'espace dû aux chemins déjà en place [3]. En fait, le problème de routage ainsi que celui du placement des composantes sur une plaque de circuit imprimé sont démontrés être de complexité NP-complet [4].

Les objets qu'il faut connecter peuvent être vus comme des composantes électriques à différents niveaux. Dans un circuit intégré, les composantes sont des cellules (cells) qui occupent une certaine région du circuit intégré que nous ne pouvons pas utiliser pour le routage. Sur un circuit imprimé, les composantes sont des circuits intégrés et des composantes électroniques qui occupent une région de la plaque, mais cette région peut être utilisée pour le routage du circuit imprimé. Les problèmes de routage de circuit

imprimé ressemblent beaucoup a ceux de routage de cellules de circuit intégré.

Le routage est un gros problème dans le sens qu'il prend beaucoup de mémoire et de temps de calcul. Un routeur est un programme qui automatise le processus de routage. Les objectifs premiers d'un routeur sont de trois ordres:

1. de router entièrement toutes les connexions;
2. de donner une solution qui est possiblement réalisable suivant les technologies;
3. d'être efficace avec l'utilisation des ressources informatiques.

Ces trois buts sont mutuellement exclusifs. Un bon routeur réalisera un compromis entre les objectifs qui nous intéressent le plus.

Le problème de routage n'est pas nécessairement réalisable. La géométrie du placement des composantes sur une plaque de circuit imprimé est considérablement importante. Il est facile de placer les composantes afin que le problème de routage soit impossible. Cependant, de petits changements dans le placement peuvent changer le problème d'impossible, à difficile, à raisonnable.

1.2 LES ROUTEURS

Les routeurs sont des programmes automatiques ou semi-automatiques pour router complètement ou partiellement un certain circuit. Dans notre cas, nous allons porter notre attention sur le routage de circuit imprimé. Ces programmes utilisent des techniques de recherche ou de fouille afin de trouver un chemin d'un point à un autre sur un circuit imprimé, en respectant certaines contraintes. Pour ne nommer que certaines de ces techniques de recherche, il y a les recherches en profondeur d'abord ou en largeur d'abord, les recherches aveugles, les recherches meilleures d'abord, les recherches exhaustives et enfin les heuristiques [5]. Ces techniques sont efficaces pour des objectifs différents et des applications différentes. Les heuristiques sont des recherches très rapides qui demandent une connaissance approfondie du domaine d'application de la recherche. C'est une recherche spécialisée, rapide qui n'assure pas de trouver le meilleur résultat. Si on utilise plutôt les recherches exhaustives, elles assurent de trouver la meilleure solution à notre recherche, si elle existe. Mais nous ne sommes pas assurés de pouvoir terminer la recherche dans un temps raisonnable.

Nous pouvons donc remarquer que le choix d'une technique de recherche n'est pas facile et nous devons

habituellement faire des compromis afin d'obtenir un système performant et acceptable. Le temps et la qualité de la recherche sont les deux critères qui sont directement proportionnels l'un à l'autre. Nous devons faire des compromis sur ces deux critères.

Plusieurs routeurs existent de nos jours. Il est difficile de choisir ou de dire lequel est le meilleur, puisqu'ils ne répondent pas tous aux mêmes objectifs. De plus, il est très difficile d'évaluer leur niveau de performance puisqu'ils sont implantés sur des ordinateurs différents, ayant des puissances différentes, et parce qu'on leur soumet des ensembles de connexions ou circuits différents.

Les principales contraintes imposées aux routeurs sont:

- fil ou route le plus court possible suivant la distance de Manhattan;
- route de grandeur précise ou avec un délai de propagation fixe;
- nombre limité de couches;
- largeur minimale d'une route;
- diamètre minimal d'un beigne de broche;

- diamètre minimal d'une traverse;
- nombre maximal de traverses pour un fil;
- direction privilégiée d'une couche;
- nombre limité de routes;
- nombre limité de passes ou d'itérations pour router complètement un circuit imprimé;
- largeur des grilles;
- nombre de couches que relie une traverse;
- grandeur de la région de routage;
- forme de la région de routage;
- nombre et grandeur des canaux;
- nombre et grandeur des échangeurs.

La distance de Manhattan n'est pas la distance comme nous la connaissons en géométrie, soit la longueur de la droite la plus courte possible entre deux points (distance euclidienne) (Voir figure 1.2).

La distance de Manhattan est la distance comme la mesure un chauffeur de taxi sur l'ile de Manhattan soit: d+e.

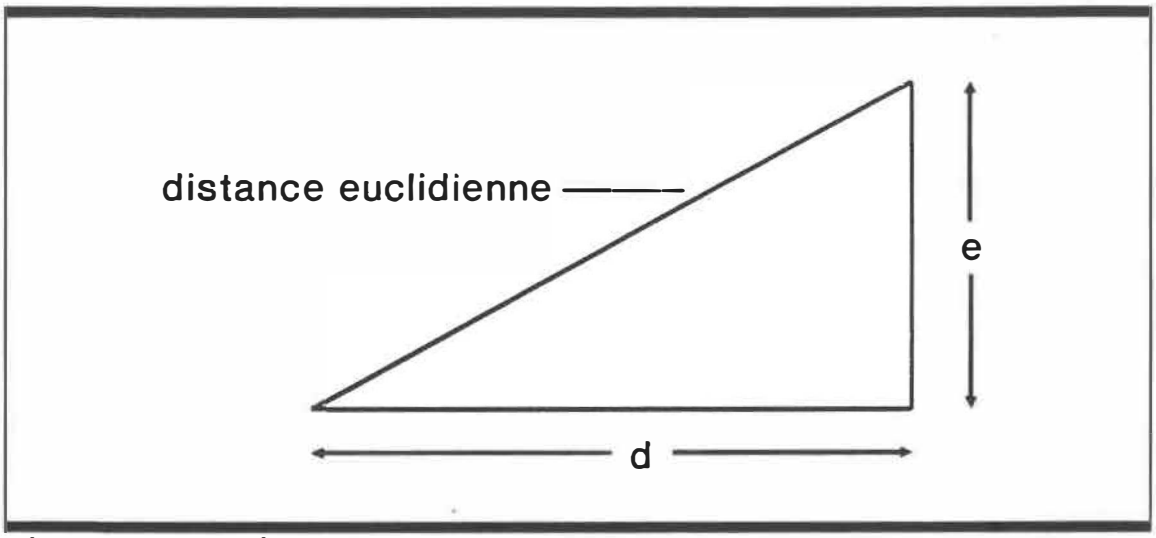


Figure 1.2 Distance de Manhattan

Nous devons donc maintenant vérifier les différences qui existent entre tous les routeurs qui ont été développés depuis M. Lee. Cette catégorisation des routeurs nous permettra de trouver quel est le meilleur routeur existant.

CHAPITRE 2

ARCHITECTURE HIÉRARCHIQUE

Une bonne manière de comparer les routeurs est de le faire par catégorie. Il faut comparer des pommes avec des pommes et les bananes avec les bananes.

2.1 TYPES DE ROUTEURS

Il y a deux critères qui permettent de différencier les routeurs. Le premier critère qualifie le mode de fonctionnement du routeur (voir figure 2.1). Les routeurs peuvent être séquentiels, itératifs ou hiérarchiques. Ceux qui s'exécutent séquentiellement peuvent se séparer en deux classes, les simples passes et les multi-passes.

Les programmes à une seule passe sont des programmes séquentiels qui vont compléter ou essayer de compléter le routage d'un seul coup. Il n'y a pas de pré-traitement ni de modification d'un chemin déjà en place. Les routeurs à plusieurs passes utilisent les premières passes pour s'informer des difficultés du circuit. Ensuite ils

planifient le routage qu'ils vont faire et enfin dans les dernières passes, effectuent le routage proprement dit.

Contrairement aux routeurs séquentiels, les routeurs itératifs ont la possibilité d'enlever des chemins qu'ils ont déjà placés. Ceci leurs permettent de router un chemin qui était bloqué par un autre. Les routeurs itératifs sont habituellement plus lents que les routeurs séquentiels, mais ils garantissent un pourcentage de complétude plus élevé.

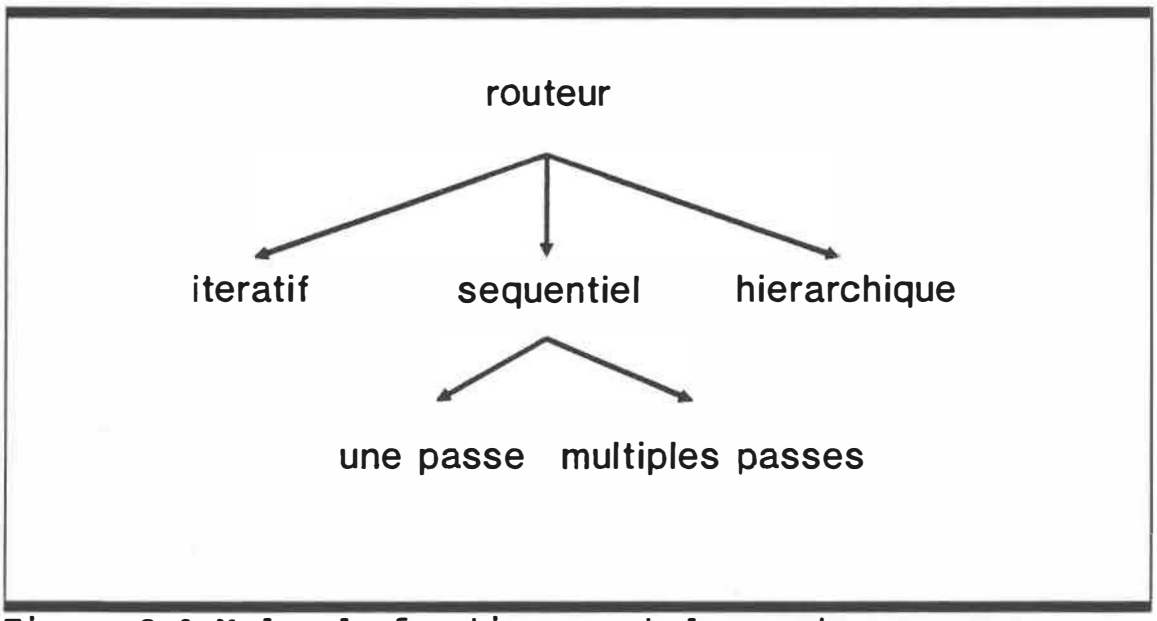


Figure 2.1 Modes de fonctionnement des routeurs.

Enfin, les routeurs hiérarchiques utilisent des niveaux d'abstractions successifs. Les plus hauts niveaux dans la hiérarchie effectueront un travail grossier, mais important pour la planification du routage final. Plus nous descendons dans la hiérarchie et plus le travail dans le routage est précis et long.

Enfin, il est possible de combiner certaines de ces classes de routeur afin d'obtenir par exemple un routeur hiérarchique et itératif.

Le second critère qui permet de différencier les routeurs est le type d'algorithme de routage utilisé. Dans les types d'algorithme, nous pouvons en identifier deux classes. Ceux traitant le problème globalement et ceux qui ne traitent qu'une région de l'espace de routage. Nous les nommerons routeurs globaux et locaux respectivement (voir figure 2.2).

Les routeurs globaux peuvent se diviser en deux sous-classes. D'une part nous avons les routeurs avec grille de travail et d'autre part ceux sans grille de travail. Dans la sous-classe des routeurs globaux avec grille de travail, nous avons les algorithmes de la famille de Lee. Ces derniers sont sûrement les plus vieux ancêtres des algorithmes de routage.

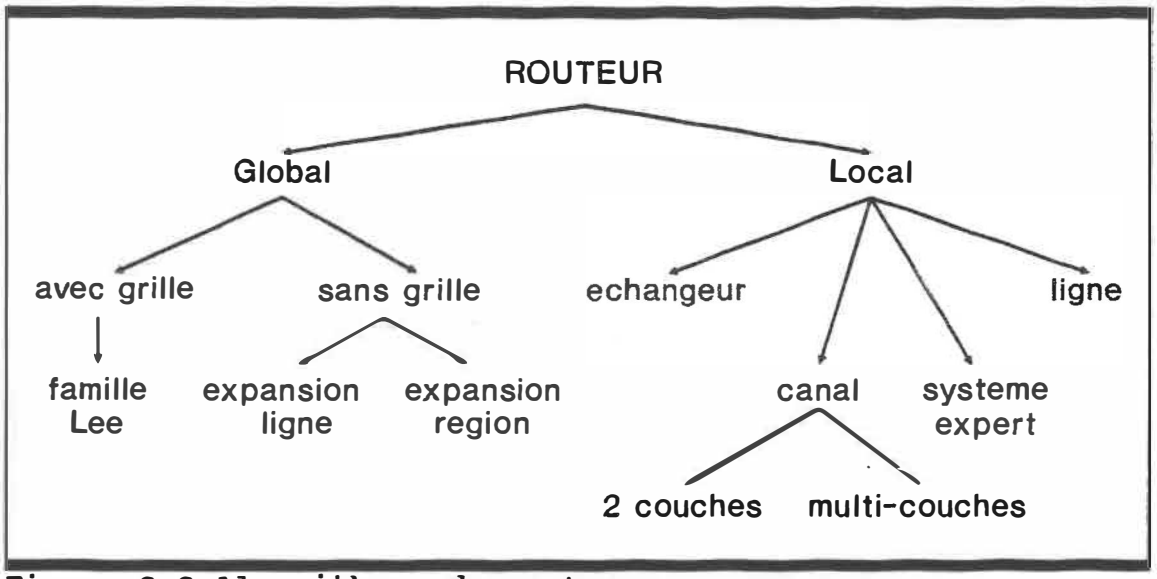


Figure 2.2 Algorithmes de routages.

Dans la sous-classe des routeurs globaux sans grille de travail, nous en avons deux types différents: les algorithmes par expansion de ligne et les algorithmes par expansion de région. Les algorithmes par expansion sont moins exhaustifs que les algorithmes de la famille de Lee mais ils sont beaucoup plus rapides et consomment moins de mémoire.

Les routeurs locaux se divisent en quatre sous-classes: les routeurs par canal, les échangeurs, les routeurs de lignes et enfin les systèmes experts. Les routeurs par canal sont habituellement divisés en deux types distincts: les routeurs à deux couches et ceux à plus de deux couches.

Chacun de ces routeurs sera décrit plus en détails à la section suivante.

2.2 ALGORITHMES DE LA FAMILLE DE LEE

Cette famille d'algorithmes est sûrement la plus ancienne de toutes les familles d'algorithmes de routage. Ces algorithmes sont basés sur une recherche exhaustive globale d'un chemin dans un espace de travail restreint par une grille de base montrée à la figure 2.3. Le premier algorithme est introduit par Lee [1] en 1961. L'algorithme de Lee assure de trouver le plus court chemin -suivant la distance de Manhattan- qui existe entre deux points si un chemin existe. L'algorithme examine tous les chemins possibles avant de trouver le bon. C'est un algorithme très lent qui utilise beaucoup de mémoire. Plusieurs personnes se sont attardées à l'amélioration des performances (vitesse et mémoire) de l'algorithme de Lee [6,7,8]. Chacune de ces modifications augmente la rapidité de l'algorithme tout en diminuant sa consommation en mémoire. Cependant, certaines de ces modifications ont le désavantage de ne plus être des recherches exhaustives.

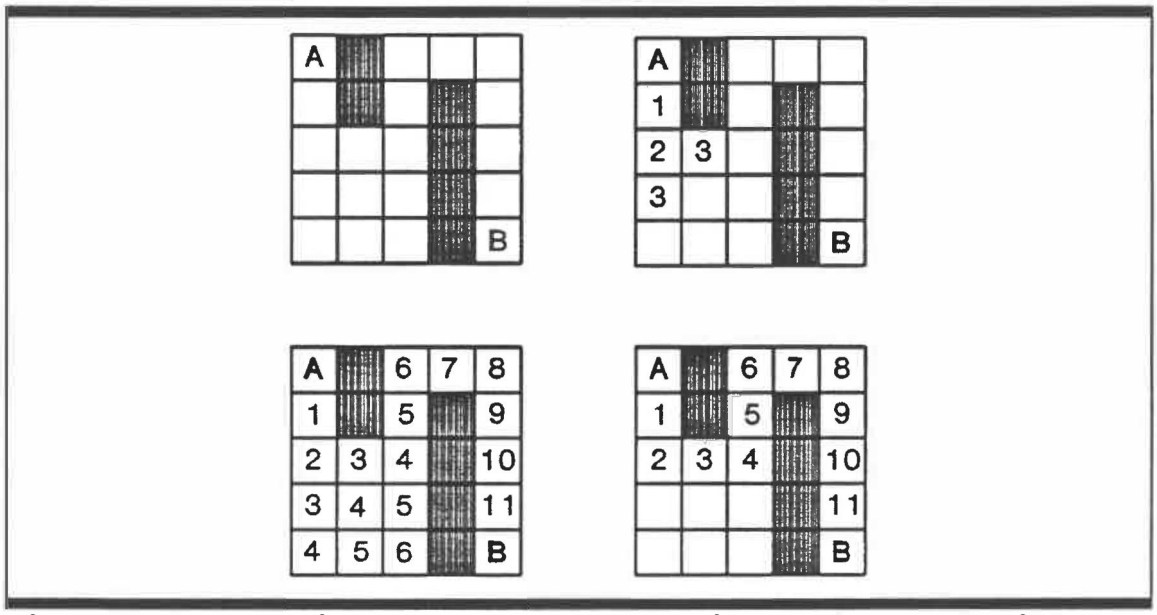


Figure 2.3 Fonctionnement d'un algorithme de la famille de Lee.

Le GREEDY routeur de DEC [3] augmente les contraintes de travail pour obtenir plus de rapidité. Le routeur utilise une grille de travail limitée tandis que le routeur de Lee travaille sur une grille de travail la plus fine possible. Il ne peut utiliser que des lignes horizontales ou verticales, pas de diagonales. Les tracés sont tous de la même largeur. Il ne permet pas d'utiliser la technologie de montage en surface.

Les algorithmes de la famille de Lee utilisent toujours une grille de travail. Plus cette grille sera fine et plus le traitement sera long.

2.3 ROUTEUR SANS GRILLE

Les algorithmes de routage sans grille de travail ont l'avantage d'utiliser beaucoup moins de mémoire que ceux de la famille de Lee. De plus, puisqu'ils n'ont pas de régions prédéfinies de routage, ils sont plus flexibles aux changements de technologie comme l'utilisation de lignes fines dans le design et la fabrication de circuits imprimés. L'aggorithme construit des régions d'expansion qui dépendent des obstacles et de limites du circuit. Voir figure 2.4.

Mackenzie [9] propose une méthode qui utilise le concept d'expansion d'une région dans toutes les directions sans rester sur une grille de travail. Le routeur permet les simples ou multi-couches en plus de supporter les composantes de surface et les routes de largeur variable. Le routeur est dirigé par une fonction de coût, complexe et longue à évaluer.

Clow [5] utilise une technique de recherche de lignes sans grille de travail. Il effectue une recherche globale d'un chemin en évitant les obstacles, sans utiliser de canal. Il utilise aussi une fonction de coût pour diriger les recherches. Son algorithme se veut une spécialisation de l'algorithme de Lee.

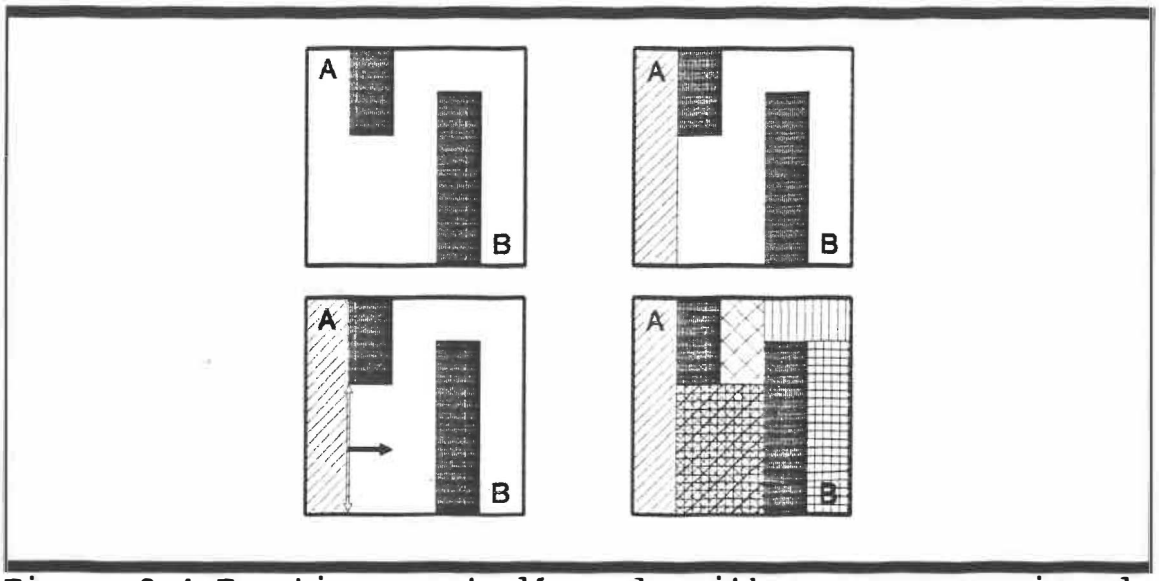


Figure 2.4 Fonctionnement d'un algorithme par expansion de régions.

2.4 ROUTEUR PAR CANAL

Un canal peut être soit horizontal ou vertical. La figure 2.5 présente un canal de routage dont initialement, l'intérieur était complètement libre pour le routage. Le circuit intégré est divisé en rangées et en colonnes de largeur variable. Le routeur effectue son travail soit dans une rangée, soit dans une colonne. L'espace de travail est réduit et génère ainsi un gain de rapidité et une diminution de l'espace mémoire requis.

Le routeur par canal de Bobba et Smith [2] est paramétrique. Ces paramètres, lesquels sont variables, sont le nombre de couches, le nombre de canaux, la densité de

traverses... Le routeur fonctionne en deux étapes. Premièrement, il établit une liste des fils à poser et assigne les traverses nécessaires. À la seconde étape, les couches sont attribuées et le routage s'effectue. Le routeur n'a pas été complètement testé. Seul de petits designs ont été expérimentés.

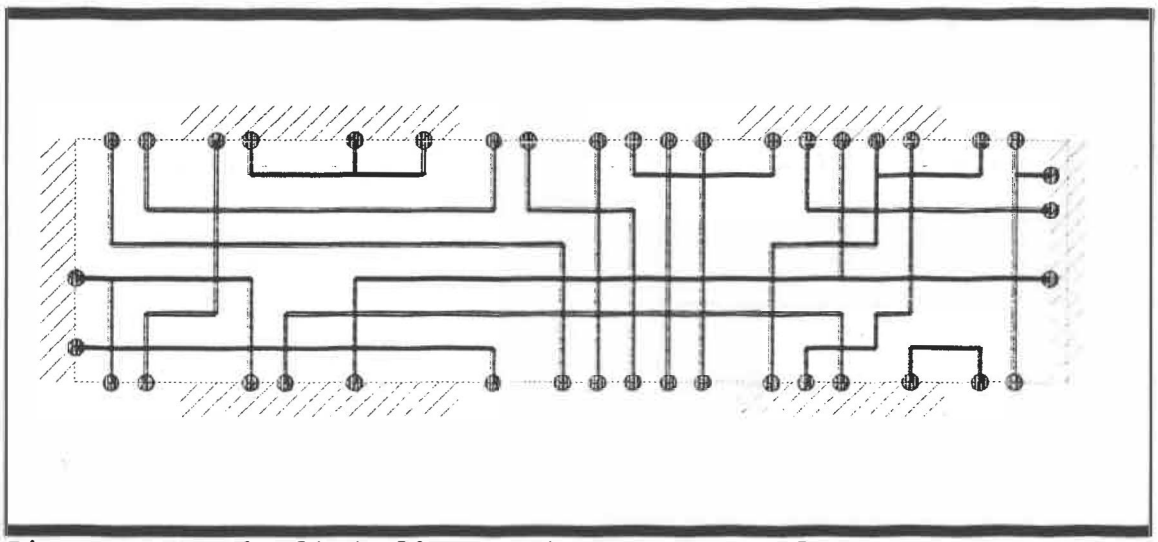


Figure 2.5 Résultat d'un routage par canal.

Le CONCURRENT routeur de Hanafusa [10] semble être très rapide, il fonctionne lui aussi en deux étapes. Premièrement chaque fil est assigné à un canal, ce qui a pour effet de vérifier la densité de fils dans chaque canal. Ensuite le routeur procède à l'assignation des voies (tracks) dans chaque canal. Le routeur utilise des grilles dans chaque canal.

Le routeur de Dupenloup [11] utilise un routeur par canal comme celui de Dogleg [11] ou celui de Deutsch [11]. Il utilise également un routeur par échangeur. Le routeur a été développé pour le routage de cellules en VLSI, problème pour lequel il y a des régions d'obstacles. Il utilise deux niveaux de routage. Au second niveau, on retrouve un routeur par canal horizontal et un autre par canal vertical.

Finalement, le routeur de Yoshimura [12] se veut une version améliorée du routeur par canal LEFT EDGE de Hashimoto [12]. Il semble très rapide.

Certains de ces routeurs sont destinés uniquement à des circuits intégrés à deux couches tandis que d'autres peuvent travailler sur plusieurs couches. Le travail à plusieurs couches augmente le nombre de possibilités de routage. Il faut un mécanisme pour choisir la couche qui permettra à un chemin d'être le plus efficace possible.

2.5 ROUTEUR PAR ÉCHANGEUR

Pour travailler avec un routeur par échangeur, le circuit intégré est divisé en plusieurs petits carreaux de grandeur fixe. Le routeur travaillera successivement dans chacun de ces petits carreaux. Habituellement, les routeurs

par échangeur sont utilisés conjointement avec des routeurs par canal. La région de routage de l'échangeur est définie par l'intersection d'un canal horizontal et d'un canal vertical. Voir figure 2.6.

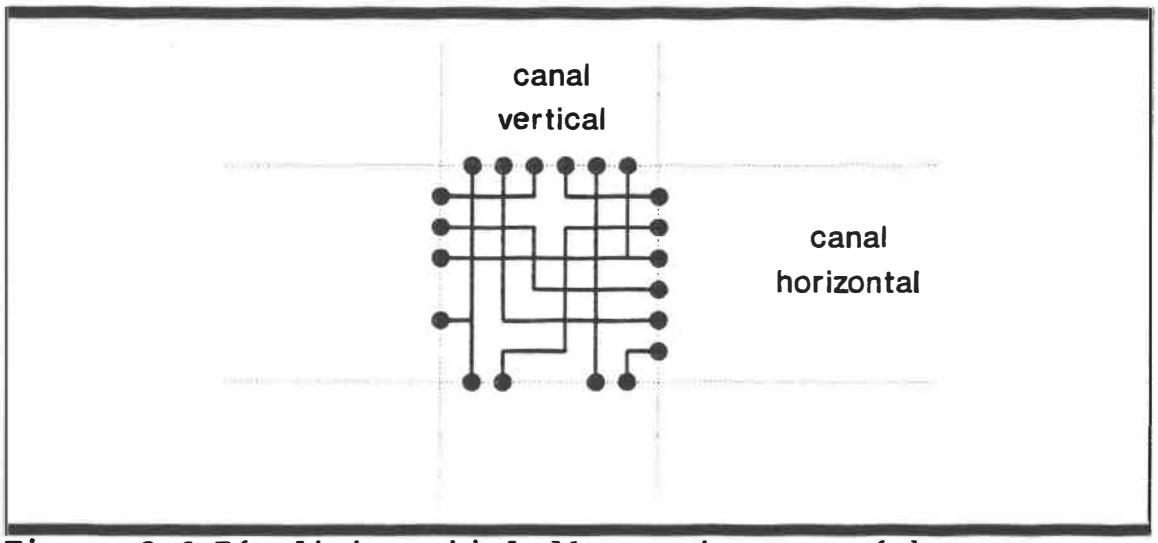


Figure 2.6 Résultat partiel d'un routeur par échangeur.

Le routeur par échangeur de Malgorzota [13] utilise des heuristiques afin de router dans deux dimensions en utilisant deux couches distinctes. Il produit un routage du style Manhattan ou les points de contacts se retrouvent aux frontières des régions de routage.

2.6 ROUTEUR DE LIGNES

Les routeurs de lignes permettent de router plusieurs points rectilinéaires. Ce sont des algorithmes très

efficaces pour simplifier des réseaux de connexions sur une même ligne. Voir figure 2.7.

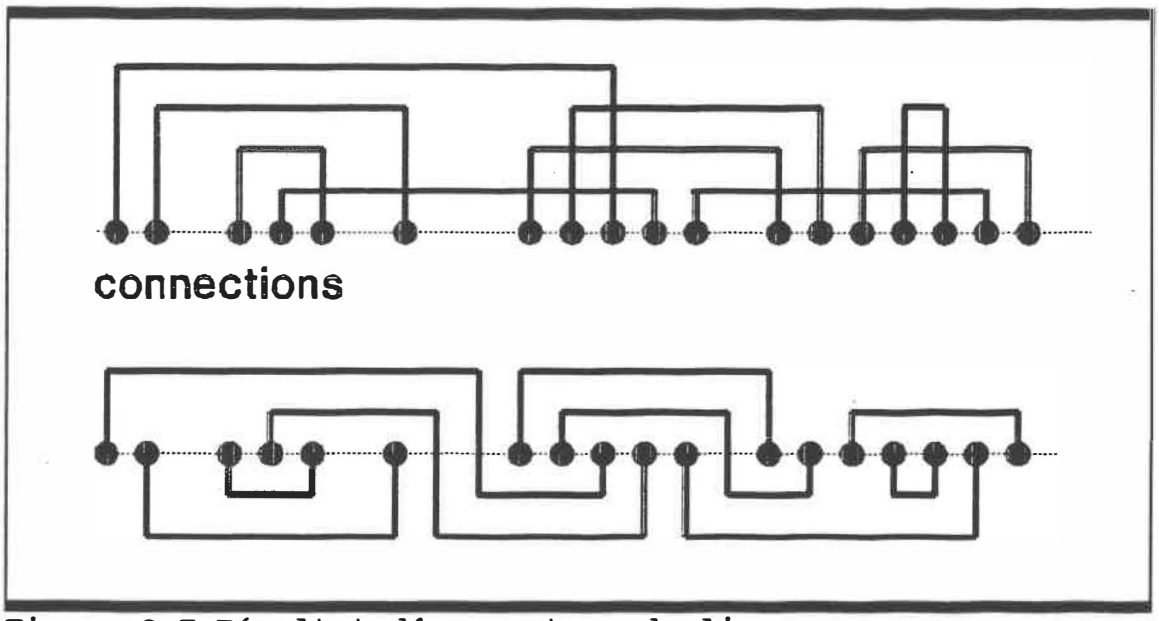


Figure 2.7 Résultat d'un routeur de lignes.

Le routeur de Han et Sahni [14] est un algorithme pour router des points sur une même ligne en utilisant deux couches. Les chemins sont analysés avant que le routage ne soit fait. Il utilise l'algorithme de canal gourmand (greedy channel algorithm) de même qu'un algorithme de blocage afin d'obtenir le pourcentage de complétude le plus élevé possible.

Le CONCURRENT routeur que nous avons abordé précédemment utilise un routeur par ligne afin d'effectuer l'assignation des voies dans les canaux.

2.7 ROUTEUR PAR SYSTÈME EXPERT

Les routeurs par système expert sont habituellement des routeurs locaux. Ils tentent de résoudre des problèmes très spécifiques. Ce sont des routeurs très lents. C'est une approche nouvelle qui n'est pas beaucoup utilisée dans les systèmes de conception assistée par ordinateur de circuits imprimés commerciaux. L'approche semble très intéressante quoique très lente et lourde à implanter.

Le routeur de Joseph [15] s'occupe de blocages locaux. Il nécessite un premier routage de tous les fils. Certains de ces fils peuvent être partiellement routés dû à un blocage. Au lieu d'essayer un autre chemin, le premier routeur laisse la route incomplète pour le second routeur par système expert. Celui-ci utilise des règles afin d'identifier la géométrie du blocage dont on peut voir une représentation à la figure 2.8. Il modifie les chemins voisinant le blocage afin de compléter le routage. En modifiant la géométrie avoisinante, il peut créer de nouveaux blocages. Il n'assure pas une réussite à 100%. Il a été développé pour une technologie de circuits imprimés à six couches avec des traverses fixes. Le système est séparé en trois blocs différents: la mémoire de travail, la mémoire pour les règles et enfin le moteur d'inférence, implanté en LISP.

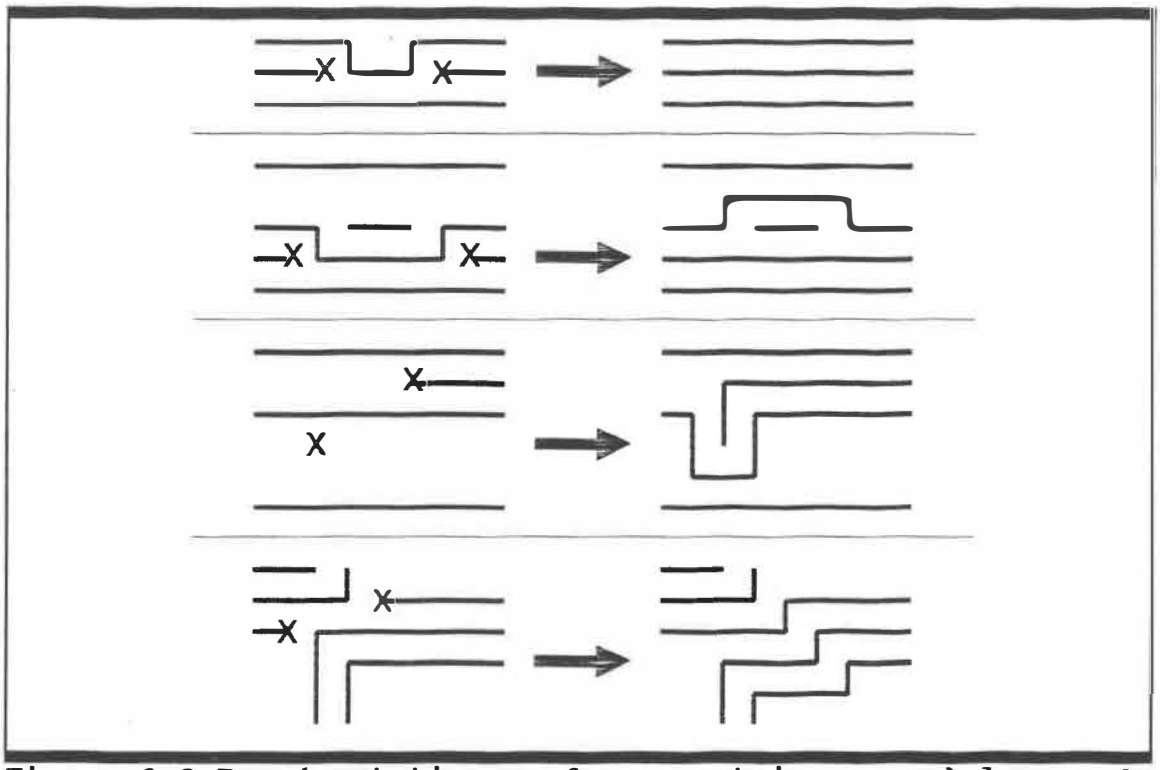


Figure 2.8 Représentation de certaines règles de simplification d'un routeur par système expert.

Le routeur de Fujita et Goto [16] est un système intéractif à base de règles. Il fonctionne pendant le processus de design sous l'ordre du concepteur. Le système est réalisé à partir des connaissances de l'expert concepteur. Il utilise un interpréteur de clause du 1er ordre en calcul de prédicat. Les règles sont des clauses de Horn du type de PROLOG et sont générées par le concepteur. Le système a été développé pour solutionner des problèmes de routage à grande échelle. Le système semble lent et il n'est pas évident qu'un concepteur a les capacités ou l'intérêt

d'entrer lui-même ses règles dans la machine.

2.8 ROUTEUR SÉQUENTIEL

Les routeurs séquentiels sont les routeurs les plus constants en temps d'exécution. Ils effectuent une ou plusieurs passes de tous les fils qu'il faut router. Une fois un chemin placé, celui-ci devient une contrainte fixe avec laquelle il faut travailler jusqu'à la fin du routage. Très souvent, ces routeurs placent des chemins à des endroits bloquant par la suite le passage de plusieurs autres chemins. Ces chemins, malheureusement ne pourront plus être déplacés.

2.9 ROUTEUR ITÉRATIF

Les routeurs itératifs sont des algorithmes qui permettent d'enlever ou de recommencer le routage d'un fil déjà routé. Si un fil est impossible à router, alors l'algorithme éliminera les fils qui encombrent le routage dans la région de blocage et effectuera un nouveau routage des fils enlevés. C'est ce qu'on appelle le processus de murissement (rip-up).

Le routeur de Rosenberg [17] utilise un algorithme de la famille de Lee combiné à une approche par relaxation de

Lagrange. Le routeur utilise une fonction de coût afin d'orienter la recherche de chemin. La méthode semble très longue à exécuter puisqu'il y a peu d'amélioration pendant les dernières itérations.

2.10 ROUTEUR HIÉRARCHIQUE

Les routeurs hiérarchiques fonctionnent par niveaux d'abstraction successifs. C'est une forme de formalisation du travail à exécuter et une planification grossière du routage qui devra être fait dans les niveaux hiérarchiques inférieurs.

Le routeur de Kowamura [18], appellé routeur hiérarchique dynamique, n'est pas un vrai routeur hiérarchique par niveaux d'abstraction. C'est plutôt un routeur itératif qui à défaut de niveaux d'abstraction utilise des niveaux hiérarchiques dans l'exhaustivité du routage. Un routage ne passe pas par tous les niveaux. S'il échoue à un niveau dans le routage, il passe à un second niveau qui est plus puissant ou exhaustif. Ce sont trois algorithmes différents de routage qui sont actionnés chacun à son tour lorsque son prédécesseur a échoué.

Le routeur de Kessenich [19] est un routeur à deux niveaux de hiérarchie. Le premier niveau utilise une grille à grosses mailles ou gros grains. Un routage grossier est utilisé afin de vérifier la densité de fils ou de traverses dans chaque maille afin d'éviter les congestions. Le second niveau est un routeur par échangeur qui travaille avec une grille beaucoup plus fine que le premier niveau, ce qui permet de compléter les détails du routage. L'approche est très intéressante puisqu'elle permet d'utiliser une précision arbitraire, des largeurs de trace variables, des grandeurs et un nombre de points et traverses variables, d'utiliser plusieurs couches et de combiner plusieurs technologies sur un même circuit.

Enfin, le routeur de Neveda [20] a été développé pour le design à multi-couches. Il exploite la possibilité de faire des traverses partielles, pas de bord en bord du circuit imprimé. Le routeur utilise cinq niveaux de hiérarchie. Au premier niveau, il estime le nombre de couches requises. Au second, il détermine globalement les broches ou traverses requises pour chaque réseau. Au niveau 3, il effectue un routage global soit grossier par un routage initial et un reroutage si nécessaire. Au quatrième niveau, il fait l'assignation des chemins à leur couche respective et enfin au cinquième et dernier niveau, il effectue un routage

détaillé.

Nous pouvons conclure de cette classification, qu'il est difficile de choisir le routeur qui sera le meilleur pour une certaine application précise. De plus, Il est très difficile de comparer tous ces routeurs, puisqu'ils sont toujours évalués par le routage de réseaux de connections différents de même que sur des ordinateurs différents.

Pour bien comparer les performances des routeurs, il faudrait une métrique tenant compte de la densité du circuit, du nombre de connections, du nombre de traverses, du nombre de fils, du nombre de couches, des dimensions de l'espace de routage,... Il faudrait que la métrique réflète bien les différents circuits ou bancs d'essais utilisés.

2.11 ROUTEUR PAR HIÉRERCHIE D'ABSTRACTIONS SUCCESSIVES.

Après avoir fait la classification de la section précédente, nous pouvons tenter de répondre à la question suivante: Quel est le meilleur Routeur?

Nous ne pouvons pas obtenir une réponse absolue. Chacun des routeurs des sections précédentes ou une combinaison d'algorithmes de routages avec certains modes de

fonctionnement sont efficaces pour des problèmes de routage particuliers. Le meilleur routeur serait celui qui incorpore toutes les bonnes caractéristiques de tous les algorithmes de routage et des modes de fonctionnement et qui pourrait travailler avec des contraintes de temps et de mémoire respectables.

Nous allons donc introduire une nouvelle approche au routage qui est plus qu'un routeur, c'est un environnement de routage [31].

2.11.1 HIÉRARCHIE

Le routeur possèdera plusieurs niveaux de hiérarchie. Par opposition à une hiérarchie de routage par échec, nous emploierons une hiérarchie par succès. Le routeur de Kawamura [10] est un routeur utilisant une hiérarchie par échec. Si un premier niveau de routage échoue, le contrôle est passé à un second niveau afin d'obtenir un succès.

Le routeur par hiérarchie de succès passe à un niveau plus bas dans la hiérarchie uniquement lorsque le niveau précédent a obtenu un succès. La hiérarchie est ordonnée de façon à obtenir des niveaux d'abstractions successives. Le travail à haut niveau se fait sur des objets grossiers ce qui limite la quantité de données à traiter et le temps d'exécution. Les plus hauts niveaux de la hiérarchie s'occupent de la planification des routes qu'il y aura sur le circuit imprimé. C'est sûrement l'étape la plus importante dans le routage d'un circuit imprimé si nous voulons obtenir un routage complet de tous les fils.

Plus le routeur descend dans les derniers niveaux de la hiérarchie, plus le routage devient local, précis et final. Puisqu'il n'est pas possible de prédire que la planification de haut niveau nous permettra par une première

itération de compléter le circuit imprimé, il sera possible de remonter dans la hiérarchie afin de modifier la planification faite. A chaque niveau du routage, il sera possible de remonter d'un niveau ou de plusieurs niveaux d'un seul coup. Il sera également possible de remonter au niveau du placement.

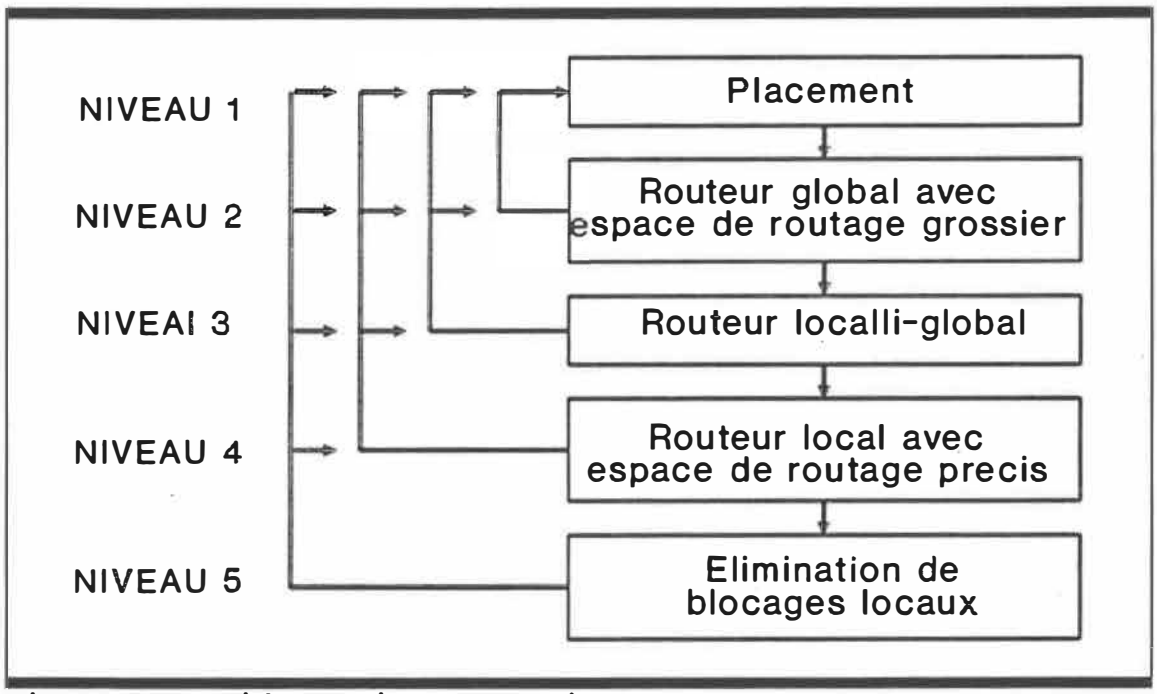


Figure 2.9 Hiérarchie du système.

La particularité de ce nouveau routeur est l'étape du placement, laquelle fait partie intégrante du module de conception physique d'un circuit imprimé. (Voir figure 2.9). Le module de conception physique est une composante à base de connaissance laquelle décide si un niveau de la hiérarchie

du système est acceptable ou pas. Si le travail fait à un certain niveau est acceptable, nous descendrons dans la hiérarchie. Dans le cas contraire, la composante à base de connaissance décidera quelle étape nous devrons recommencer. Il se peut que nous recommencions une étape au niveau hiérarchique présent ou encore certaines étapes au niveau hiérarchique supérieur comme le témoignent les flèches à la figure 2.9.

2.11.2 SYSTÈME EXPERT

Le module de conception physique faisant parti du système global, voir figure 2.10, est un système expert. Pour qualifier un programme de système expert, il faut que la connaissance du système provienne d'un expert dans ce domaine. Notre module de conception est bien un système expert puisqu'il utilise la connaissance d'expert dans le domaine du routage. Cependant la différence entre notre système expert et les systèmes experts traditionnels est que nos experts ne sont pas humains. Ce sont des algorithmes qui sont nos experts pour certaines résolutions de problème de routage. Notre système final aura donc la connaissance de plusieurs algorithmes experts en routage.

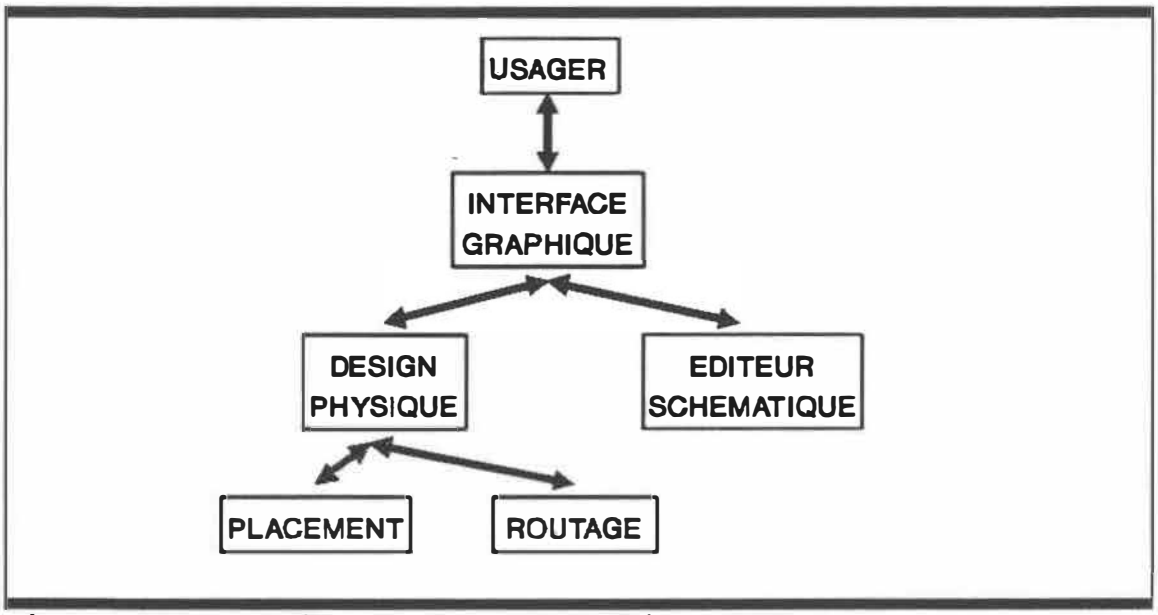


Figure 2.10 Schéme global du système.

2.11.3 BASE DE CONNAISSANCE

La base de connaissance de l'ensemble du système est divisée en plusieurs sections lesquelles se rattachent à un ou plusieurs modules du système global. Voir figure 3.11.

Les sections PLACEMENT et ROUTAGE de la base de connaissance renferment des heuristiques et des algorithmes permettant de bien placer ou router des connexions dans certains contextes d'application physique.

La section conception physique renferme les métriques, lesquelles permettent d'évaluer non seulement la complexité d'un circuit mais ainsi le travail fait à chaque niveau. De plus cette section de la base de connaissance renferme le système expert capable de choisir l'algorithme ou l'heuristique de routage qu'il faut employer dû aux contraintes physiques ou évolutives du développement du circuit imprimé.

La section contrainte et technologie renferme la connaissance des contraintes technologiques dû à la fabrication comme:

-la largeur des routes
-la grosseur des traverses et des ports
-la densité de traverses et de ports
-l'espacement entre les routes
....

D'autres contraintes sont spécifiées par le designer telles que la forme et la grandeur de la plaque voulue, l'emplacement de pièces fixes comme les connecteurs d'entrées et de sorties, le nombre de couches maximales.

Enfin, il y a des contraintes évolutives avec l'avancement du travail de placement/routage comme le nombre

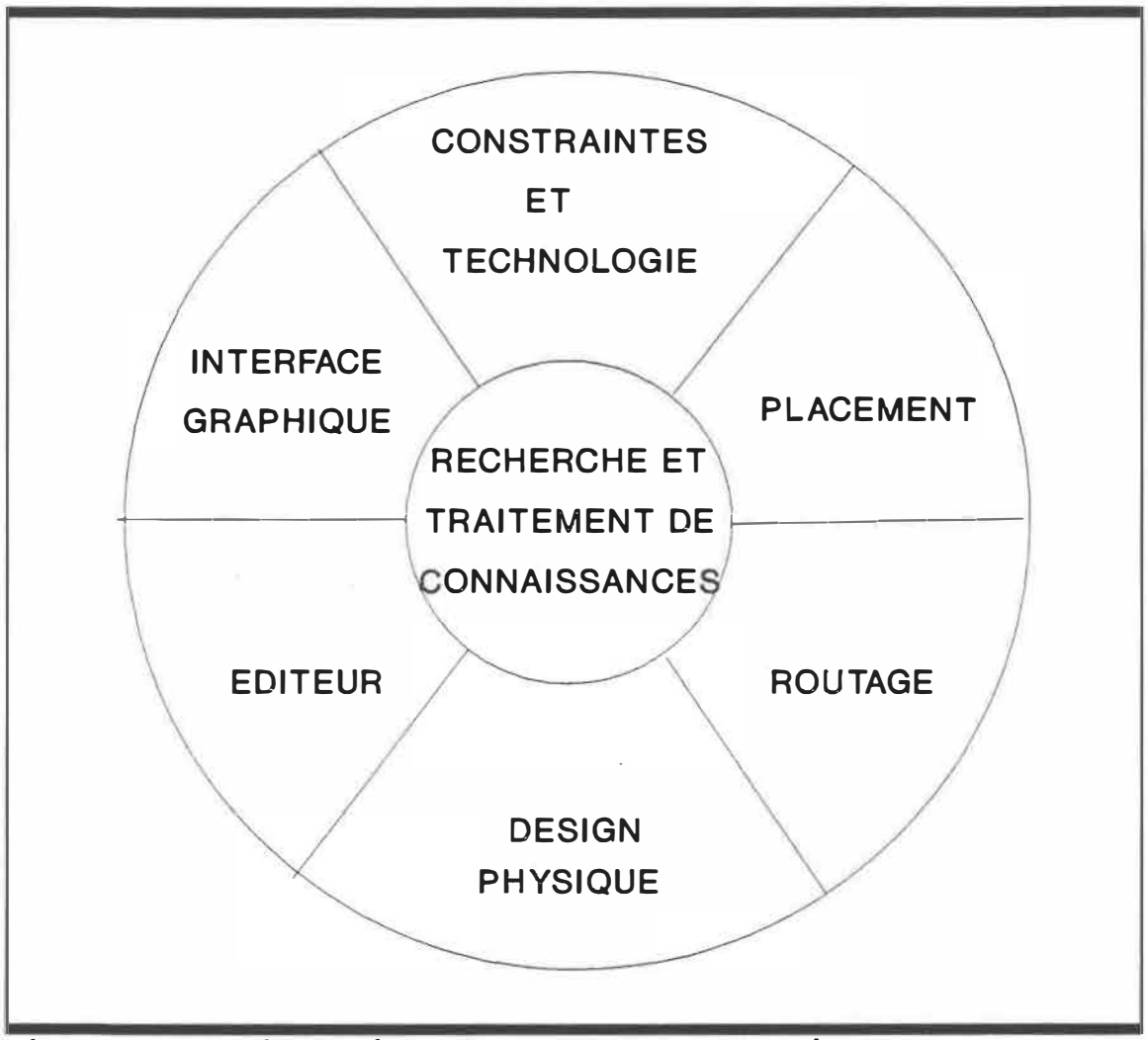


Figure 2.11 Séparation de la base de connaissance.

et la largeur des canaux, le nombre et largeur des échangeurs, la grosseur de la grille pour les routages globaux, le nombre limité d'itérations pendant les processus de mûrissement...

La connaissance de l'éditeur renferme les spécifications des composantes de chaque famille.

- TTL
- CMOS
- Microprocesseur
- Analogique: ampli op, transistor diodes ...
- Connecteurs
- RLC: résistance, condensateur, bobine
- Divers: transformateur, antenne

Ces connaissances sont organisées sous forme d'objet. Voir figure 2.12.

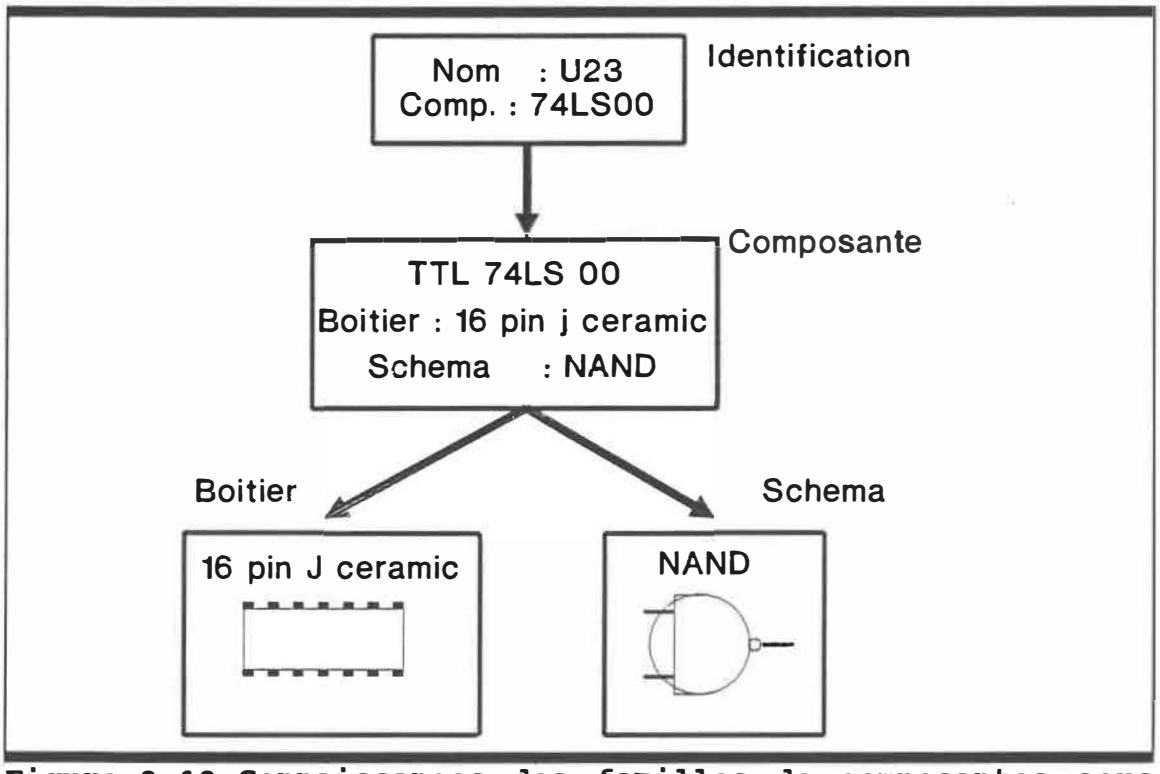


Figure 2.12 Connaissances des familles de composantes sous forme d'objets

Afin d'obtenir un environnement de routage assez puissant, permettant aussi de faire de la simulation, nous avons besoin d'une description assez complète de chaque objet. Voir tableau 2.1. Toutes ces informations seront traitées grâce à des fonctions de haut niveau permettant de manipuler des objets [21]. Nous devons avoir également des fonctions spécifiques permettant de manipuler des informations telles que:

- créer une instance de composante
- détruire une composante
- insérer une composante
- modifier une composante
- trouver les ancêtres et les descendants d'une composante

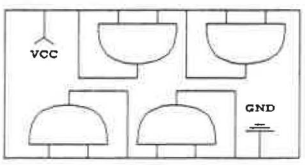
La section de connaissance de l'interface graphique possède des connaissances en géométrie et graphisme comme:

- des chemins parallèles
- des chemins perpendiculaires
- un point d'intersection de chemin.
- une région de blocage
- une connection ...

Elle contient aussi l'information permettant de gérer les fenêtres d'affichage en graphisme. On y retrouve la description des éléments constituants les mondes graphiques.

Finalement le coeur de cette base de connaissance, constitué d'un moteur d'inférences, permet de gérer et de rechercher les connaissances qui sont nécessaires à chaque module.

TABLEAU 2.1 BASE DE COMPOSANTES

Nom	U5
Fonction	NAND
Description	NAND a deux entrée, 4 par boitier.
Configuration des broches	
Nombre de broches	14 pour
-numéro de la broche	7 chaque
-nom de la broche	GND broche
-description de la broche	mise à la masse
Configuration de la boite	
Dimension	
-largeur	300
-hauteur	800
-distance entre broches	100
Dessin	
Représentation schématique	
Nombre de schéma dans le boitier	4
Dessin	
Caractéristiques DC	
Condition de test	
-paramètre	
Temp 0^0 to 70^0	
-état	
Vcc 5v ± 5%	

Caractéristiques	
-symbole	IOH
-paramètre	output high current
-limites ⟶	(minimum, typique, normal)
	(-1.0 , -1.4 , -)
-unités	a
Capacité AC	
Condition de test	
-paramètre	Temp 25^{0}C
-état	Freq. 1.0 MHz
Capacité	
-symbole	CIN
-paramètre	Input capacitance
-maximum	
-unité	5 pF
-état	VIN = Ov
Caractéristiques AC	
Condition de test	
-paramètre	Temp 0^{0}C to 70^{0}C
-état	VCC + 5V ± 5%
Caractéristiques	
-symbole	ZRC
-paramètre	Read cycle time
-limites	(min, typique, max)
	(450, - , -)
-unité	ns

2.11.4 NIVEAU PLACEMENT

Le niveau placement utilisera la liste de réseaux à router provenant du module d'édition schématique. Il attribuera à chaque composante, un emplacement physique sur la plaque. A chaque composante correspondra une référence de coordonnées physiques et une orientation.

Cette étape du design comprend deux modes de fonctionnement: un mode automatique et un mode semi-automatique. Dans le mode automatique, le module placera les

composantes sur la plaque qui aura été soit définie par le concepteur soit choisie par le module. Le placement devra respecter le plus possible certaines contraintes:

- les composantes sont orientées dans la même direction.
- il y a une distribution uniforme des composantes sur la plaque.
- la plaque garde une forme la plus carré possible.
- grandeur maximale de la plaque.

Dans le mode semi-automatique, le concepteur pourra intéragir avec le module afin d'effectuer le placement manuellement. La partie automatique s'exécutera au début et lui permettra d'effectuer des changements ensuite.

Pour les deux modes, le concepteur devra définir auparavant les caractéristiques de la plaque soit: la forme, les grandeurs ... Il devra de plus identifier les composantes qui seront à des endroits fixes sur la plaque comme:

- connecteurs d'entrée/sortie
- alimentation
- mise à la masse
- dissipateur de chaleur ...

Dans les deux modes, le concepteur pourra voir sur son poste de travail, la représentation extérieure de chaque composante, son identification, les broches et les connections sous forme d'élastique.

Le module complètera la liste de réseaux en y ajoutant la position de chaque composante sur la plaque.

Il produira un fichier permettant de dessiner l'identification et la représentation extérieure des composantes sur le circuit imprimé final. Ce dessin est fait avec de la peinture et permet d'identifier les endroits ou seront soudées les composantes électroniques.

Il produira le masque de soudure qui sera utilisé dans les étapes de fabrication.

Une fois cette première étape réalisée, le module passe à l'évaluation de la métrique d'analyse de la densité de ports sur la plaque. Il envoit cette mesure au module de conception physique. Le module de conception physique décide si le processus de design échoue ou réussit à cette étape. En cas d'échec, un autre placement est effectué afin d'optimiser la distribution des composantes sur la plaque. Si le processus de design réussit, le design est transféré au

niveau 2 de la hiérarchie d'abstraction successive soit, le routage global à gros grain.

2.11.5 NIVEAU ROUTAGE GLOBAL À GROS GRAIN.

Le routage à gros grain sera un routeur global. Il utilisera une grille à grosse maille. La grosseur des grilles sera déterminée en fonction des dimensions de la plaque voulue. Nous obtiendrons ainsi, un nombre de mailles respectables afin d'obtenir un traitement global de tous les chemins en un temps raisonnable. Cette étape mettra en évidence la complexité du routage en plus des endroits ou la congestion est intense.

Si toutes les connections ne sont pas routables, le module de conception physique choisira l'une des actions suivantes:

1- enlèver les fils dont les routes bloquent le routage d'un fil et recommencer le routage de ceux-ci ultérieurement.
2- recommencer le placement si trop de connections n'ont pu être routées.

Le reroutage s'effectue en identifiant les réseaux qui bloquent un fil,puis en les enlevant. Il effectue ensuite le routage des chemins pas encore placés et par la suite recommence le routage des chemins enlevés.

Si après quelques itérations d'enlèvements/reroutages, il est impossible de placer toutes les connexions, le placement est recommencé.

2.11.6 NIVEAU ROUTAGE GLOBALI-LOCAL

Le routeur globali-local est utilisé comme tampon entre le routeur global à gros grain et le routeur local. C'est en fait un raffinement du routeur à gros grain en ce sens qu'il propose et planifie le travail qu'aura à faire le routeur local.

Le routage de niveau 3 sera restreint puisque son espace de recherche sera confiné à l'espace à gros grain choisi lors du premier routage. Ceci permet d'effectuer un routage fin sans avoir à utiliser tout l'espace de travail.

Le routeur placera les traverses qui sont nécessaires entre les routes verticales et horizontales. Nos appelerons cette étape, le placement de traverses.

Si le routeur à fin grain est incapable de router certains chemins, il devra revenir à une des étapes précédantes de la hiérarchie soit au routeur global à gros grain ou au module de placement. Par ailleurs, il pourra également effectuer des itérations au niveau de routage globali-local lui-même.

2.11.7 NIVEAU ROUTAGE LOCAL

A ce niveau, nous utilisons des algorithmes de routage locaux. La grille de travail à ce niveau devra respecter les contraintes physiques de fabrication.

Si nous obtenons 100% de complétude dans le routage des réseaux à cette étape, nous avons terminé le placement/ routage.

Si le routage est presque complet, nous passons au niveau suivant, lequelle permet d'éliminer des blocages locaux.

Dans le cas d'impossibilité de compléter le routage, le module de conception physique décidera à quel niveau nous devons remonter dans la hiérarchie pour reprendre le travail.

2.11.8 NIVEAU ÉLIMINATION DE BLOCAGE LOCAL

Le résultat d'un routage incomplet du niveau précédant est ce que nous appellons un blocage local. L'élimination de blocage local est un système expert qui essaie d'éliminer ou de résoudre des problèmes de blocage (quand un chemin ne peut être router complètement) très localisé. Nous utiliserons l'approche de Joseph [4] dans son système BLOREC (Blockage Recovery Systeme).

Ce module s'occupera de faire la connexion de fils qui n'ont pas été routés complètement, sans toutefois modifier les routes qui ont déjà été routées. Ceci est fait en réarrangeant certains segments de réseaux déjà routés par des déplacements horizontaux et verticaux. Le travail effectué ressemble à celui d'un compresseur à déchets qui pousse les ordures dans le fond d'un sac. Le système expert pousse les routes afin de se libérer un espace suffisant pour le routage du fil bloqué.

2.12 IMPLANTATION

Pour un premier prototype, les niveaux 1, 4 et 5 sont laissés de côté. Le routeur global utilisé est une modification d'un routeur de la famille de Lee travaillant

sur un espace de recherche grossier à trois dimensions. Le routeur globali-local est aussi un routeur de la famille de Lee effectuant une recherche globale avec une grille de travail la plus fine possible, mais dans un espace restreint correspondant au routage grossier établi au niveau 2. Ce premier prototype, utilise qu'un seul algorithme par niveau. Nous allons vérifier s'il est possible de faire une hiérarchie dans l'espace de recherche.

L'implantation du système utilise une approche de programmation logique à l'aide de l'environnement de travail Arity Prolog. Cet environnement permet un prototypage rapide et offre un bon support à l'implantation de systèmes experts ou de système à base de connaissances.

2.13 MOTIVATION DE LA HIÉRARCHISATION

Nous allons tenter de mettre en évidence les raisons qui ont motivé notre approche hiérarchique. Pour se faire, nous allons comparer le nombre de cases qu'il faut visiter ou tester pour effectuer le routage complet d'une connexion avec un algorithme de LEE dans un premier temps, et avec un algorithme hiérarchique dans un second temps.

L'algorithme de Lee utilise une grille de travail très fine, en faite la grille la plus fine possible. Le nombre de voisins visités est donc proportionnel à l'aire de la région d'expansion. Pour une région d'expansion bornée par la plaque du circuit imprimé comme à la figure 2.13 a, nous obtenons une expansion proportionnelle à la surface du rectangle. Si la région d'expansion est complètement libre, comme à la figure 2.13 b, nous obtenons une expansion proportionnellement quatre fois plus grande.

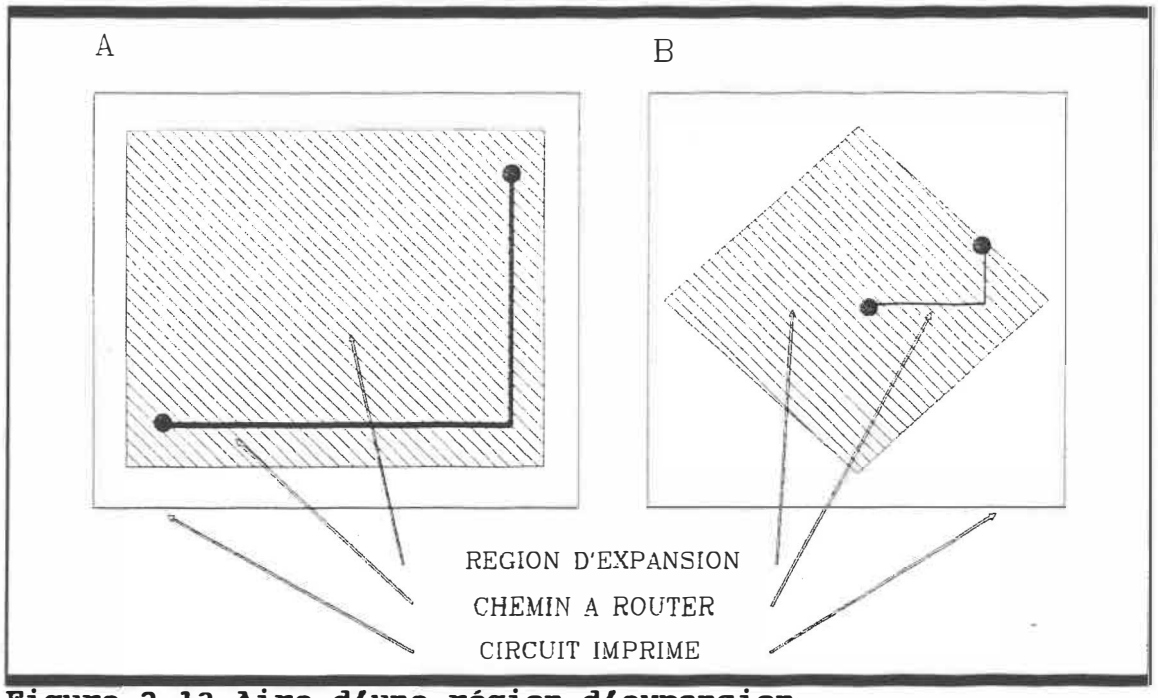


Figure 2.13 Aire d'une région d'expansion.

Définissons quelques termes

x,y : nombre de mailles de base entre le point de départ et le point d'arrivée.

nbx,nby : nombre de grosses mailles en x et en y entre le point de départ et le point d'arrivée.

Nv : nombre de voisins visités pendant l'expansion.

Avec ces quelques définitions, nous obtenons approximativement, pour un routage comme à la figure 2.13 a, un nombre de voisins visités défini par Nv = x X y tandis que pour le routeur hiérarchique nous obtenons:

$$Nv = (nbx \times nby) + \left(\frac{y}{nbx}\right) x + \left(\frac{x}{nby}\right) y$$

et nous avons les formules suivantes pour le routage de la figure 2.13 b. Premièrement, avec un algorithme de Lee nous avons:

$$Nv = 4(x \times y)$$

tandis qu'avec le routeur hiérarchique la formule serait:

$$Nv = 4(nbx \times nby) + \left(\frac{y}{nby}\right)x + \left(\frac{x}{nbx}\right)y$$

Nous pouvons apercevoir grâce à la courbe de la figure 2.14, que la hiérarchisation de l'espace d'expansion est très avantageuse pour un nombre approprié de grosses cases.

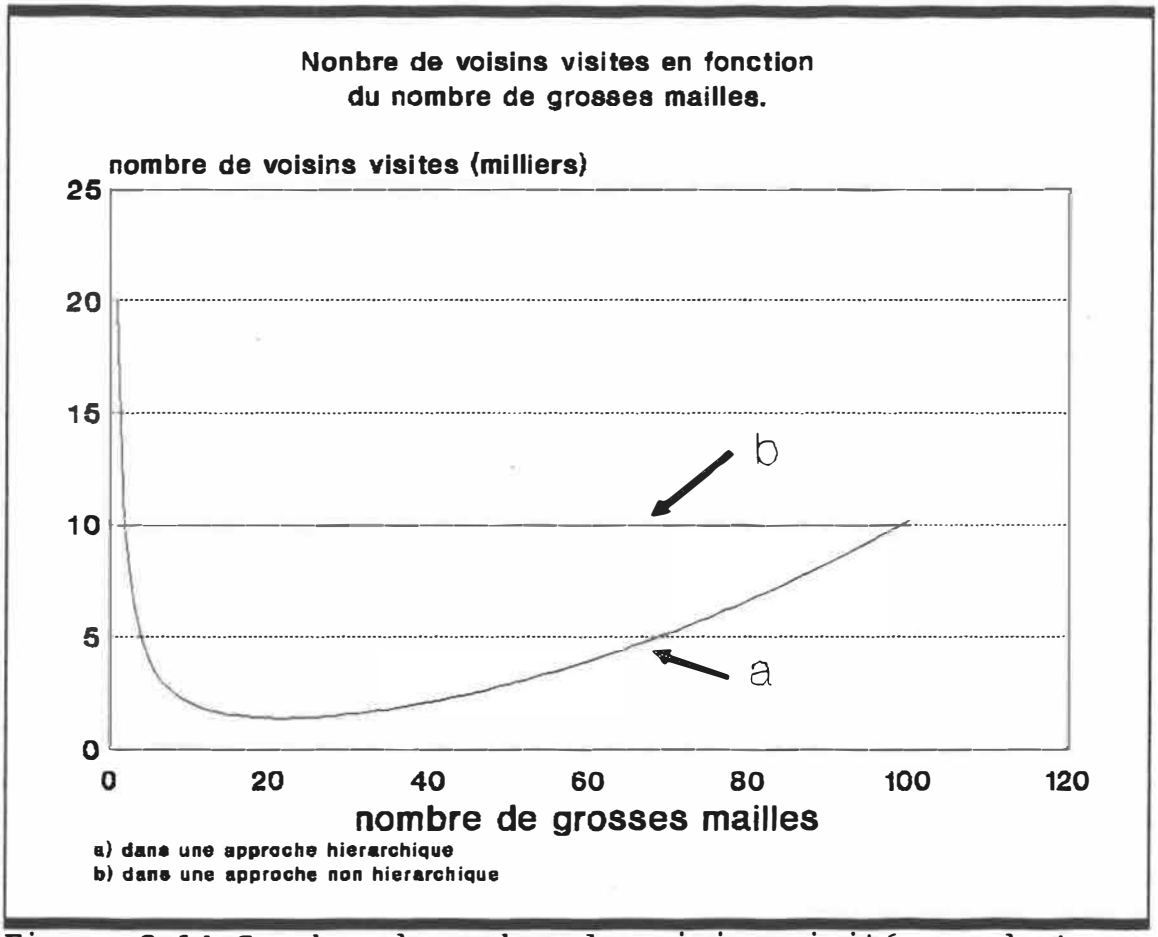


Figure 2.14 Courbes du nombre de voisins visités pendant une expansion.

Dans cette simulation du nombre de voisins visités durant l'expansion afin de router un chemin, nous avons fixé le nombre de grilles de base entre les points de départs et d'arrivées à 100. Nous avons aussi, pour fin de simplification, fixé le nombre de grosses cases en x égal au nombre de grosses cases en y. Le graphique représente donc le nombre de voisins visités lors de l'expansion en fonction du nombre de grosses cases avec nbx = nby et x = y = 100.

Nous voyons donc que pour un nombre de grosses cases d'environ 15 à 30, la hiérarchisation diminue grandement le nombre de voisins qu'il faut visiter lors de l'expansion permettant de router un chemin.

L'architecture hiérarchique est intéressante puisque lors du routage précis de bas niveau, nous visiterons les petites cases se trouvant uniquement dans les grosses mailles du chemin global grossier. En choisissant bien le nombre de grosses mailles utilisées, nous réduirons donc le nombre de voisins visités, et ainsi, le temps de calcul utilisé à produire l'expansion.

CHAPITRE 3

ENVIRONNEMENT DE TRAVAIL

3.1 CHOIX DE L'ENVIRONNEMENT DE TRAVAIL

Il est adéquat pour un environnement de travail, permettant de développer et d'implanter efficacement et rapidement la structure hiérarchique de routage, d'utiliser plusieurs langages de programmation.

Les langages de nouvelle génération comme le Prolog et le Smalltalk permettent un développement efficace des traitements symboliques. Ces langages dit d'intelligence artificielle ou de 5e génération, permettent de développer des prototypes plus rapidement. Ils offrent un bon support pour l'implantation de systèmes experts.

Pour les traitements utilisant beaucoup de calculs numériques, les langages procéduraux traditionnels sont plus performants. Ces langages sont le Pascal, le Fortran et le C.

Une première approche au développement se fera sur un ordinateur personnel IBM en utilisant le langage Prolog de Arity Corporation.

Arity Prolog est un environnement de programmation comprenant un compilateur, un interpréteur, un éditeur de texte, une base de données et un ensemble de librairies de prédicats.

L'environnement permet de créer facilement des programmes en Prolog qui utilisent des fenêtres de texte ainsi que des fenêtres "pop-up". Les fonctions permettant la création et la manipulation de fenêtres, sont très puissantes et complètes. Arity Prolog nous fournit aussi une librairie de fonctions permettant de contrôler une souris.

Il manque cependant dans l'environnement Arity Prolog 5.0X, une librairie graphique. Il n'est pas possible de dessiner ou d'exploiter les capacités graphiques haute résolution des ordinateurs. Les fenêtres sont uniquement des fenêtres de caractères, pas de graphique pixel par pixel. Les fonctions permettant de contrôler la souris fonctionnent aussi selon des coordonnées du mode d'affichage TEXTE soit 80 colonnes par 25 lignes. L'absence du support nécessaire pour exploiter les capacités graphiques haute résolution, que les

autres langages exploitent déjà, une lacune majeure de l'environnement de travail d'Arity.

Toutefois, Arity nous fournit un interface avec les langages C, Fortran. et Pascal. Il ne nous reste plus qu'à utiliser les fonctions graphiques d'un autre langage.

Les interfaces Arity Prolog à d'autres langages ont surtout été développés pour les langages de Microsoft. L'environnement Microsoft C 5.10 possède une librairie graphique rudimentaire. Elle ne permet pas les multiples fenêtres en mode graphique haute résolution, mais ceci n'est pas nécessaire et demeure toujours implantable à l'aide de plusieurs fonctions graphique de base.

Nous avons donc incorporé intégralement la librairie graphique de Microsoft C 5.10 à l'environnement de travail Arity Prolog 5.0X [30].

La bibliothèque de prédicats graphiques de haut niveau est un ensemble de prédicats permettant le multi-fenêtrage sur un écran en mode haute et basse résolution [30]. Elle supporte les adapteurs: CGA, EGA et VGA.

3.2 COMPILATION DE PROGRAMME PROLOG

L'environnement de travail Arity Prolog est principalement un environnement prolog interprèté. En utilisant l'interpréteur prolog, nous avons la possibilité, intéractivement, de modifier la base de données de l'environnement. Ceci nous permet de déverminer plus rapidement un programme Prolog. Sous le contrôle de l'interpréteur, nous avons aussi accès à un dévermineur symbolique.

Afin d'obtenir un programme exécutable à partir du système d'exploitation DOS, il est possible de compiler les programmes prolog. Certaines modifications doivent être faites dans les fichiers de codes sources pour réussir cette étape.

Toutes les clauses qui seront modifiées dynamiquement pendant l'exécution du programme par des instructions comme: **assert**, **retract** et **consult**, doivent être déclarées au début du fichier. Cette déclaration permet de spécifier qu'à l'exécution, l'accès à ces clauses se fera par le biais de la base de données et non du code compilé. Cette déclaration doit être rajoutée au début des fichiers utilisant ces clauses.

```
:- extrn NOM/ARITE:interp.
```

De plus, tous les prédicats qui sont définis dans un fichier et appelés dans un autre fichier devront être déclarés externes. Dans le fichier de définition du prédicats, nous devrons utiliser la directive suivante du compilateur pour ces prédicats:

```
:- public NOM/ARITE:far.
```

Dans le fichier qui utilise un prédicat déclaré dans un autre fichier, nous utilisons la directive suivante pour le même prédicat:

```
:- extrn NOM/ARITE:far.
```

Toutes les directives du compilateur devront être placés au début du fichier soit avant toutes déclarations de prédicat prolog.

Pour les programmes qui font un usage intensif des prédicats de modification dynamique de la base de données comme: **asserta**, **retract**, ... , la compilation n'accélère pas le temps des programmes. La compagnie Arity Prolog annonçait une diminution considérable du temps d'exécution de l'ordre

de 10. Ceci est un facteur complètement erroné.

3.3 ÉDITEUR SCHÉMATIQUE

Afin de pouvoir créer des tests pour le routeur hiérarchique, un éditeur schématique rudimentaire a été développé.

Nous pouvons voir au tableau 3.1 la liste de toutes les commandes de l'éditeur schématique.

Cet éditeur nous permet de créer des composantes DIP, SIP et des points isolés. L'éditeur fonctionne intéractivement par l'entremise des curseurs du clavier ou de la souris. Une fois les composantes créées, il est possible de les détruire ou de les déplacer. Le menu CONNEXION nous permet de définir les connexions entre les composantes qu'il va falloir router ultérieurement.

Le programme est écrit en Prolog. Ce langage est particulièrement bien structuré pour décrire et contrôler l'accès à plusieurs menus hiérarchiques comme celui du tableau 3.1.

Ce programme produit un fichier contenant l'identification de chaque composante, la position de chaque broche des composantes ainsi que la liste des connexions à router. De plus, il identifie pour chaque connexion le numéro de la broche et le nom de la composante pour les deux extrémités de chaque connexion.

TABLEAU 3.1 LISTE DES MENUS DE L'ÉDITEUR SCHÉMATIQUE

```
Aide
Rafraichissement du dessin
Lecture d'un design
Sauvegarde du design
Choix de composante
        DIP
                Choix du nombre de broches
                Choix de l'orientation
                Positionnement de la broche 1
                Valide le choix
        SIP
                Choix du nombre de broches
                Choix de l'orientation
                Positionnement de la broche 1
                Valide le choix
        Point
                Positionnement de la broche
                Valide le choix
 Connexions entre les composantes
        Nouveau réseau
                Ajoute une broche
                Enlève une broche
                Accepte réseau
        Modifie réseau
                Choisir réseau
                Ajoute une broche
                Enlève une broche
                Accepte réseau
        Détruire réseau
                Choisir réseau
```

```
Placement créer la plaque
        Dimension en X
        Dimension en Y
        Nombre de couches

Placement des composantes
        Placement composante
                Sélectionne la composante
                Sélectionne l'orientation
                Positionne la broche 1
                Accepte le placement
        Placement de connecteur
                Sélectionne le sip
                Sélectionne l'orientation
                Positionne la broche 1
                Accepte le placement
Routage
        Routage global
        Routage globali-local
Détruire une composante
Fin du programme
```

3.4 HIÉRARCHIE DE RECHERCHE

Pour vérifier la possibilité de hiérarchiser la recherche de chemins sur une plaque de circuit imprimé, nous allons utiliser uniquement deux niveaux. Le premier niveau est le routeur global à gros grain tandis que le second est le routeur globali-local.

Les deux routeurs sont des routeurs dérivés des algorithmes de la famille de Lee qui utiliseront le mode de fonctionnement itératif pour chacun des niveaux.

Le routeur global travaillera sur une grille large soit une grille plus grosse que la grille de base pour la fabrication de la plaque. Tous les tests que nous allons faire utiliseront une grille de travail de 25 MIL. En d'autres mots, il sera possible de placer sur la plaque un chemin à tous les 25 MIL. Le routeur global utilisera approximativement une grille de 150 MIL X 150 MIL ce qui correspond à 6 grilles de base en X et en Y.

La difficulté de résolution d'un problème de routage est inversement proportionnel à la grosseur de la grille de base. Par conséquent, si la grille de base est très petite, nous aurons la possibilité de placer beaucoup de chemins sur la plaque. Ceci nous donnera plus de possibilités afin de router tous les chemins. En contre partie, plus nous avons de possibilités, plus la recherche de chemins sera longue.

Dans les deux chapitres qui suivent, nous verrons les détails du routeur global à gros grain ainsi que ceux du processus itératif à chaque niveau.

CHAPITRE 4

ROUTEUR GLOBAL À GRILLE LARGE

4.1 ROUTEUR DE LEE

Le routeur global à grille large est une généralisation des algorithmes de la famille de Lee. Ces algorithmes de même que celui utilisé pour le second niveau de la hiérarchie, sont des algorithmes globaux. Ils effectuent une recherche globale pour trouver un chemin.

Ces algorithmes utilisent une grille de travail la plus fine possible. Pour une grille de travail de 25 mil., la plaque de circuit imprimé sera complètement divisé en carreaux de 25 mil. en X et en Y. Pour une plaque de 12 pouces par 12 pouces, nous obtenons un espace de recherche de 230,400 petites cases, ce qui est considérable.

Les algorithmes de la famille de Lee ont le désavantage d'utiliser beaucoup de mémoire pour représenter chacune de ces petites cases élémentaires. De plus, ils nécessitent beaucoup de temps de calcul pour la recherche de cases libres en vue de l'établissement d'un chemin sur le

circuit imprimé. Ils ont cependant l'avantage d'effectuer une recherche relativement simple qui assure de trouver, s'il y a lieu, le plus court chemin.

Le routeur cherche un chemin en vérifiant si une case jouxtant celles déjà vérifiées est libre ou non. Ensuite il commence sa recherche par le point de départ en lui affectant l'étiquette 1 (Voir figure 4.1 a). A partir de cette case étiquettée 1, nous vérifions toutes les cases jouxtant celle-

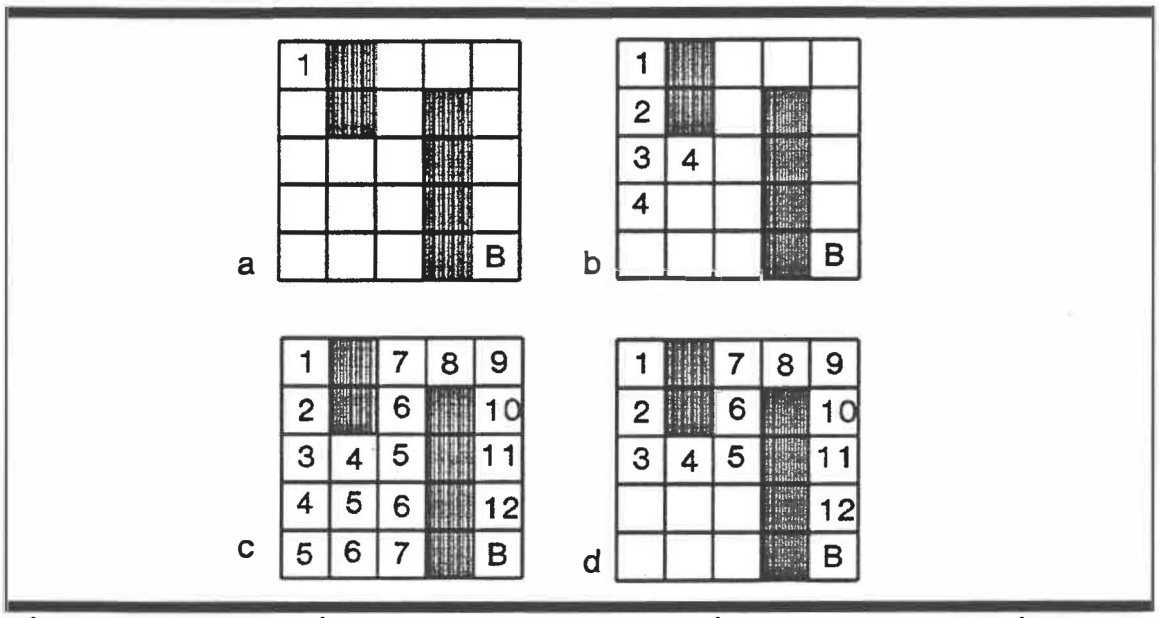


Figure 4.1 Fonctionnement d'un algorithme de la famille de Lee.

ci. Les cases voisines, (en haut, en bas, à gauche et à droite) sont vérifiées. Si elles sont libres, elles seront étiquettées d'un 2 (Voir figure 4.1 b). L'expansion de voisin en voisin se poursuit en prenant les cases 2 comme

case de référence. Cette expansion se poursuit jusqu'à ce que nous ayons atteint le point d'arrivée (figure 4.1 c). Finalement, le chemin est établit en parcourant les cases par ordre décroissant de leur étiquette en commençant au point d'arrivée (figure 4.1 d). Si un chemin existe, ce sera le plus court chemin qui existe entre les deux points.

4.2 LE NOUVEAU ROUTEUR GLOBAL.

Le nouveau routeur global à grille large utilise le même principe d'expansion de grille en grille, que les algorithmes de la famille de Lee. Toutefois, certaines modifications nous permettent d'obtenir un algorithme plus général dans la manière d'effectuer la recherche.

Premièrement, l'algorithme permet maintenant d'effectuer une recherche tri-dimensionnelle. En effet, contrairement aux routeurs utilisant un algorithme de la famille de Lee, lesquels cherchent un chemin sur un plan, le routeur global à grosse grille cherche dans plusieurs plans. Ce type de recherche est bien adapté au design de circuits imprimés à plusieurs couches tels qu'utilisés de nos jours. L'expansion est tri-dimensionnelle car nous vérifions à partir d'un point d'expansion, les voisins à gauche, à droite, en haut, en bas, sur la couche du haut et sur la

couche du bas.

Comme nous l'avons souligné au chapitre précédent, le routeur utilise une grille de travail plus grosse que la grille de base. Pour chaque maille de la grille de travail correspond une quantité physique de chemins qu'il sera possible de placer dans les directions X et Y, ainsi qu'un nombre de traverses. En agrandissant chacune de ces mailles, le nombre de cases qu'il faudra examiner lors de l'expansion de voisin en voisin diminue et conséquemment le temps global pour la recherche d'un chemin aussi. Il y a cependant un autre côté à la médaille. Plus la grille de travail du routeur global sera grosse, plus la recherche d'un chemin par les routeurs plus bas dans la hiérarchie sera longue.

Le travail fait par le routeur global à gros grain se divise en trois étapes distinctes:

1- L'initialisation des données de base et la vérification des contraintes de départ,

2- L'expansion de case en case en partant du point de départ jusqu'au point d'arrivée,

3- Décision et affectation de l'endroit ou reposera le chemin grossier.

Avec cette nouvelle méthode de travail, la recherche d'un chemin n'est plus exhaustive comme le sont les recherches avec les algorithmes de la famille de Lee. Cependant, nous avons réduit considérablement l'espace de recherche.

4.3 GRAPHÈMES DE RECHERCHE

Nous définissons des graphèmes de base nous permettant d'effectuer une recherche adéquate dans chacune des grosses cases. Puisque dans chacune des grosses mailles de recherche globale, il est possible de placer plusieurs chemins en X, en Y et plusieurs traverses, nous avons besoin de ces graphèmes afin d'empêcher l'établissement de chemins incompatibles.

Les graphèmes sont composés d'une série de trois cases juxtaposées. C'est une combinaison tri-dimentionnelle représentant toutes les possibilités d'agencement de cases pour former une portion de chemin. La liste des graphèmes ainsi que leur nom est présentée à la figure 4.2.

A chaque graphème est associé trois poids en nombre de lignes en X, en Y et en nombre de traverses. Ces poids permettent de tenir compte de l'espace physique qu'occupe

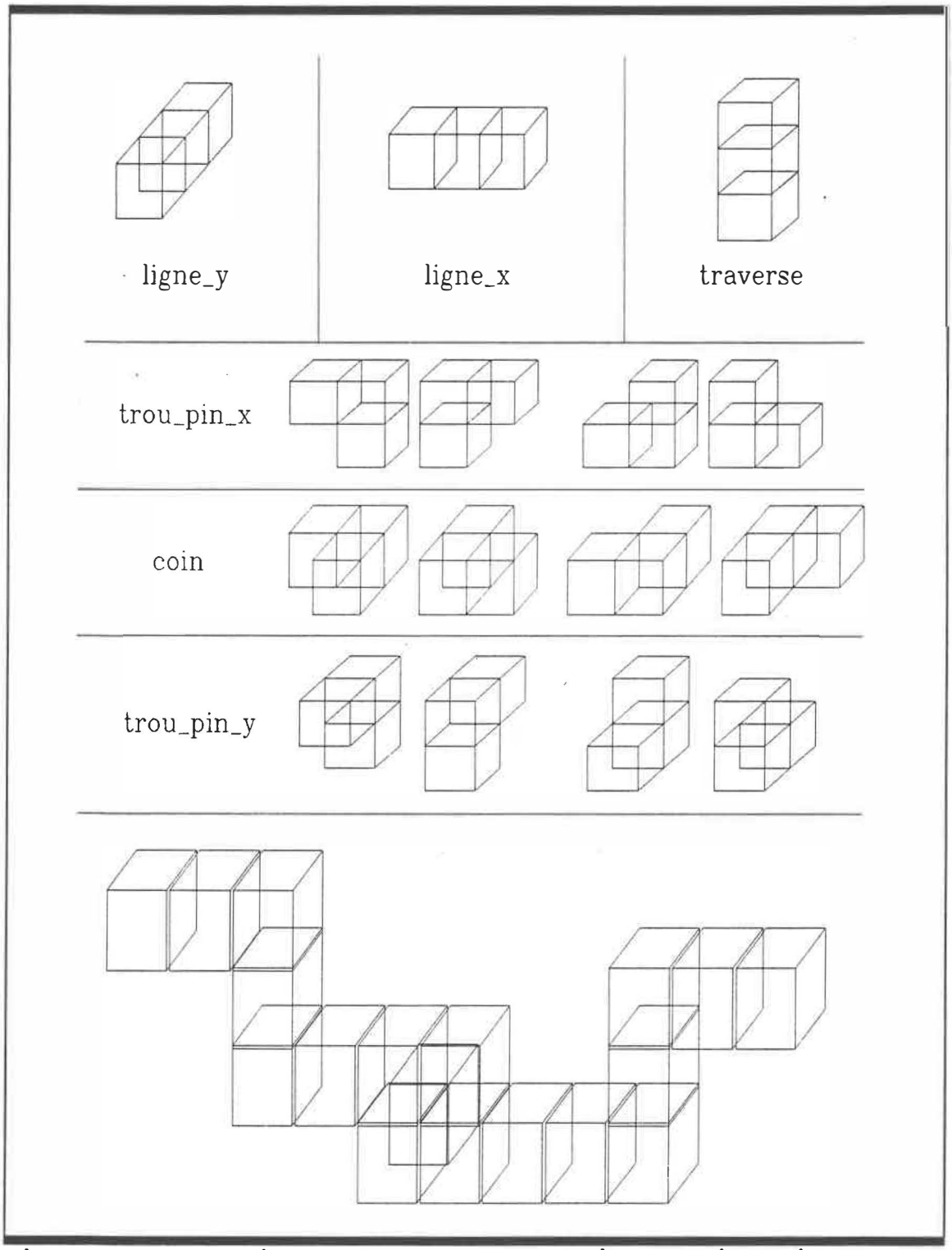


Figure 4.2 Graphèmes de base et chemin représenté par une suite de graphèmes.

chaque graphème sur un circuit imprimé. Voir tableau 4.1.

graphème	Poids		
	chemin x	chemin y	traverse
Ligne_x	1	0	0
Ligne_y	0	1	0
Trou_pin_x	0.5	0	1
Trou_pin_y	0	0.5	1
coin	0.5	0.5	0
traverse	0	0	1

Tableau 4.1 Poids pour chaque graphème

L'établissement des poids est fait de manière ad hoc. Ils ne sont pas exacts puisque lors du routage global à grosses mailles, l'espace exact qu'occupera le chemin dans chacune des cases est inconnu. Nous pouvons voir à la figure 4.3 qu'un même graphème peut utiliser plus ou moins d'espace sur le circuit imprimé.

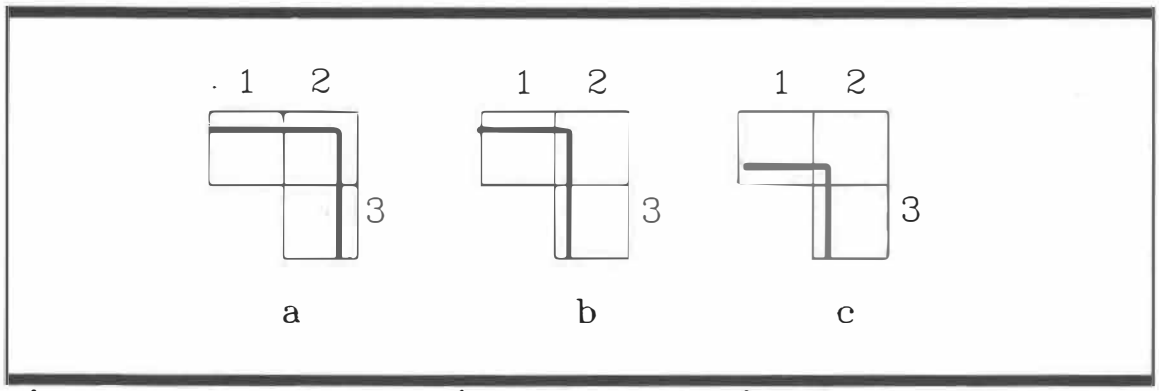


Figure 4.3 Espace occupée par le graphème COIN.

Nous voyons que sur la figure 4.3 a, dans la case 2, nous utilisons presque une ligne au complet en X de même qu'en Y. A la figure 4.3 b nous utilisons une ligne en Y et très peu d'espace en X. Finalement, pour la figure 4.3 c, la case 2 n'utilise que très peu d'espace en X ainsi qu'en Y. Ces trois configurations sont possibles mais nous ne saurons pas laquelle est la bonne avant d'avoir effectué le routage détaillé. C'est ce qui explique pourquoi, lors de la vérification d'espace physique pendant l'expansion, nous allons attribuer des poids de 0.5 chemin en X, 0.5 chemin en Y et 0 traverse pour un graphème COIN.

La vérification d'espace physique se fait en deux étapes puisque nous identifions les graphèmes seulement lors de l'expansion de voisin en voisin. Lors de l'expansion de la case 1 à la case 2, nous vérifions qu'il y ait au moins 0.5 ligne en X dans la case 2. De plus, lors de l'expansion de la case 2 à la case 3, nous vérifierons également que la case 2 possède de l'espace pour au moins 0.5 ligne en Y. Nous pourrons donc en déduire que le graphème COIN est possible à cet endroit du circuit imprimé.

4.4 STRUCTURE DE DÉPART

Le routeur global à grosses mailles utilise trois types d'informations pour connaître la configuration physique du circuit à router. Ces trois données d'entrée définissent les composantes, les réseaux de connexion et enfin les paramètres de routage.

La description des composantes s'effectue grâce à une structure complexe ressemblant à celle décrite au chapitre 2. Cette structure est néanmoins plus simple. Nous avons éliminé les caractéristiques AC et DC qui ne sont utiles que pour des simulations du comportement des composantes. Toutes les mesures sont introduites en grandeur réelle dans la structure soit en mil. La structure comporte quatre éléments:

- numéro d'identification
- nom de la composante
- dessin du boitier
- emplacement de chaque broche.

```
composante(1,nom(250,150,a),dessin(15,315,785,85),
           configuration([(50 ,50),(150,50),(250,50),
           (350,50),(450,50),(550,50),(650,50),(750,50),
           (750,350),(650,350),(550,350),(450,350),
           (350,350),(250,350),(150,350),( 50,350)])).
```

Les connexions entre les composantes sont spécifiées par une structure plus simple identifiant le nom de la composante et le numéro de la broche des deux extrémités de la connexion.

```
reseaux(1,nom(x,y,id),configuration([(a,1),(b,12)])).
```

Nous avons de plus, une liste de paramètres permettant de contrôler le routage.

```
nb_carreau_en_x(11).
nb_carreau_en_y(5).
couche (1,a).
couche (2,x).
grille (50).
nb_de_couche(2).
coor_max_x(1800).
coor_max_y(750).
coor_min_x(0).
coor_min_y(0).
```

Les deux paramètres **nb_carreau_en_x** et **nb_carreau_en_y** permettent de spécifier le nombre de grosses mailles qui représenteront le circuit imprimé. Les dimensions du circuit sont spécifiées par les quatre paramètres: **coor_max_x**, **coor_max_y**, **coor_min_x** et **coor_min_y**.

Nous pouvons contrôler l'expansion en spécifiant par les faits **couche**, l'orientation des chemins qu'il y aura sur

chaque couche. Chaque couche peut permettre:

- uniquement des chemins en x
- uniquement des chemins en y
- des chemins en x et en y.

Nous avons donc un fait pour chaque couche avec un des paramètres x, y ou a, pour un routage dans les deux directions. Le fait **nb_de_couche** nous permet de connaître le nombre de couches permises pour le routage.

Enfin, le fait **grille** nous donne le pas de la grille de base.

4.5 PRÉPARATION DES DONNÉES DE ROUTAGE

Les données de base pour le routage sont préparées durant la phase d'initialisation. Les données sont établies uniquement au départ, tandis que les deux autres étapes se reproduisent pour chaque fil qu'il faut router.

La phase d'initialisation cherche et établit les emplacements de la grille de base qui ne pourront pas être utilisés pour une traverse ou pour y placer un chemin. Ces endroits sont déterminés par des contraintes. Par examples,

- il ne doit pas y avoir de traverses dans le voisinage immédiat d'une broche de composante. Cette contrainte est dû à un manque d'espace physique parce que le beigne de soudure est plus large qu'un carreau de la grille de base.

- il ne peut pas y avoir de traverses sous les composantes. Cette contrainte peut être imposée mais n'est pas d'usage courant.

- dans une grosse case et suivant une direction X ou Y, si nous avons au moins deux cases de base illégale, nous éliminerons une ligne dans la direction de la ligne passant par les deux cases de base.

Pour trouver les cases de la grille de base, nous visitons toutes les fines cases qui contiennent des broches de composantes et appliquons à cet endroit un masque d'illégalité. Voir figure 4.4.

Dans le masque d'illégalité, qui est symétrique, le centre représente une broche de composante. Les points P (pad) identifient le beigne de soudure de la connexion. Les points SC (sans chemin) identifient les points illégaux à

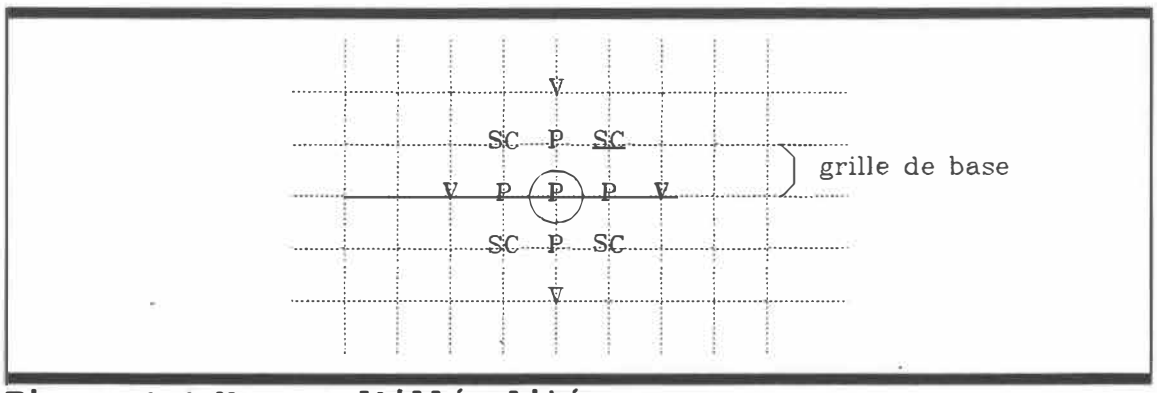


Figure 4.4 Masque d'illégalité.

l'établissement de chemins ou de traverses. Enfin, les points V (vias) représentent les points ou nous ne pouvons pas mettre de traverses.

Une fois ces cases illégales établies, la section d'initialisation crée une matrice de blocage. Chacun des blocs de la matrice de blocage représente la disponibilité physique d'une case à grosse maille.

La matrice de blocage est construite en gardant pour information, le nombre de chemins en x, en y et le nombre de traverses disponibles dans chaque grosse case, avant de commencer le routage. Dans la grosse case de la figure 4.5, nous pouvons identifier quatre lignes en x de libre et deux lignes en y. Les lignes libres sont celles qui n'ont aucune maille de base identifiée par un P ou un SC. Cette grosse maille comporte aussi 28 traverses. Nous pouvons remarquer que dans cette grosse case, il reste 28 grilles de base qui

ne portent aucune marque d'illégalité.

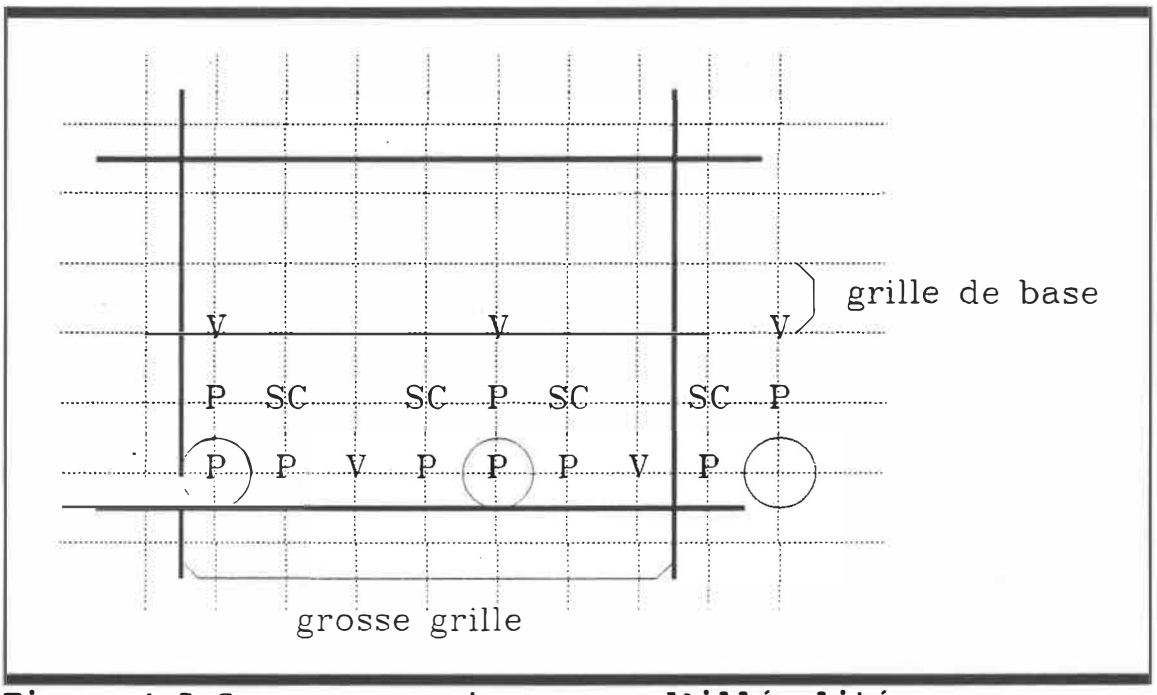


Figure 4.5 Grosse case et masques d'illégalité.

Cette matrice de blocage représente l'espace libre dans chacune des cases durant le routage de tous les fils. Chaque placement d'un fil sur la plaque modifiera la matrice de blocage pour tenir compte de l'espace occupé par ce nouveau fil.

De plus, nous gardons aussi pour chacun des blocs de la matrice de blocage, la liste des fils qui ont été placés dans chacune des cases.

Enfin, durant cette phase nous construisons la liste des connexions qu'il faut router. Cette liste est en ordre

croissant de la distance de Manhattan entre ses deux extrémités. Nous routons les petits fils en premier.

4.6 MODE D'EXPANSION

Nous présentons maintenant l'algorithme utilisé afin d'effectuer l'expansion de voisin en voisin.

```
-Commencer avec la case de départ
Répéter
    Pour chaque direction soit: haut, bas, gauche, droite,
    couche en haut et couche en bas.
        Si le bloc est libre pour un graphème
            Si la case n'existe pas
                -Créer une case identifiant son
                 emplacement, son numéro d'expansion et
                 sa direction d'expansion.
            Sinon
                Si c'est le même numéro d'expansion
                    -Rajouter la direction d'expansion
                     à celles existantes.
    Reprendre successivement tous les voisins avec les
    numéros d'expansion les plus élevés.
Tant que le point final n'est pas atteint ou qu'il n'y ait
plus d'expansion.
```

L'algorithme utilise principalement deux prédicats récursifs. Le premier permet de produire des itérations pour

tous les voisins en augmentant successivement l'étiquette représentant le numéro d'expansion. Le second prédicat commande la vérification des voisins jouxtant une case de référence. Les six directions d'expansion sont identifiées par les symboles suivants:

d: expansion vers la droite
g: expansion vers la gauche
h: expansion vers le haut
b: expansion vers le bas
cv: expansion vers la couche supérieure
cd: expansion vers la couche inférieure

La récursivité des prédicats cesse lorsque nous avons atteint le point d'arrivée ou lorsqu'il n'y a plus d'expansion. Un arrêt d'expansion résulte d'une impossibilité de passage entre le point de départ et le point d'arrivée.

Lors de vérification de l'espace libre pour l'établissement d'un graphème, nous vérifions les informations de la matrice de blocage. Au cours de l'expansion, une matrice d'expansion est créée. Il y a une correspondance directe entre chaque élément de la matrice de blocage et ceux de la matrice d'expansion.

Si l'élément de la matrice de blocage, correspondant au voisin à vérifier, est libre, se référant aux poids de chaque graphème, nous vérifions si l'élément de la matrice d'expansion existe. Si cet élément n'existe pas, nous n'avons donc jamais visité ce voisin. Il faut donc le créer. Par contre, si cet élément existe, la case a déjà été visitée et nous devons vérifier, d'une part, s'il s'agit d'une solution alternative ou d'autre part s'il s'agit d'une solution entrainant un chemin plus grand. Afin de déceler une solution alternative, nous vérifions le numéro d'expansion. Le numéro d'expansion du voisin visité doit être le même que le numéro de l'expansion courante. Nous pouvons affirmer, uniquement par cette condition, que nous traitons une solution alternative. Par conséquent, nous retiendrons la direction d'expansion dans l'élément de la matrice d'expansion.

Nous créons un élément de la matrice d'expansion en utilisant les coordonnées en X et en Y de la case en plus du numéro de la couche. Nous retiendrons de plus, une liste comprenant comme élément l'identification et la direction de tous les ancêtres de la case d'expansion courante.

Lorsque nous visitons la case (2,2,1), voir figure 4.6, qui n'avait jamais été visitée, nous la créons puisqu'elle était libre. Nous créons un fait:

case(2,2,1,3,[[1,2,1,d]])

qui signifie que la case (2,2,1) est un voisin de 3e expansion qui a un ancêtre; la case (1,2,1). Cette descendance provient d'une expansion dans la direction d.

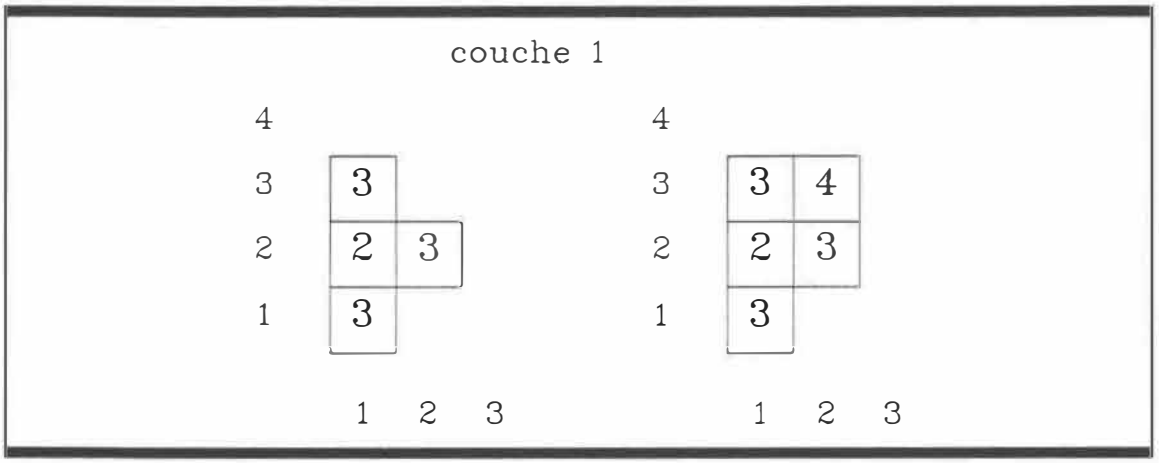


Figure 4.6 Visite de cases.

À la 4e expansion, nous obtiendrons une autre case:

case(2,3,1,4,[[2,2,1,h],[1,3,1,d]])

Cette case possède deux ancêtres:

- la case (2,2,1) ancêtre de direction h et
- la case (1,3,1) ancêtre de direction d.

Avec ces faits, nous gardons en mémoire tous les chemins possibles à partir d'un point d'expansion.

4.7 CONTRAINTES ADDITIONNELLES

Certaines contraintes additionnelles ont été ajoutées afin de forcer une distribution uniforme des chemins sur la plaque du circuit imprimé.

Nous empêchons l'établissement de graphèmes de type: **ligne_x**, **ligne_y**, **trou_pin_x** et **trou_pin_y** dans les grosses cases qui contiennent des broches de composantes.

4.8 DÉCISION DU CHEMIN GROSSIER

Du point d'arrivée au point de départ

> Construire la liste de toutes les cases qui forment le chemin en utilisant les ancêtres de chaque case.

Pour toutes les cases du chemin trouvé

> Modifier la matrice de blocage afin de tenir compte du nouveau chemin pendant les routages subséquents.

Un prédicat récursif permet de trouver les cases qui constitueront le plus court chemin entre les deux points d'extrémités. Le prédicat utilise comme référence,le point d'arrivée. De ce point de référence, il utilise l'information dans la matrice d'expansion pour connaître l'ancêtre du point de référence. Une fois l'ancêtre obtenu, il devient le

nouveau point de référence. Tous ces points de références sont récursivement insérés dans une liste qui constituera le chemin grossier.

Il arrive qu'il y ait plus d'un ancêtre à un point de référence. Tous ces ancêtres constituent des solutions alternatives qui produiront des chemins de la même longueur. Nous choisirons l'ancêtre qui se propage dans la même direction que ses descendants. Ce qui a pour effet de choisir un chemin qui aura le moins de changement de direction possible.

À la figure 4.7 nous pouvons remarquer que la recherche d'ancêtres nous offre deux possibilités à la case "a". Nous pouvons remarquer également que la direction de propagation des descendants de la case "a" était vers le haut. Nous allons donc prévilégier la solution utilisant la case "b" qui se propage aussi vers le haut, plutôt que la solution utilisant la case "c" qui entraîne un changement de direction vers la gauche.

Nous pouvons exprimer le même choix en disant que nous prévilégions les graphèmes **ligne_x** et **ligne_y** par rapport aux graphèmes: **coin**, **trou_pin_x**, **trou_pin_y**.

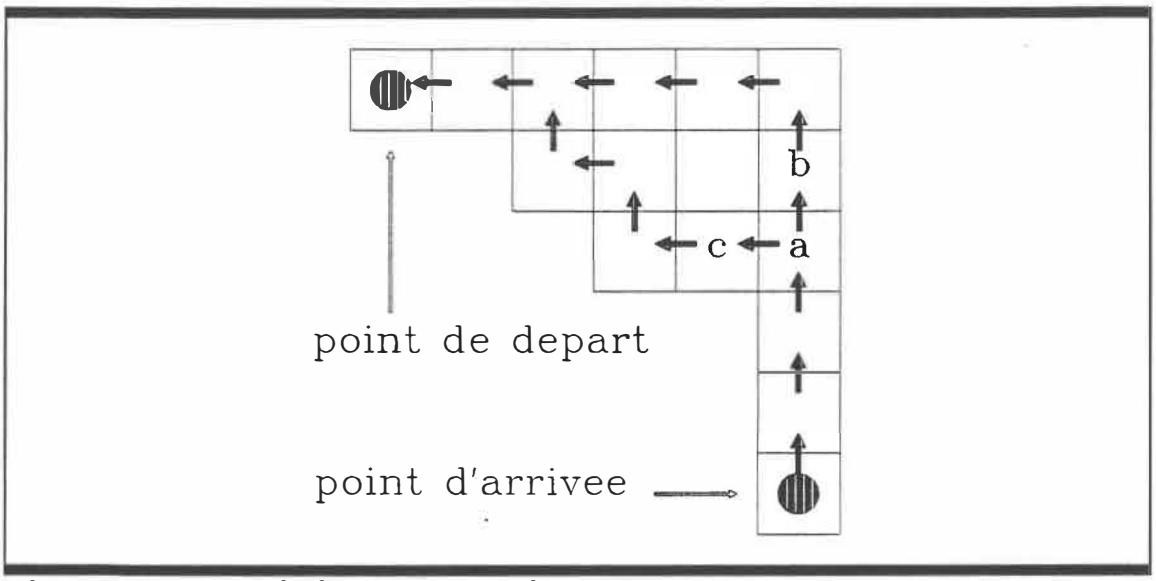


Figure 4.7 Décision du chemin.

Une fois que toutes les cases constituant le chemin grossier sont déterminées, nous connaissons donc la série de graphèmes représentant ce chemin. Nous utilisons chacun des graphèmes ainsi que les poids, représentant l'espace physique occupé, pour modifier la matrice de blocage. À chaque élément de la matrice de blocage auquel est associé un graphème, nous déduirons de sa capacité physique, les poids associés au graphème. Nous pourrons ainsi, lors de routages subséquents, tenir compte des connexions qui ont déjà été routées.

4.9 FORME DES RÉSULTATS

Nous obtenons trois résultats très importants: la liste des cases constituant le chemin, la matrice de blocage

et la matrice d'expansion.

La liste des cases constituant le chemin grossier est présenté par un fait, de la façon suivante:

chemin(5,[(1,1,1),(1,2,1),(1,2,2),(1,3,2),(1,4,2)]).

Le premier élément du fait identifie le numéro de la connexion. Le second élément représente la liste dont chacune des constituantes identifie chacune des cases grossières formant le chemin. Chaque identification de case spécifie la position en X, la position en Y et le numéro de la couche.

Comme nous l'avons mentionné à la section précédente, la matrice de blocage reste active pour le routage de toutes les connexions d'un même circuit imprimé. Elle prend note de tous les chemins qui ont déjà été routés. De plus, chacun des éléments de la matrice de blocage tiendra à jour une liste de toutes les connexions qui traversent la case grossière lui correspondant.

La matrice d'expansion est détruite après chaque routage complet d'une connexion. Si le routage n'a pu être complété pour un chemin, la matrice d'expansion sera utilisée

pour contrôler le reroutage du chemin. Avec cette matrice, nous connaissons les régions de blocage qui correspondent aux extrémités de l'expansion.

4.10 AVANTAGES ET INCONVÉNIENTS DE PROLOG

L'utilisation du langage Prolog nous a permis de développer un premier prototype assez rapidement. C'est un langage qui est particulièrement bien approprié pour la représentation et la manipulation des graphèmes et des poids qui leurs sont associés.

Puisque le langage possède son propre moteur d'inférence, il est relativement facile de construire et de manipuler la matrice de blocage. Il est aussi un outil puissant pour la manipulation de liste. Toutefois, ce langage comprend certains désavantages, puisqu'il possède son moteur d'inférence. En d'autres mots, c'est lui qui contrôle la recherche dans la base de données. Enfin, lors de l'expansion, il est très facile d'effectuer, par retour arrière, des traitements sur toutes les cases d'expansion qui possède le même étiquette. Cependant, pour s'assurer qu'il a obtenu toutes les cases portant le même étiquette, le moteur d'inférence doit consulter la base de données au complet. Par conséquent, ce traitement devient de plus en plus long au fur

et à mesure que l'expansion se poursuit.

Nous remarquons principalement ce problème lorsque nous vérifions si une des grilles de base est libre ou non lors du calcul de la capacité initiale d'une case grossière. Cette vérification s'effectue en appelant la clause:

not(sans_pin(X,Y,C,_)).

Nous vérifions qu'il n'y a pas de masque d'illégalité pour la grille de base (X,Y,C). Pour vérifier cette affirmation, le moteur d'inférence doit fouiller toute la base de données pour s'assurer qu'il n'y a aucun fait **sans_pin(X,Y,C,_)**. Cette fouille est atrocement couteuse en temps d'exécution puisque nous devons l'effectuer pour chaque grille de base du circuit imprimé.

Pour accélérer drastiquement cette recherche, il est possible avec le compilateur Prolog de créer et d'utiliser des variables C à l'intérieur de clauses Prolog. En effet, en créant un tableau d'entier, il est possible de représenter toutes les grilles de base qui sont illégales et ainsi d'effectuer une vérification directe sans fouille exhaustive de la base de données, de l'illégalité ou non d'une grille de base.

Cette solution, quoique beaucoup plus rapide, a le désavantage de prendre plus de mémoire et nous oblige à compiler le code Prolog. Nous perdons donc ainsi tous les avantages de l'interprétation de code Prolog.

CHAPITRE 5

PROCESSUS DE MÛRISSEMENT

Le processus de mûrissement fait partie du module de conception physique. C'est la partie du contrôle de la hiérarchisation qui s'occupe de contrôler l'enlèvement et le reroutage de chemins à chaque niveau de la hiérarchie.

Lorsque le routage grossier d'un fil a bien fonctionné, il n'y a pas de mûrissement. Nous passons simplement au fil suivant dans la liste des connexions à router. Des décisions importantes sont prises uniquement lorsqu'un fil n'a pu être routé a un niveau grossier. Dans ce cas, nous devrons décider des actions à prendre afin de réussir le routage de la connexion non routée ce qui aura pour effet, de faire progresser le routage du circuit imprimé.

Le processus de mûrissement s'effectue en deux étapes. Premièrement, nous chercherons les meilleurs endroits pour continuer le routage qui a été bloqué. Ensuite, à la seconde étape nous déciderons des fils qu'il faut enlever et rerouter afin de permettre le routage complet de celui qui est bloqué.

5.1 DESCRIPTION DES STRUCTURES DE DÉPART

Principalement, nous utilisons les mêmes structures que celles du routeur grossier en plus d'une nouvelle liste tenant compte des fils restant à router à chaque fois que nous activons le processus de mûrissement.

Nous utilisons la matrice d'expansion comme matière première du processus de mûrissement. Cette matrice, décrite au chapitre 4, représente l'état d'avancement du routage grossier une fois que la connexion a été déclarée bloquée et impossible à router dans les conditions présentes.

La matrice de blocage sera utilisée afin de modifier l'état présent du routage. Nous libérerons dans cette matrice, les espaces utilisés par certains chemins déjà routés que nous enlèverons pour permettre le routage de celui actuellement bloqué.

Les chemins que nous déciderons d'enlever afin de faire progresser le routage du circuit imprimé, seront remis dans la liste des chemins qui n'ont pas encore été routés.

La nouvelle liste: **liste_rip_up**, représente les fils qui restent à router à chaque fois qu'un fil a été bloqué pendant

le routage grossier. Cette liste représente la même chose que la liste de réseaux à router sauf qu'elle ne sera pas modifiée. Une nouvelle liste sera créée à chaque fois qu'un chemin sera bloqué. Nous utiliserons cette liste pour empêcher les cycles dans le processus de l'enlèvement et de reroutage de fils. La première liste est créée avant d'entrer dans le processus de mûrissement et elle contient la liste initiale de tous les réseaux à router sur le circuit imprimé.

5.2 RECHERCHE DES RÉGIONS DE BLOCAGE

A cette étape du processus de mûrissement, nous cherchons les cases de l'expansion qui sont les endroits les plus appropriés pour poursuivre le routage bloqué.

Algorithme:

Pour toutes les cases d'expansions

-Calculer la distance entre la case et le point d'extrémité du chemin à router.

-Calculer l'indice de la région idéale pour la poursuite du routage.

-Mettre toutes les cases dans une liste en ordre croissant, selon leur indice de région idéale.

La distance calculée, est la distance de Manhattan entre les coordonnées de la case d'expansion et les coordonnées de la case du point d'extrémité de la connexion bloquée.

X,Y coordonnées de la case d'expansion

Fx,Fy coordonnées de la case d'extrémité.

Distance manhattan = abs(X-Fx) + abs(Y-Fy)

Cette distance permet de calculer l'indice de région idéale à la poursuite de l'expansion. Nous n'utilisons pas uniquement la distance de Manhattan comme indice de région idéale parce qu'elle ne reflète que les cases les plus proches du point final qui ne sont pas nécessairement les meilleures cases pour continuer l'expansion comme l'indique la figure 5.1.

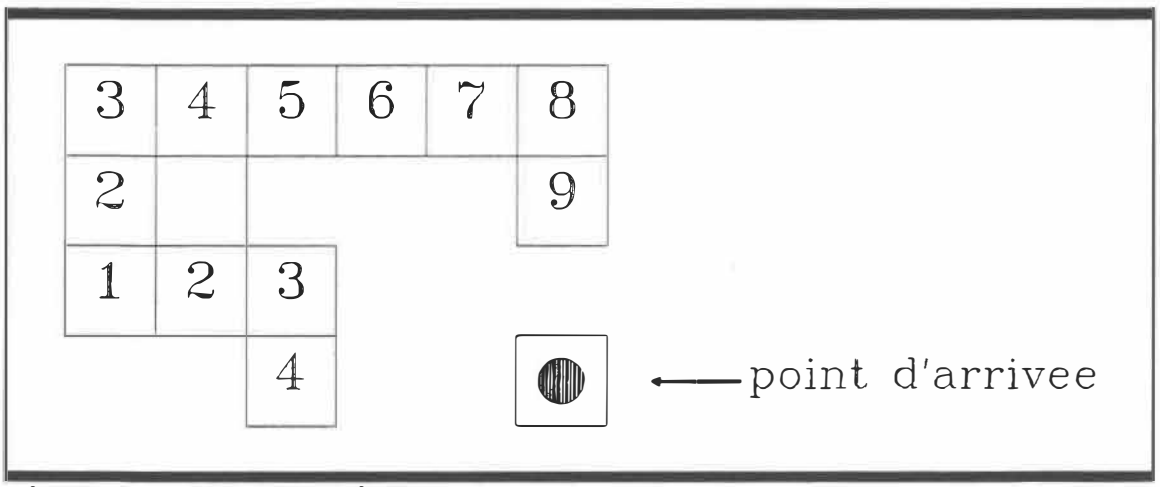


Figure 5.1 Expansion apres un blocage.

Pour cet exemple, la case 9 est à une distance de 3 du point final tandis que la case 4 est à une distance de 4. Nous aurions choisi la case 9 comme région idéale en utilisant uniquement la distance de Manhattan. Ce n'est pas ce que nous voulons. Nous voulons trouver, non seulement les cases qui se sont le plus rapprochées de la case finale, mais le plus rapidement possible. Il nous faut donc pénaliser les détours. Ainsi, nous calculerons l'indice de région idéale comme suit:

x,y : coordonnées de la case d'expansion.

Fx,Fx : coodonnées de la case d'arrivée.

E : Étiquette d'expansion d'une case.

Indice de région idéale = 2 X (abs(Fx-X) + abs(Fy-Y)) + E.

Le terme E permet de tenir compte de l'allongement inutile de l'expansion. Le facteur 2 permet de donner la priorité à la distance de Manhattan. Selon l'exemple de la figure 5.1, le chemin se terminant à l'expansion 9 obtient un indice de région idéale de 2(3) + 9 = 15 tandis que celui de l'expansion 4 obtient un indice de 2(4) + 4 = 12. Les cases ayant l'indice de région idéale le plus faible seront choisies en priorité. La case 4 sera donc identifiée comme étant une meilleure région pour poursuivre l'expansion de la connexion bloquée que la case 9.

Une fois les indices de région idéale calculés pour chacune des cases d'expansion, nous construisons une liste identifiant chacune des cases de l'expansion. Cette dernière est ordonnée selon l'indice de région idéale.

Pour des raisons de limitation dans la grosseur d'une liste sous l'environnement Arity Prolog, cette liste contient un maximum de 120 identifications des meilleures cases d'expansion pour la poursuite du routage. De plus, toujours sous l'environnement Arity Prolog, une clause, un prédicat ou une liste ne peut dépasser individuellement 4K. Cette contrainte nous force à réduire le nombre de cases d'expansion dans la liste à un maximum de 120.

5.3 DÉCISION DE CHEMIN À ENLEVER

Le calcul du facteur, permettant d'identifier les régions les plus propices à la poursuite de l'expansion, est une des étapes du processus de mûrissement d'un circuit imprimé. L'algorithme du processus au complet se présente ainsi:

- Faire la liste des cases d'expansion par rapport à leur indice de région idéale.
- Trouver les fils qui ont bloqué l'expansion.

Pour chacun des fils qu'on veut enlever,

- Libérer l'espace occupée par ce fil dans la matrice de blocage.
- Remettre ce fil dans la liste des connexions à router.

- Remettre le fil bloqué au début de la liste des connexions à router.
- Détruire la matrice d'expansion.
- Continuer le routage des connexions non routées.

Chacune de ces étapes est relativement simple. Cependant, l'étape pendant laquelle nous voulons trouver les fils qui ont bloqués l'expansion, demande des explications additionnelles. Voici l'algorithme permettant de trouver ces fils encombrants:

Pour les cases d'expansion, du meilleur indice de région idéale au pire.

Pour les six directions d'expansion permises.

Si la direction était bloquée.

- Calculer le nombre de modifications à effectuer pour libérer la case dans cette direction.

Pour les voisins, en ordre croissant du nombre de modification à effectuer

- Trouver les routes qui permettent de libérer la bonne direction et qui n'ont pas leur point de départ ou d'arrivée dans cette case.

- Garder ces routes

Si aucune route ne peut être enlevé.

- Recommencer complètement le routage de tous les chemins en commençant par celui bloqué.

L'algorithme n'enlève pas de fils qui ont un point de départ ou d'arrivée dans cette case car ces fils ont cette contrainte fixe. En effet, même si nous enlevons ces fils et les reroutons, ils se retrouveront toujours dans la même case puisque c'est un point de départ ou d'arrivée.

L'algorithme peut enlever un ou plusieurs fils déjà routés. Ceci afin d'assurer que l'expansion pourra se poursuivre dans la direction qui nous intéresse. En effet, si nous avions permis l'enlèvement d'un seul fil, il aurait pu se produire, lors du reroutage d'un fil, le bloquage de

celui-ci aux mêmes endroits étant donné qu'il y avait plusieurs chemins qui le bloquaient dans la même case d'expansion.

Lorsque nous vérifions les routes qui peuvent être enlevées, nous vérifions les directions bloquées qui nous intéressent. Nous allons aussi vérifier que ces fils à enlever n'ont jamais encore été enlevés en ayant la même configuration physique de routage. En d'autres mots, nous empêchons le reroutage d'un même fil s'il n'y a pas eu amélioration ou modification des fils déjà routés. Nous voulons empêcher les cycles d'enlèvement\reroutage du ou des mêmes fils sans amélioration notable des résultats du routage d'un circuit imprimé.

La dernière alternative du processus de mûrissement permet de recommencer le routage de tous les fils. Cette action est introduite pour des raisons de sécurité. En effet, nous voulons obliger le système à essayer des solutions complètement nouvelles plutôt que de stopper le routage par un échec.

5.4 FORME DES RÉSULTATS ET DE LA COMMANDE GÉNÉRÉE

Une fois le processus de mûrissement terminé, le contrôle du routage est repris par le routeur global. Celui-ci utilisera une liste de connexions à router plus grande qu'avant le blocage. Nous aurons dans cette liste:

1- le fil qui était bloqué
2- les fils que nous avons enlevés par ordre de grandeur
3- les fils qui n'ont pas encore été routés.

Nous avons aussi, la matrice de blocage qui a été modifiée. Lorsque les fils à enlever sont identifiés, nous utilisons les faits **chemin** qui leurs sont associés. Ces faits, nous indiquent l'emplacement et le type des graphèmes qui constituent les chemins à enlever. Ces informations sont utilisées afin de libérer l'espace qui était utilisé par les chemins dans la matrice de blocage.

Finalement, la matrice d'expansion sera détruite. Nous aurions pu poursuivre l'expansion aux endroits où elle était bloquée, ce qui aurait réduit le temps de calcul d'une nouvelle expansion complète. Cependant, en recommençant complètement l'expansion, nous nous assurons de vérifier

toutes les nouvelles régions d'expansion qui ont été libérées par l'enlèvement de chemins bloqueurs. Une poursuite de l'expansion, aurait entrainé uniquement la vérification des nouvelles régions d'expansion à proximité des régions de blocage. De plus, il est difficile de déterminer le niveau de poursuite de l'expansion puisqu'il se peut que les régions idéales d'expansion aient un étiquette d'indice plus petit que celle de la dernière expansion produite.

Dans l'exemple de la figure 5.1, la région idéale était étiquetée 4, il aurait fallu poursuivre l'expansion à cette étape après avoir enlevé les fils bloqueurs. Par conséquent, il faudrait donc détruire les cases d'expansion de 5 à 9 ce qui est plus long que de recommencer au complet.

Nous allons, dans les chapitres qui suivent, comparer les résultats de l'algorithme hiérarchique avec les résultats de routeurs commerciaux et ceux d'un expert humain. Nous allons identifier les manques de l'algorithme hiérarchique et proposer des moyens d'améliorer les résultats.

CHAPITRE 6

ÉVALUATION DES RÉSULTATS

Dans ce chapitre, nous examinons les résultats du routage de petits exemples produits par le routeur hiérarchique. De plus, nous vérifions la faisabilité d'une hiérarchisation de l'espace de recherche lors du routage. Puisque le routeur global à grosse grille donne des résultats qui sont grossiers, il est difficile de vérifier la validité de ces résultats. Nous présenterons donc dans les résultats du routage une fois que le routeur global à grosse maille et que le routeur utilisant l'algorithme de Lee aient terminé leur travail. Nous pourrons donc comparer le routage final du routeur hiérarchique avec ceux produit par un expert humain ou un routeur commercial, et ce en utilisant des exemples identiques.

Tous les exemples qui suivent ont été routés en utilisant une grille de travail de 25 mil ainsi que deux couches de routage permettant chacune le routage dans les directions X et Y. Les hachures horizontales représentent les chemins de la couche 1 tandis que les hachures verticales représentent ceux de la couche 2.

Les fichiers contenant les données d'entrées pour chacun des exemples de routage sont présentés en annexe (Annexe 1). On y trouve principalement, la définition des composantes, des connexions à router et enfin des paramètres de routage pour chacun des exemples présentés.

6.1 EXEMPLE 1

Le premier exemple est constitué de deux composantes de huit broches chacune. La figure 6.1 illustre les quatres connexions constituant le circuit à router. Les figures 6.2 et 6.3 représentent les résultats du routeur hiérarchique. Dans le premier cas, nous avons utilisé trois grosses mailles pour le routeur global à grosse maille et ce en x et en y. Le second routage, celui de la figure 6.3,

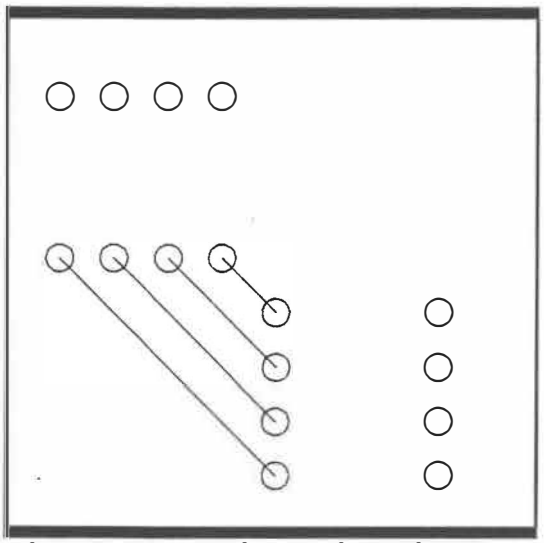


Figure 6.1 Visualisation de l'exemple 1.

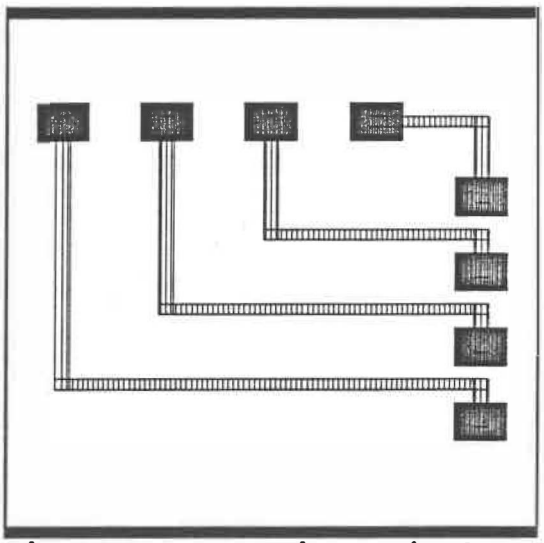


Figure 6.2 Premier résultat du routeur hiérarchique avec l'exemple 1.

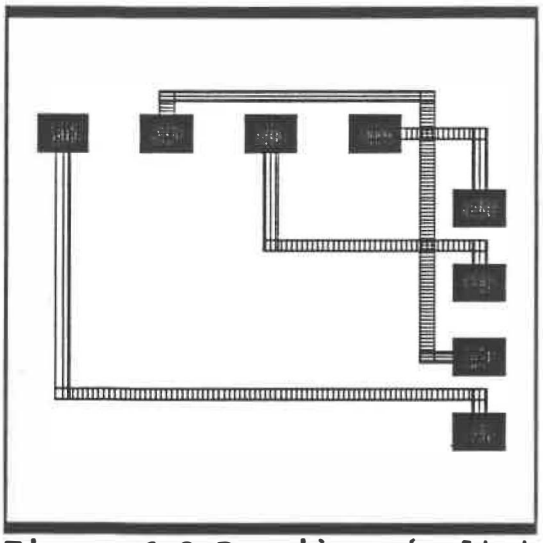


Figure 6.3 Deuxième résultat du routeur hiérarchique pour l'exemple 1.

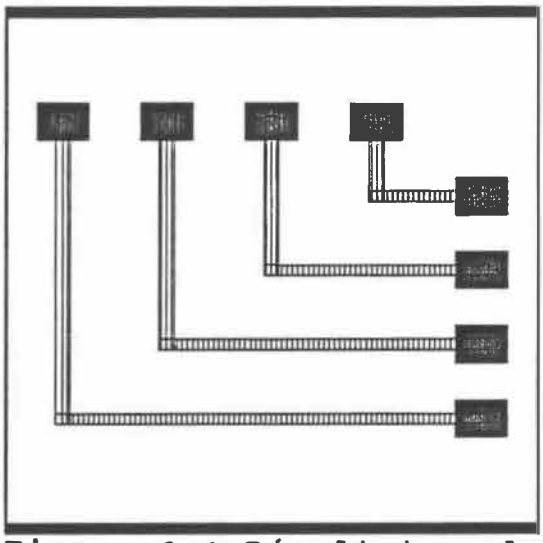


Figure 6.4 Résultat de l'expert humain pour l'exemple 1.

utilise cinq grosses mailles dans les deux directions. La figure 6.4 représente une solution idéale produite par un expert en routage de circuits imprimés.

Nous remarquons que la solution de la figure 6.2 respecte les contraintes imposées quoiqu'elle utilise deux couches. L'imposition de routage sur une seule couche aurait conduit, indépendamment du nombre de grosses mailles employées, à une solution similaire à celle de la figure 6.3.

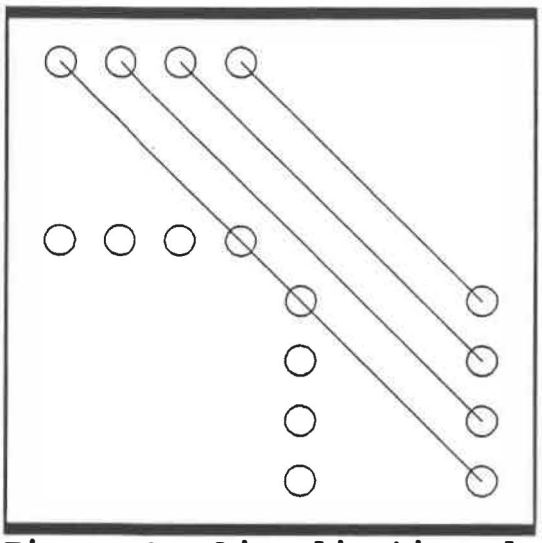


Figure 6.5 Visualisation de l'exemple 2.

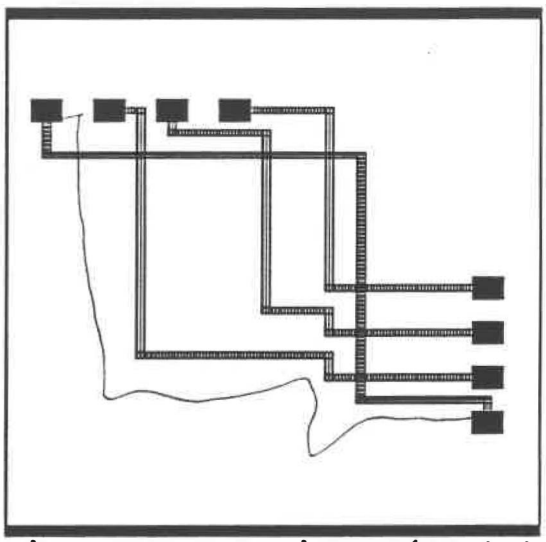


Figure 6.6 Premier réseutat du routeur hiérarchique pour l'exemple 2.

6.2 EXEMPLE 2

Le deuxième exemple utilise les mêmes composantes que l'exemple précédent. La figure 6.5 montre le nouvel ensemble de connexions qui doit être routé.

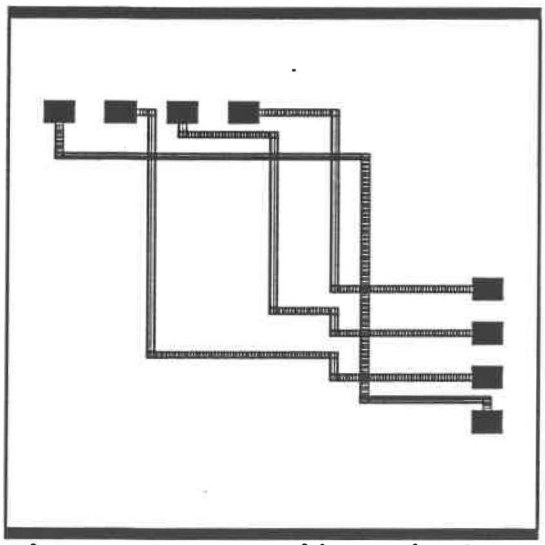


Figure 6.7 Deuxième résultat du routeur hiérarchique pour l'exemple 2.

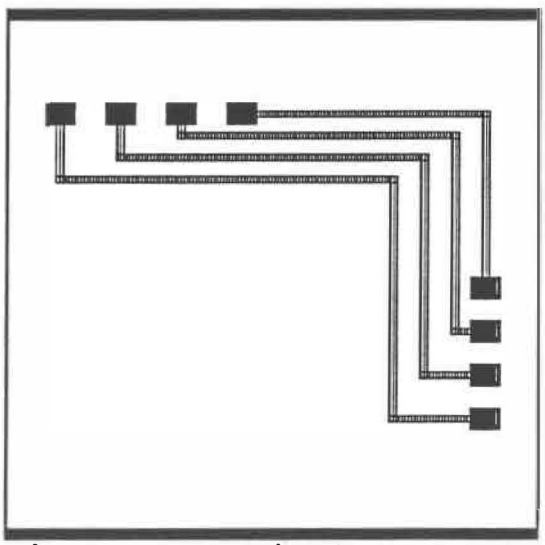


Figure 6.8 Résultat de l'expert humain pour l'exemple 2.

Les figures 6.6 et 6.7 sont des solutions différentes mais acceptables, obtenues en faisant varier le nombre de grosses mailles. La figure 6.8 représente la solution de l'expert.

6.3 EXEMPLE 3

Le troisième exemple utilise toujours les mêmes composantes que les exemples précédents. Cependant, nous allons compliquer le problème en ajoutant des connexions à router. Comme l'indique la figure 6.9, nous commandons le routage de huit connexions parallèles. La figure 6.10 présente le résultat du routeur hiérarchique alors que la figure 6.11 présente le résultat du routage d'un expert. Nous

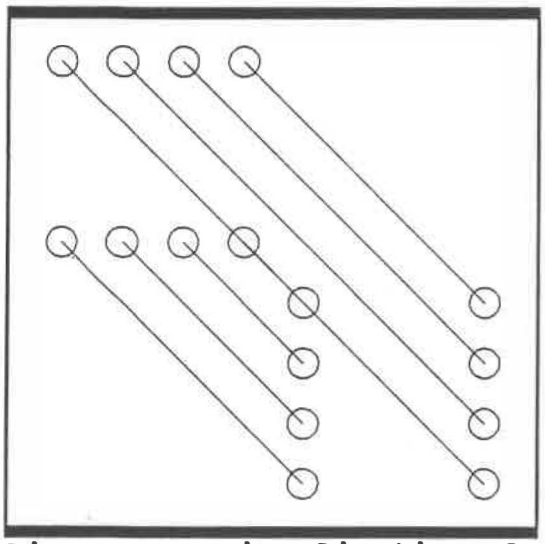


Figure 6.9 Visualisation de l'exemple 3.

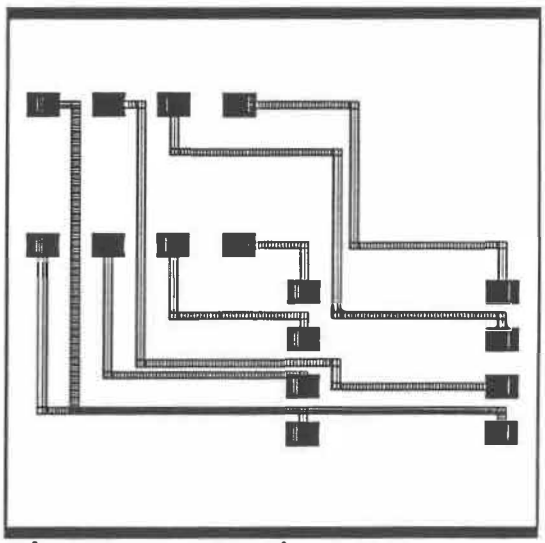


Figure 6.10 Résultat du routeur hiérarchique pour l'exemple 3.

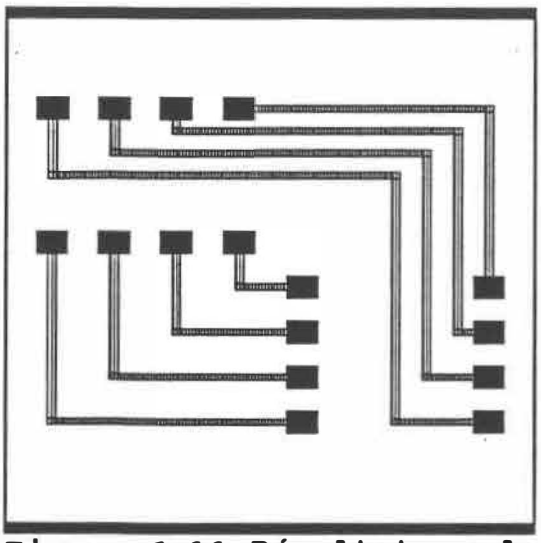


Figure 6.11 Résultat de l'expert humain pour l'exemple 3.

remarquons principalement que le résultat du routeur hiérarchique n'est pas symétrique. En effet, il manque une planification sous forme de bus pour les ensembles de fils.

6.4 EXEMPLE 4

Cet exemple ressemble beaucoup au précédent. Il utilise les mêmes composantes ainsi que les huit connexions.

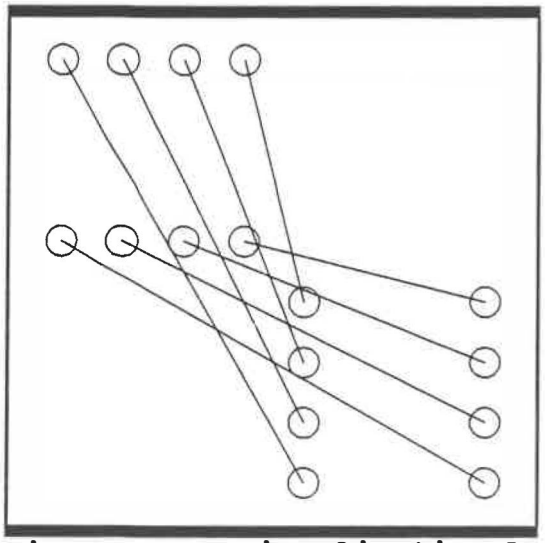


Figure 6.12 Visualisation de l'exemple 4.

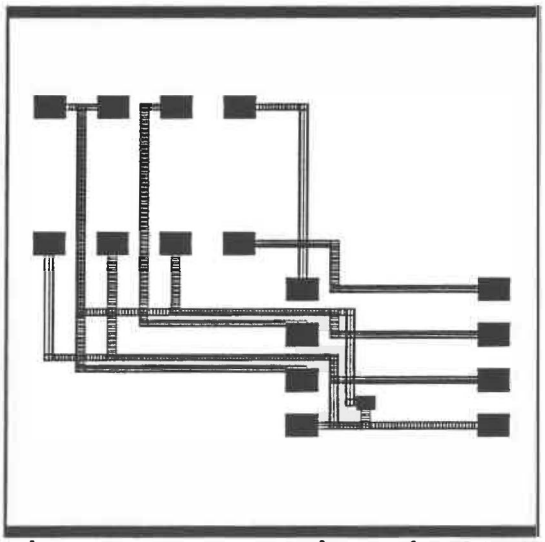


Figure 6.13 Premier résultat du routeur hiérarchique pour l'exemple 4.

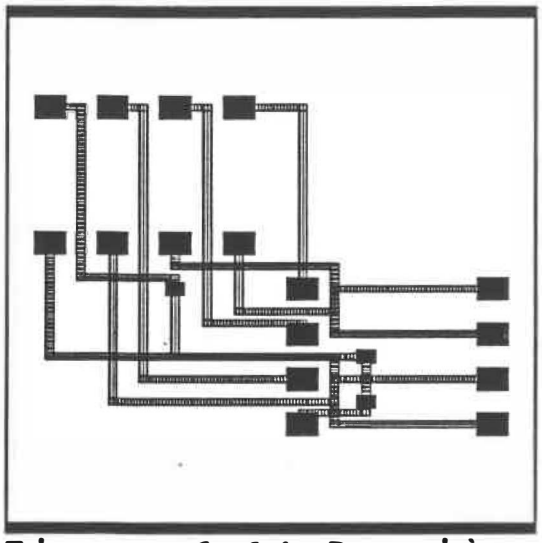


Figure 6.14 Deuxième résultat du routeur hiérarchique pour l'exemple 4.

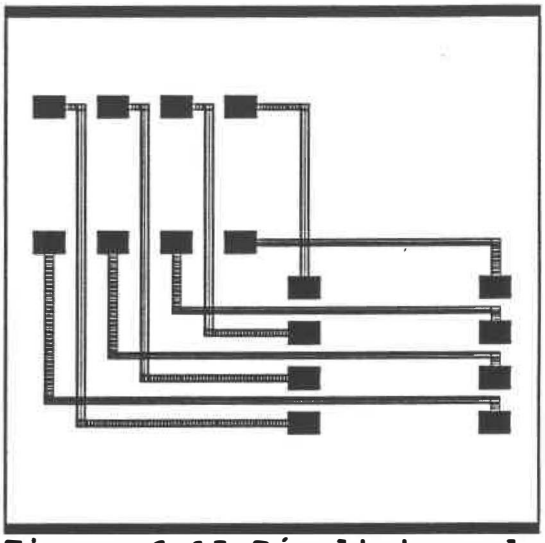


Figure 6.15 Résultat de l'expert humain pour l'exemple 4.

Toutefois, les connexions sont définies de manière à se croiser. La figure 6.12 présente l'ensemble des connexions à router. Les figure 6.13 et 6.14 présentent des solutions alternatives de routage qui ont été obtenues en utilisant un nombre de grosses mailles différent pour chaque cas. Il faut remarquer que la solution de la figure 6.13 est plus optimale que celle de la figure 6.14 puisqu'elle utilise deux traverses en moins. La solution de l'expert à la figure 6.15 est, par ailleurs, nettement plus avantageuse puisqu'elle n'utilise qu'une seule couche et aucune traverse.

6.5 EXEMPLE 5

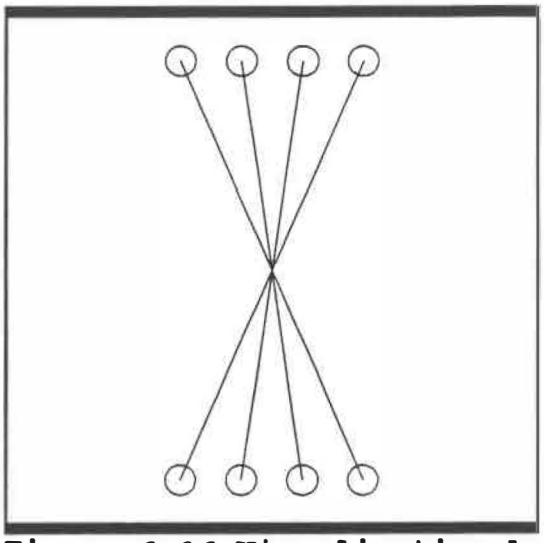


Figure 6.16 Visualisation de l'exemple 5.

Le cinquième exemple ne comprend que quatre connexions. Ces dernières se croisent au centre de la plaque comme le montre la figure 6.16. Cet exemple est très intéressant puisqu'il est impossible d'obtenir une solution en utilisant une seule couche. En effet, une solution respectant les contraintes de connectivité nécessitera un minimum de deux traverses. La figure 6.17 présente la solution du routeur hiérarchique. Cette solution utilise uniquement deux traverses comme celle de l'expert humain, présentée à la

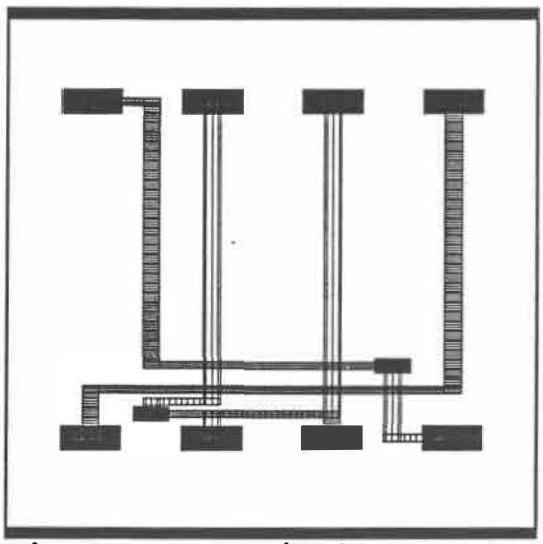


Figure 6.17 Résultat du routeur hiérarchique pour l'exemple 5.

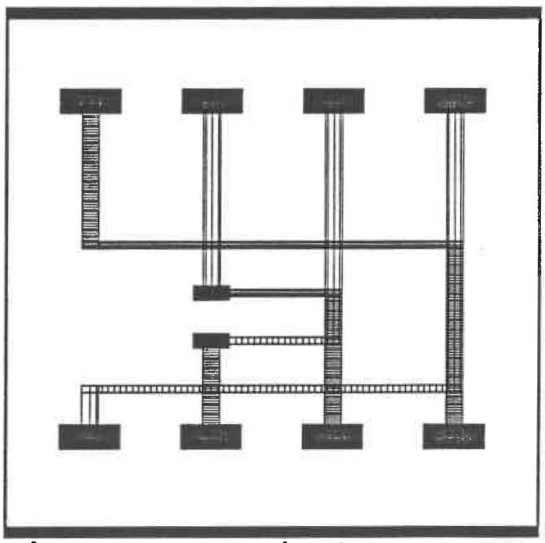


Figure 6.18 Résultat de l'expert humain pour l'exemple 5.

figure 6.18. Nous pouvons remarquer que les deux solutions respectent les contraintes de connectivité en n'utilisant que deux traverses. Néanmoins, la solution de l'expert est nettement plus esthétique. Malheureusement, le routeur hiérarchique ne connaît pas cette caractéristique d'un circuit imprimé.

6.6 EXEMPLE 6

L'exemple 6 démontre le type de problème pour lequel un routeur par ligne excelle. Le circuit à router, présenté à la figure 6.19, est un ensemble de quatre connexions donc les points d'arrivée et de départ sont colinéaires. De plus, les quatre connexions sont imbriquées les unes dans les autres. Les figures 6.20 et 6.21 représentent respectivement les solutions du routeur hiérarchique et celle de l'expert. Nous remarquons que les deux solutions sont pratiquement identiques à l'exception de certains points de départ dont le chemin associé se dirige dans une mauvaise direction sur une courte distance.

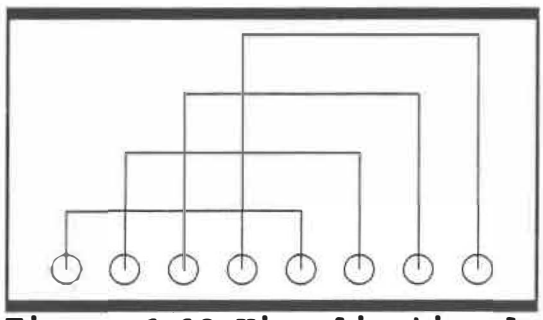


Figure 6.19 Visualisation de l'exemple 6.

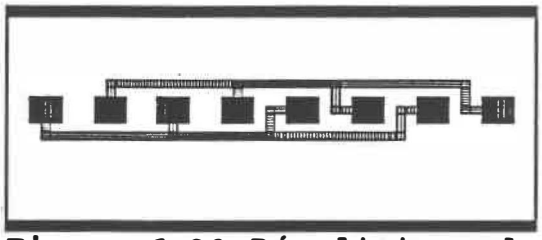


Figure 6.20 Résultat du routeur hiérarchique pour l'exemple 6.

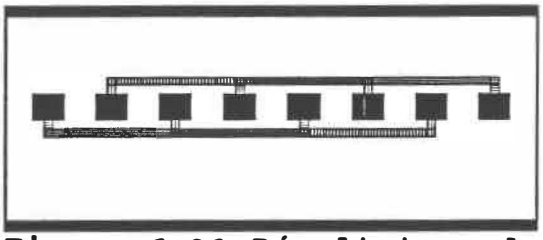


Figure 6.21 Résultat de l'expert humain pour l'exemple 6.

6.7 EXEMPLE 7

Le septième exemple est le dernier petit test qu'a subi le routeur hiérarchique. Il comprend un ensemble de huit connexions parallèles situées entre deux composantes de huit broches. Cet exemple, représenté à la figure 6.22, simule le routage de deux composantes de mémoire placées sur une plaque, l'une au dessus de l'autre et dont toutes les broches d'adresses et de données seraient connectées ensembles. Nous remarquons que la solution du routeur hiérarchique à la figure 6.23, et celle de l'expert à la figure 6.24 sont équivalentes. En effet, les deux solutions nécessitent l'utilisation de deux couches. Il est toutefois apparent que la solution de l'expert a l'avantage d'utiliser plus uniformément l'espace disponible sur le circuit.

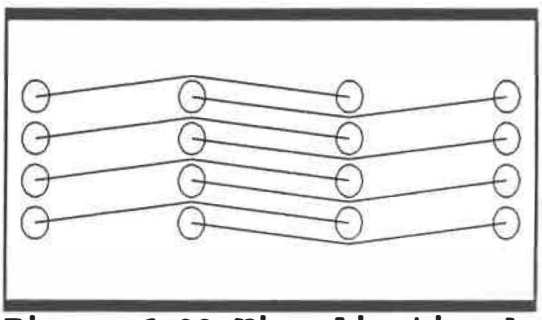


Figure 6.22 Visualisation de l'exemple 7.

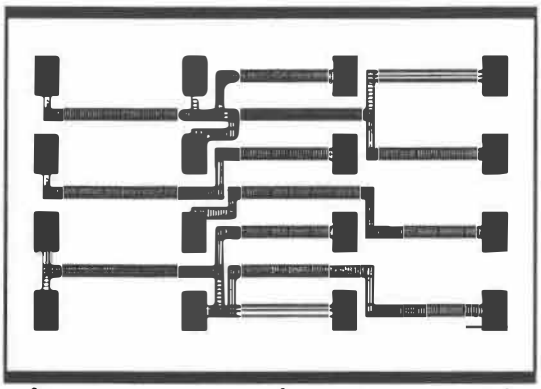


Figure 6.23 Résultat du routeur hiérarchique pour l'exemple 7.

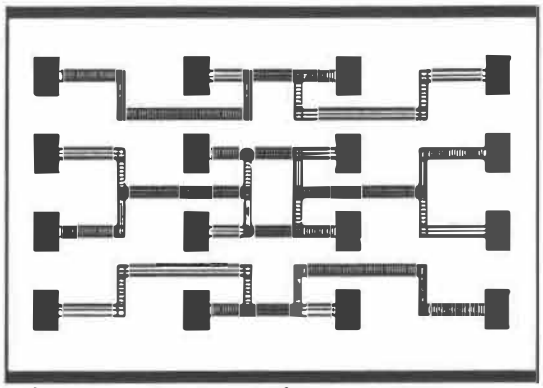


Figure 6.24 Résultat de l'expert humain pour l'exemple 7.

6.8 EXEMPLE 8

L'exemple huit est beaucoup plus complexe que les précédents. Comme le démontre la figure 6.25, il s'agit de deux composantes de seize broches et d'un ensemble de seize connexions. Les connexions ont été définies de manière à ce

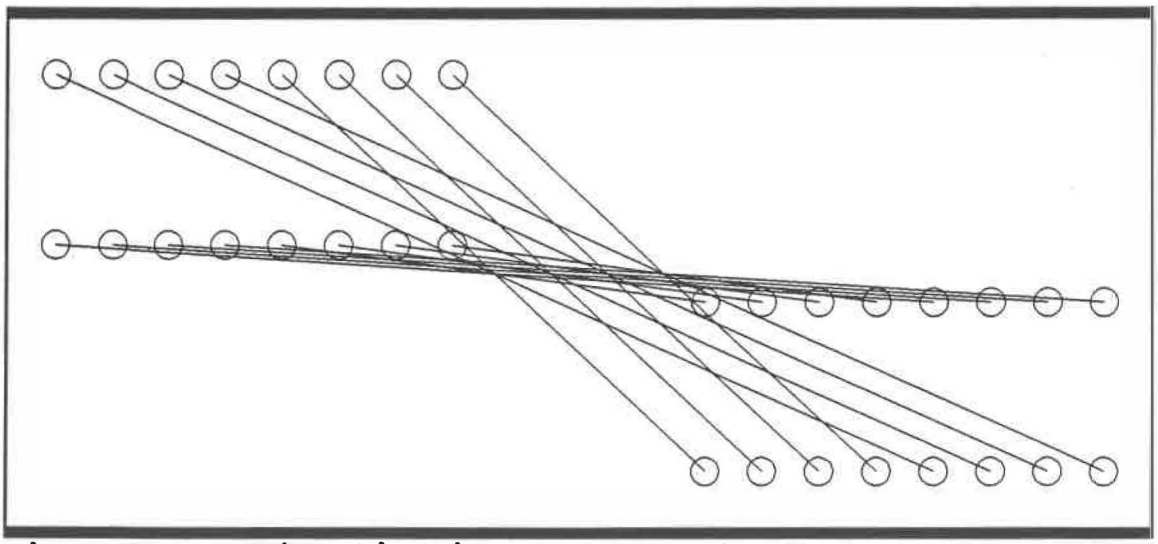


Figure 6.25 Visualisation de l'exemple 8.

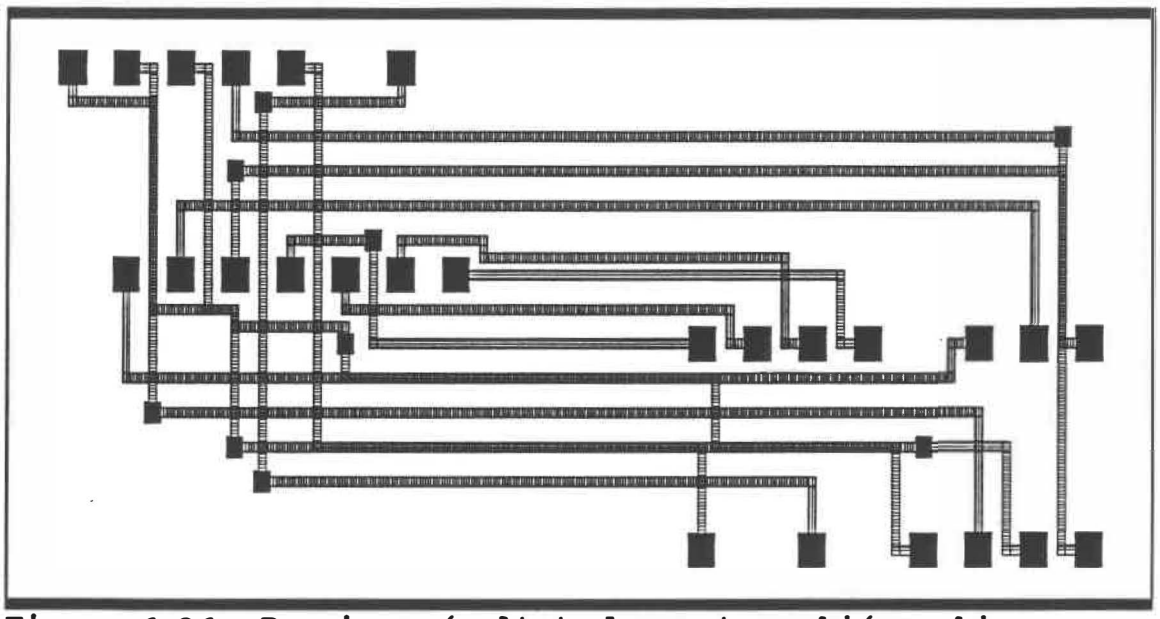


Figure 6.26 Premier résultat du routeur hiérarchique pour l'exemple 8.

qu'elles convergent toutes vers une région commune au centre du circuit imprimé. Cette convergence a pour effet de complexifier le routage. La figure 6.26 présente une première solution obtenue par le routeur hiérarchique. Ici, le résultat du routage n'est pas complet étant donné que trois connexions n'ont pu être routées et neuf traverses ont été utilisées afin de router les autres. La solution de la figure 6.27 n'est pas complète elle non plus. Le nombre de grosses mailles dans les deux directions a été modifié ce qui génère un circuit imprimé avec une connexion supplémentaire. Il ne manque cette fois que deux fils pour compléter le circuit; cependant, ce routage nécessite douze traverses.

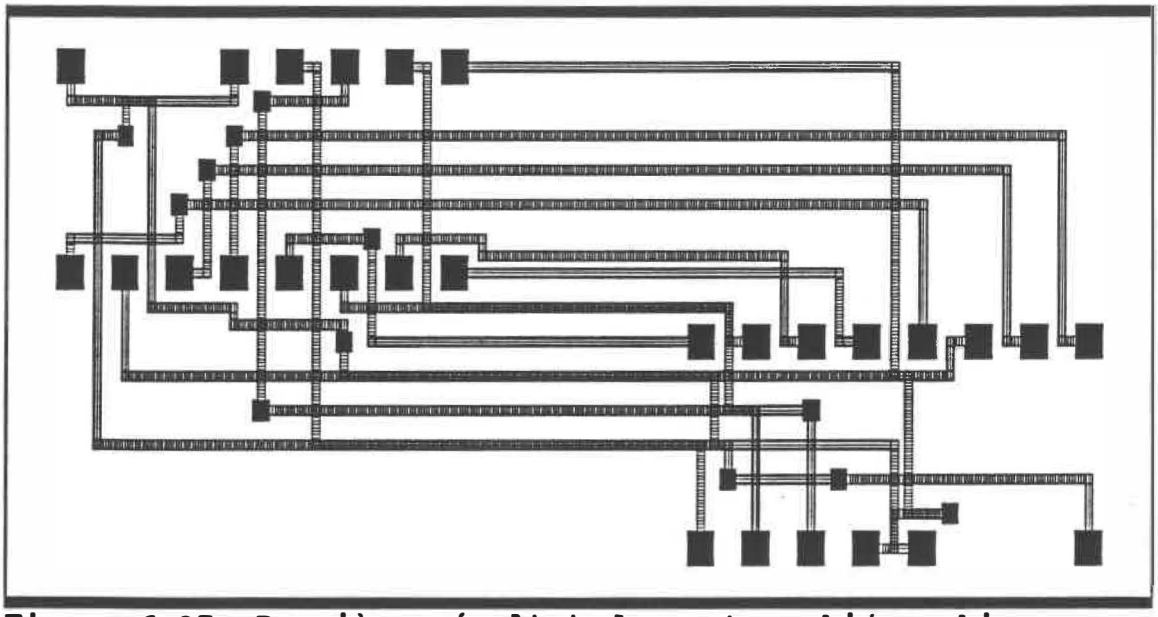


Figure 6.27 Deuxième résultat du routeur hiérarchique pour l'exemple 8.

Notez que la plupart des chemins horizontaux, dans les deux solutions du routeur hiérarchique sont routés avec un espacement de 50 mil entre eux. Même si la grille de travail est de 25 mil, le routeur est incapable de rapprocher les fils afin de libérer certains espaces pour le routage de d'autres fils. De même, à la figure 6.27, on remarque le placement non optimal de certaines traverses. Les quatre traverses, dans le coin supérieur gauche du circuit imprimé, forment une ligne diagonale. Cette arrangement bloque beaucoup de lignes en X

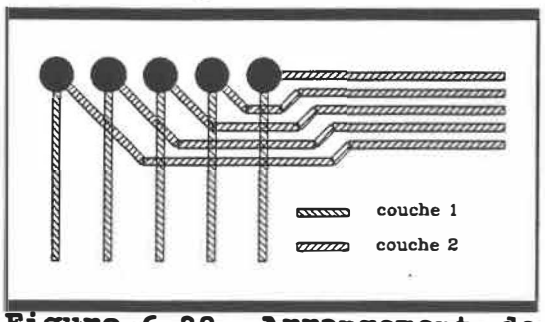


Figure 6.28 Arrangement de traverses idéale.

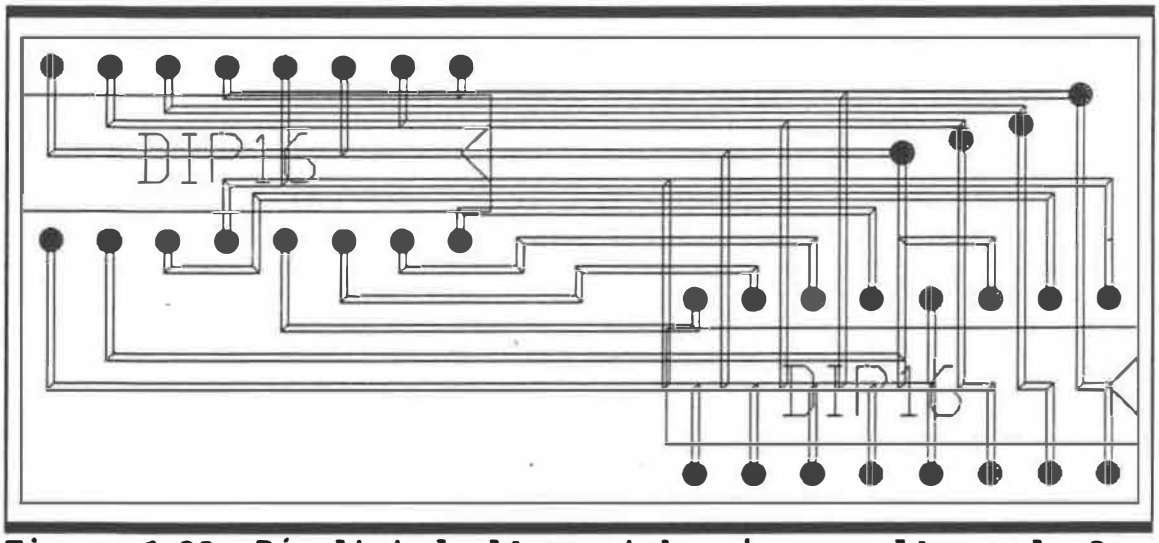


Figure 6.29 Résultat de l'expert humain pour l'exemple 8.

et en Y qui ne pourront plus être utilisé. Le placement de traverses sur une ligne diagonale est une des pire facons de les positionner. Un meilleur arrangement serait celui de la figure 6.28. La figure 6.29, présente la solution de l'expert. En Annexe 2 est présentée la solution de l'expert par deux figures, lesquelles représentent chacune des couches de routage séparément.

Il faut aussi remarquer que le routeur hiérarchique n'utilise pas la symétrie des connexions à router. En utilisant la symétrie des connexions, il est possible de former des bus qui, une fois routés, utiliseront moins d'espace de routage et produiront une solution plus esthétique.

6.9 EXEMPLE 9

Le neuvième et dernier exemple est un banc d'essai qui a été emprunté a la documentation d'un cours de formation sur l'utilisation du logiciel P-CAD. Cet exemple, comme le montre la figure 6.30 est très complexe. Il est composé de quatre composantes de seize broches chacune. L'espacement entre chaque composante est très restreint. Nous devons y router un ensemble de 28 connexions.

La figure 6.31 présente la solution d'un expert. Le routage a été effectué en utilisant une grille de travail de 50 mil et en permettant le routage à 45^0, ce qui réduit la grille réelle de travail. Le routage produit par l'expert est complet et nécessite deux couches et 14 traverses.

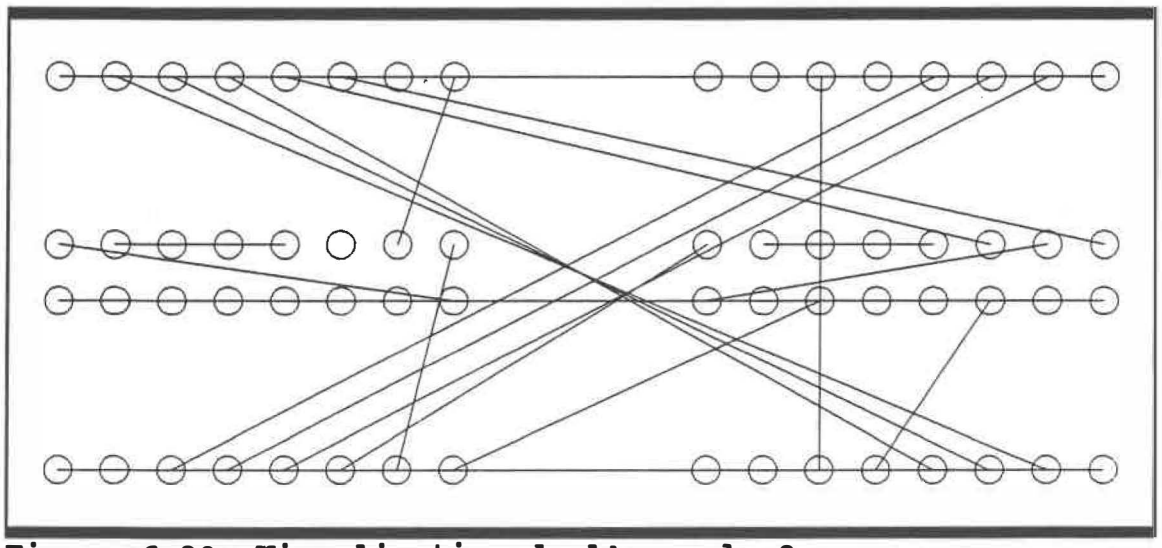


Figure 6.30 Visualisation de l'exemple 9.

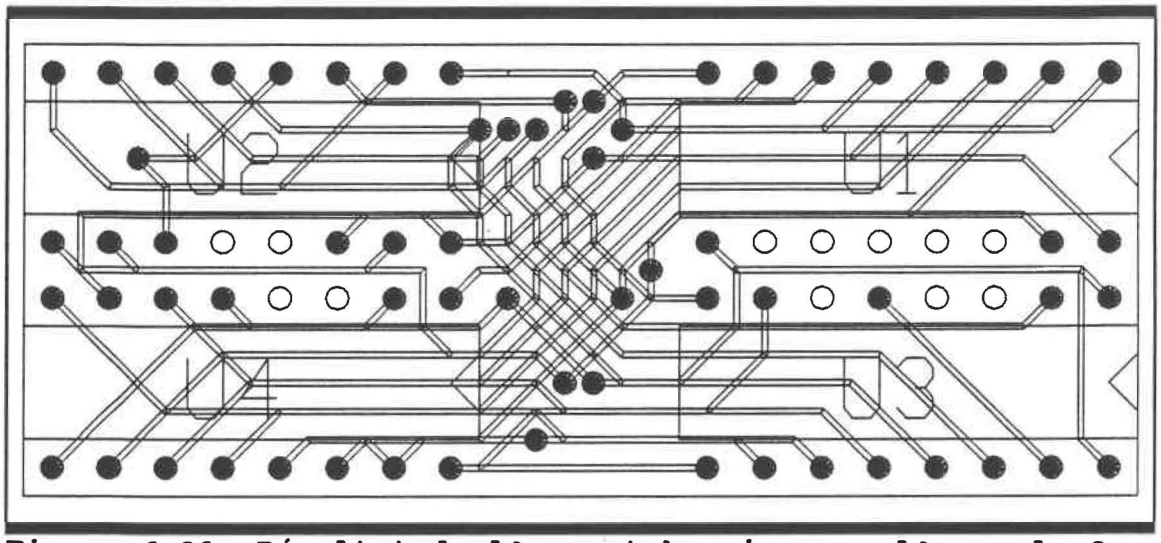


Figure 6.31 Résultat de l'expert humain pour l'exemple 9.

La solution de la figure 6.32 représente le résultat d'un routeur automatique commercial. Le circuit n'a pas été routé complètement, en utilisant une grille de travail de 25 mil. Le routage final utilise deux couches, 27 traverses et laisse sept fils non routés. Le routeur commercial offre des

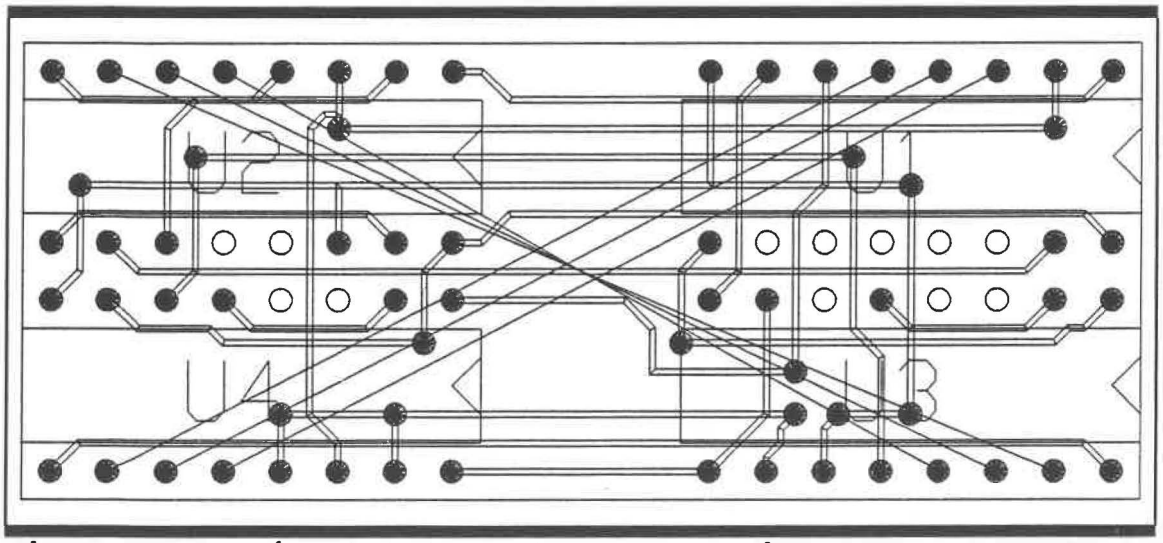


Figure 6.32 Résultat du routeur commercial pour l'exemple 9.

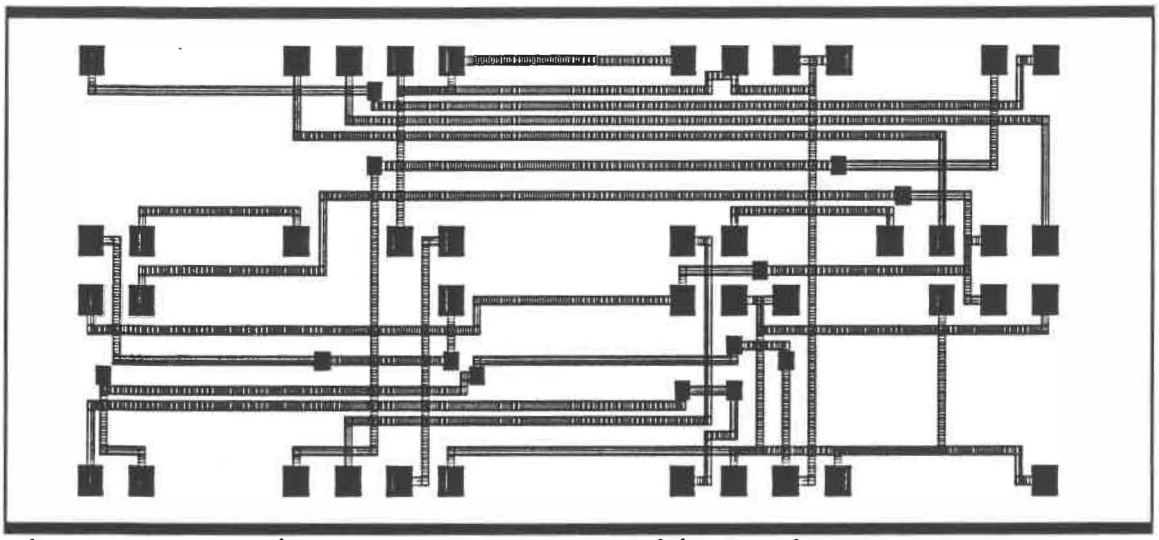


Figure 6.33 Résultat du routeur hiérarchique pour l'exemple 9.

performances nettement inférieures a celles d'un expert humain. Les deux figure, 6.31 et 6.32, sont décomposées afin de faire voir chacune des couches. Voir Annexe 2.

La figure 6.33 présente la solution obtenue a l'aide de notre routeur hiérarchique. Le routage a été effectué en utilisant une grille de travail de 25 mil, en utilisant des chemins en X ou en Y. Le routeur n'a réussi qu'à router 23 connexions; cinq connexions n'ont pu être placées. Ces dernières n'ont pu être positionné principalement pour les mêmes raisons que celles introduites lors des sections précédentes.

Nous devons cependant remarquer que le nombre de traverses employées reste intéressant. Le routeur n'a utilisé que 13 traverses pour les 23 premières connexions.

Une hiérarchisation dans l'espace de recherche de routage est donc réalisable. Toutefois, afin de produire des résultats plus esthétiques et ainsi plus comparables aux résultats d'un expert, le routeur global à grosses mailles nécessitera quelques modifications. Les performances d'un expert humain sont considérablement plus élevées et demeurent toujours un idéal à atteindre.

Dans le chapitre suivant, nous examinerons les principaux correctifs a appliquer au routeur hiérarchique afin qu'il produise des résultats comparable à ceux d'un expert humain.

CHAPITRE 7

CONCLUSION ET VOIES DE RECHERCHE FUTURES

Dans ce chapitre, nous présentons brièvement les problèmes qui ont été dégagés dans les chapitres précédents. De plus, nous identifierons les problèmes moins apparents mais dont la solution permettrait l'obtention d'un meilleur routage de circuit imprimé. Nous présenterons également le travail qu'il reste à faire afin de terminer complètement le routeur hiérarchique présenté au chapitre 2. Enfin, nous identifierons les voies de recherche possibles qui ont été mises en évidence par l'implantation de ce premier prototype.

7.1 PROBLÈMES RELATIFS AU ROUTEUR GLOBAL À GROSSES MAILLES.

Le problème majeur du routeur global est qu'il génère des chemins impossibles à router. Le routeur travaille maille par maille et s'assure pour chacune qu'il n'y ait pas d'incohérence du genre; une ligne en X et une ligne en Y sur la même couche. Toutefois, dans l'ensemble de toutes les mailles, il y a des impossibilités de routage. Par exemple, a la figure 7.1, même si les contraintes d'espacement physique et de compatibilité de graphèmes sont respectées

pour chacune des cases, il sera pourtant impossible de réussir le routage local puisque les chemins 1 et 2 devront nécessairement se croiser dans l'espace délimité par ces deux cases.

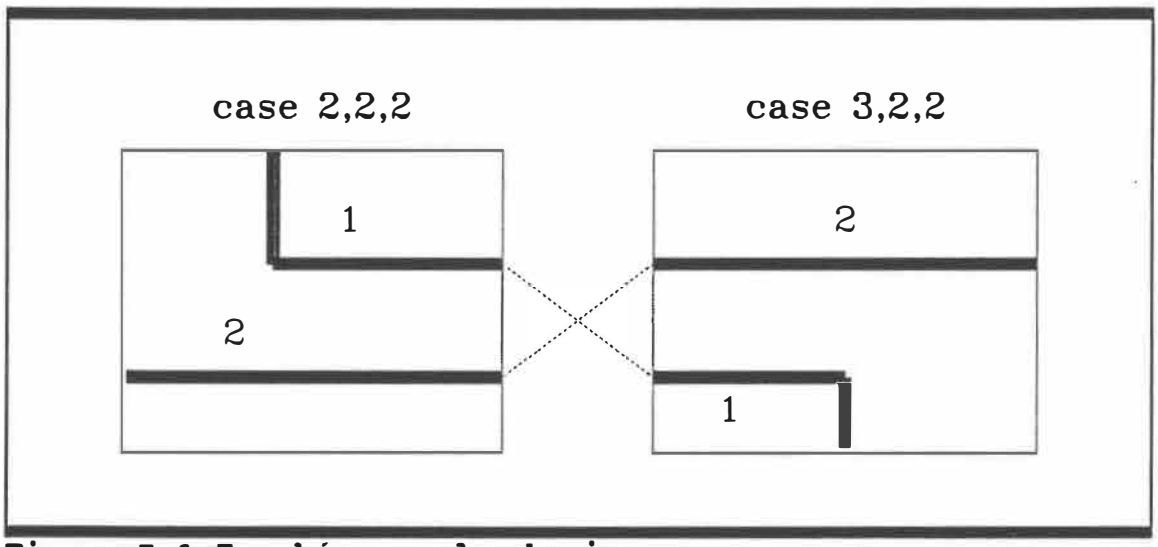


Figure 7.1 Incohérence de chemins.

Il serait donc important d'effectuer une vérification de la cohérence pour l'ensemble des cases. Il faudra vérifier, pour chaque établissement de graphème dans une case donnée, si les contraintes globales, imposées par les fils déjà routés, sont respectées. Pour l'exemple décrit à la figure 7.1, il faudra donc vérifier, lors de l'établissement du graphème **ligne_x** de la connexion 2 à la case 3.2.2, que les ancêtres du fil 2 sont positionnés plus haut dans la case, que ceux du fil 1. Cette contrainte étant, ici non respectée, il faudra trouver un autre chemin.

Le manque de planification du routeur global est une lacune ayant un inpact important sur les résultats de routage. En effet, si le routeur reconnaissait les connexions qui sont presque parallèles, la formation de bus de routage serait possible. Ces bus de routage permettront de router plusieurs fils simultanément. Nous obtiendrons ainsi des résultats plus esthétiques, plus simples et plus rapidement.

Le regroupement par bus et les stratégies de planification des endroits de routage sont des connaissances difficiles à identifier. Habituellement, ce sont des petits trucs du métier qui font la différence entre un expert humain et un programme informatique.

Les facteurs d'espacement utilisés pour chaque graphème donnent de bons résultats pour les exemples présentés. Il faudrait cependant, vérifier si ces facteurs demeurent tout aussi adéquats pour des contraintes de routage différentes. Il se peut bien qu'ils ne soient plus adéquats pour le routage, par exemple, d'un circuit imprimé à 32 couches dont la grille de travail est de 10 mil.

7.2 PROBLÈMES RELATIFS AU ROUTEUR LOCAL.

Concernant le routeur local, il a été précisé, au chapitre précédent, qu'il utilisait des emplacements non-optimaux pour les traverses. L'utilisation d'heuristiques ou encore d'expertise pour le placement des traverses, permettrait d'utiliser plus efficacement l'espace de routage. En effet, une heuristique qui place les traverses colinéairement, permettrait un gain important d'espace utilisable pour le routage de connexions additionnelles. Il est important de préciser, que le choix de l'emplacement de chaque traverse doit se faire avant le routage local.

Nous avons constaté, par le biais de certains exemples du chapitre précédent, une perte d'espace entre deux chemins parallèles parce qu'ils n'étaient pas rapprochés. Alors, nous croyons que l'utilisation d'un compacteur permettrait de rapprocher le plus possible les chemins parallèles, libérant ainsi un espace additionnel utilisable pour le routage d'autres connexions. Une autre approche serait de modifier ou d'utiliser un routeur local autre que celui de Lee avec objectif de performance, non plus le plus court chemin, mais le moins d'espace gaspillé. Un espace gaspillé signifiant ici un espace non utilisé qui devient inutilisable.

D'autre part, puisque le routage réalisé par le routeur global à grosses mailles est assez restrictif sur l'espace de recherche qu'utilise le routeur local, il serait plus adéquat d'utiliser,au niveau local, un routeur par expansion de ligne ou de région. Ces algorithmes sont moins exhaustifs dans leur recherche que ceux de la famille de Lee, mais ils sont cependant plus rapides. De plus, un routeur par expansion de ligne serait très bien adapté pour fonctionner conjointement avec un compacteur de lignes.

7.3 PROBLÈMES LIÉS À L'ENVIRONNEMENT.

Le langage Prolog est excellent pour effectuer du prototypage rapide et des traitements symboliques. Un routage composé de graphèmes est un exemple de traitement symbolique qui bénéficie de l'utilisation d'un langage comme Prolog.

Il y a cependant des désavantages à utiliser le langage Prolog sous l'environnement DOS. Nous savons que l'architecture d'un PC repose sur le principe d'une segmentation de 64K. Sous Arity Prolog, nous disposons d'une pile globale maximale de 64K. Puisque la force de Prolog se trouve dans l'utilisation de la récursivité dans les prédicats, nous sommes très hypothéqués par une pile de si

petite dimension maximale. Dès qu'un prédicat hautement récursif est utilisé, nous avons des chances d'obtenir une erreur de débordement de la pile.

Arity Prolog s'est défini une quantité de mémoires de 2K appelé **Page**. Chaque liste et définitions de prédicats doivent pouvoir se limiter à une page. Dans la plupart des applications, à l'exception de quelques unes, cette contrainte ne pose pas de problème. Pendant le processus de mûrissement, nous créons une liste de toutes les distances entre les cases d'expansion et la case d'arrivée. Cette liste est limitée à 120 cases afin de respecter la limite de 1 **page** mémoire pour représenter la liste.

7.4 VOIES DE RECHERCHE FUTURES.

La structure hiérarchique, décrite au chapitre 2 n'est pas complète. En effet, il reste à développer les niveaux 1,3 et 5. Comme nous l'avons déjà décrit au chapitre 2, le niveau 1 s'occupe de faire le placement des composantes sur la plaque. Le niveau 3 est un routeur globali-local. C'est lui qui effectuera le tampon entre l'espace de routage du routeur global et celui du routeur local. De plus, c'est encore lui qui s'occupera de placer les traverses aux endroits appropriés, réglant ainsi le problème énoncé pour le

routeur local. Enfin, le niveau 5 est un système expert pouvant résoudre des petits problèmes de routage très localisés.

En plus d'utiliser la théorie des graphes afin d'éliminer les incohérences dans le routage et d'utiliser des heuristiques ou expertises afin de planifier de manière adéquate le routage de tous les fils, nous proposons trois avenues supplémentaires d'amélioration.

Premièrement, nous pensons qu'une recherche par graphèmes serait plus intéressante qu'une recherche par case utilisée par les algorithmes de Lee. Les graphèmes resteraient toujours reliés à un ensemble de trois grosses mailles du routage global. Nous chercherions donc des graphèmes qui peuvent se juxtaposer et non, des cases voisines dans lesquelles nous aurions assez d'espace pour placer un graphème. Avec cette méthode de recherche, toute l'information nécessaire, pour vérifier de la non cohérence de certains chemins, devient disponible; cela n'est pas possible avec l'algorithme de Lee global. Cette recherche aurait également l'avantage de donner des résultats plus facilement interprétables du point de vue cohérence que ceux fournis par le routeur global à grosses mailles actuel.

Pour une seconde amélioration, il s'agit d'introduire une méthode pour résoudre les réseaux de connexions qui comprennent plus de deux extrémités. Il existe déjà beaucoup d'algorithmes pour résoudre cette tâche efficacement comme le "Minimum Spanning Tree" [2]. Ces algorithmes permettent de trouver des points intermédiaires d'un réseau de connexions à plusieurs extrémités. Ils visent à réduire la longueur totale des fils afin d'effectuer le routage de tous les réseaux comme le montre la figure 7.2.

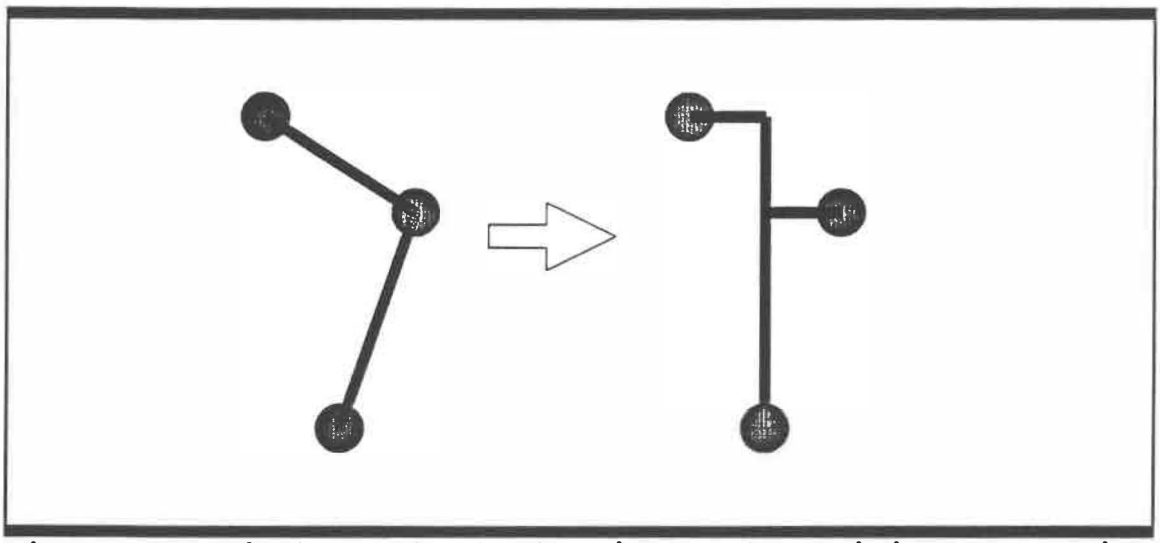


Figure 7.2 Résultat de l'algorithme du "Minimum Spanning Tree".

Enfin, la dernière voie de recherche que nous proposons est la possibilité d'utiliser une grille de routage variable. Par grille de routage variable, nous entendons une grille dont toutes les cases n'ont pas la même dimension.

Pour ce faire, une représentation de grilles comme le "Corner Stitching" [22] est envisageable. Cette méthodes permet un découpage plus fin dans les régions de congestion et les régions où se situent les broches de composantes. Le découpage permettant ainsi de tenir compte de la difficulté de routage dans ces régions de congestion tout évitant de perdre du temps de calcul pour des régions de routage plus simple.

En plus des travaux à continuer à chacun des niveaux de la hiérarchie, nous avons des efforts de réflexion à mettre sur les objectifs de performance découlant de la structure hiérarchique elle-même.

La complétion du module de conception physique, lequel contrôle le déroulement du routage à chacun des niveaux, nécessite le développement des métriques permettant d'évaluer les performances de chacun des niveaux. Ces métriques devront permettre, des mesures sur la distribution des fils sur tout le circuit imprimé, la localisation des régions de congestion et devront refléter l'atteinte de certains critères esthétiques.

Nous devrons étudier les possibilités et les difficultés de l'implantation de routeur pouvant chercher des chemins

dans toutes les directions et non pas seulement dans les directions X et Y. Nous avons remarqué que les experts humains utilisaient ce type de routage lequel n'est pourtant pas la première ressource des routeurs automatiques commerciaux.

Afin d'augmenter la flexibilité et la puissance du routeur hiérarchique nous devrons étudier les possibilités d'un système expert pouvant faire un choix d'algorithme de routage différent à chaque niveau dépendamment des contraintes physiques. Pour ce faire, plusieurs algorithmes devront être implantés et testés sur plusieurs exemples afin de faire ressortir les caractéristiques et avantages de chacun. De là, nous devrons développer un système expert qui serait capable d'identifier les caractéristiques d'un problème de routage et d'ensuite choisir le ou les algorithmes qui seront les plus performants pour le problème donné. Pour ce faire, nous devrons également développer une structure hiérarchique de recherche commune à tous les algorithmes.

Enfin, il serait intéressant d'étudier l'application possible de cette hiérarchie de routage à des problèmes de placement et de routage de design de composantes à très grande échelle d'intégration. Il serait même intéressant

d'étudier la possibilité d'une structure hiérarchique ayant assez d'expertise et de flexibilité aux contraintes de placement/routage afin de résoudre les problèmes de design, de circuit intégré et de circuit imprimé.

Ce projet nous permet d'affirmer que l'implantation d'une structure hiérarchique et itérative de routage est possible. En effet, nous avons démontré que des algorithmes aussi vieux que ceux de la famille de Lee peuvent être modifiés afin d'obtenir les capacités de routage sur des grilles variables permettant ainsi, la hiérarchisation des espaces de recherche pendant le processus de routage.

Nous pouvons aussi affirmer, que la commercialisation d'un logiciel de placement/routage exige une somme énorme de travail. Ces logiciels, en plus d'exiger le routeur lui-même, demandent la constitution d'une base de données de composantes électroniques et des interfaces d'imprimantes, de traceurs de courbes et d'écrans. De plus, ces logiciels fournissent des simulateurs de circuit, des outils de vérification des règles de design et bien plus encore. La réalisation d'un routeur commercial requière plusieurs années homme.

Nous avons pu également identifier les problèmes reliés à l'implantation d'une structure hiérarchique utilisant plusieurs algorithmes de routage. Ensuite, nous avons, pris conscience de l'existence de caractéristiques qui différencient les résultats d'un routeur automatique de ceux d'un expert humain.

L'expert humain dépasse largement un routeur automatique surtout par sa capacité d'adaptation aux changements des contraintes et grâce à son expérience de la planification à effectuer avant d'entreprendre un routage. Il faut rappeler que l'humain est une merveilleuse machine que nous devrons toujours tendre à imiter et utopiquement dépasser.

L'utilisation d'un langage de 5^{e} génération, nous a montré qu'un bon outil peut nous aider à obtenir un prototype très rapidement. Le traitement symbolique, la récursivité et le moteur d'inférence du langage Prolog en font un langage bien adapté aux techniques de prototypage. Par ailleurs, la puissance d'un langage comme Prolog diminue grandement lorsque le langage est implanté sur une machine de faible puissance. Même la meilleure implantation du langage Prolog sur ordinateur personnel de type IBM PC et compatibles, Arity Prolog possède des performances amoindries dû à une architecture matérielle non adéquate.

BIBLIOGRAPHIE

1. LEE, C.Y., "An Algorithm for Path Connections and Its Applications", IRE Transactions on electronic computers, Sept. 1961, pp. 346-365.

2. BOBBA, V.S., SMITH, J.W., "A Parameter-Driven Router", 23rd Design Automotion Conference, 1986, pp. 810-818.

3. DION J., "Fast Printed Circuit Board Routing", 24th ACM/IEEE Design Automotion Conference, 1987, pp. 727-734.

4. GAREY, M.R., JOHNSON, D.S., "Computers and Intractability: A Guide to the Theory of NP_Completeness", ed. W. H. Freeman and Company, San Francisco, 1979, 338 p.

5. CLOW, G.W., "A Global Routing Algorithm for General Cells", 21st Design Automation Conference, 1984, pp.45-51.

6. SHELDON, B., AKERS, JR., "A Modification of Lee's Path Connector Algorithms", IEEE transactions on Electronic Computers, Feb., 1967, pp. 97-98.

7. RUBIN, F., "The Lee Path Connection Algorithm", IEEE Transactions on computers, Vol. C-23, no.9, Sept. 1974, pp. 907-914.

8. HOEL, J.H., "Some Variations of Lee's Algorithm", IEEE Transactions on Computers, Vol. C-25, N° 1, Jan. 1976, pp.19-24.

9. FINCH, A.C., MACKENZIE, K.J., BALSDON, G.J., SYMOND, G., "A Method for Grid Less Routing of Printed Circuit Boards", 22nd Design Automation Conference, 1985, pp.509-515.

10. TADA, T., HANAFUSA, A., "Router System for Printed Wiring Boards of Very High-Speed", very large-scale computers, 23rd Design Automation Conference, 1986, pp.791-797.

11. DUPENLOUP, G., "A Wire Routing Scheme for Double-Layer Cell Arrays", 21st Design Automation Conference, 1984, pp.32-37.

12. YOSHIMURA, T., "An Efficient Channel Router", 21st Design Automation Conference, 1984, pp.38-44.

13. MAREK-SODDOWSKA, M., "Two-Dimensional Router for Double Layer Layout", 22nd Design Automotion Conference, 1985, pp.117-123.

14. HAN, S., SAHNI, S., "Layering Algorithms for Single Row Routing", 22nd Design Automotion Conference, 1985, pp.516-522.

15. JOSEPH, R.L., "An Expert Systems Approach to Completing Partially Routed Printed Circuit Boards", 22nd Design Automation Conference, 1985, pp.523-528.

16. FUJITA, T., GOTO, S., "A Rule-Based Routing System", IEEE 1983, pp.451-454.

17. ROSENBERG, E., "A New Iterative Supply/Demand Router with Rip-Up Capability for Printed Circuit Boards", 24th ACM/IEEE Design Automation Conference, 1987, pp.721-726.

18. KAWAMURA, K., UMEDA, M., SHIRAISHI, H., "Hierarchical Dynamic Router", 23rd Design Automation Conference, 1986, pp.803-809.

19. KESSENICH, J., JACKOWAY, G., "Global Forced Hierarchical Router", 23rd Design Automation Conference, 1986, pp.798-802.

20. NAVEDA, J.F., CHANG, K.C., DU, H.C., "A New Approach to Multi-Layer PCB Routing with Short Vias", 23rd Design Automation Conference, 1986, pp.696-701.

21. WON, K., HONG-TOI, C., JAY, B., "Operations and Implementation of Complex Objects", 24th Design Automotion Conference, 1981, pp.626-633.

22. OUSTERHOUT, J.K., "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools", IEEE transactions on computer-aided design of integrated circuits and systems, Vol. CAD-3, No. 1, January 1984, pp.87-100.

23. RITCHIE, D.M., BW kerninghon, "Le langage C", Ed. masson, Paris, 1983, 218p.

24. "Using the Arity/Prolog Compiler and Interpreter", Reference manual bu Arity Corporation, 1987, 266p.

25. "The Arity/prolog Language Reference Manual", Reference manual by Arith Corporation, 1988, 320p.

26. "The Arity Advanced Toolkit", Reference manual by Arity Corporation, 1988, 60p.

27. "Microsoft C Run-Time Library Reference", Reference manual for Microsoft C, Ver. 5.10, 1987, 687p.

28. ARTWICK, Bruce A., "Applied Concepts in Microcomputer Graphics", Ed. Prentice-Hall, New Jersey, 1984, 374p.

29. CAMARERO, R., "Introduction à la conception assistée par ordinateur", Notes de cours, École Polytechnique de Montréal, Sept. 1984, 185p.

30. LANGHEIT, C., BERNARD, J.C., "Bibliothèque de prédicats graphiques pour l'environnement de travail Arity Prolog", Éditions de l'École Polytechnique de Montréal, 1989, 85p.

31. LANGHEIT, C., BERNARD, J.C., "IARS: An Intelligently Assisted Routing System", The Second International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Vol.2, 1989, pp.544-552.

ANNEXE 1

A1.1 FAITS DE L'EXEMPLE 1

Voici la liste des faits qui ont permis de définir l'exemple 1 de routage. En premier, nous avons la liste des faits **composante**, permettant de définir chacune des broches de chaque composante.

```
composante(1,nom(200,550,a),
           dessin(0,650,300,50),
           configuration([(  0,400),
                          (100,400),
                          (200,400),
                          (300,400),
                          (300,700),
                          (200,700),
                          (100,700),
                          (  0,700)])).

composante(2,nom(550,200,b),
           dessin(450,0,650,300),
           configuration([(400,300),
                          (400,200),
                          (400,100),
                          (400,  0),
                          (700,  0),
                          (700,100),
                          (700,200),
                          (700,300)])).
```

Il nous faut aussi définir les connexions qui seront routées sur le circuit imprimé. Les faits **reseaux** permettent de définir les points de départ et d'arrivée de chaque connexion.

```
reseaux( 1,0,configuration([(a,1),(b, 4)])).
reseaux( 2,0,configuration([(a,2),(b, 3)])).
reseaux( 3,0,configuration([(a,3),(b, 2)])).
reseaux( 4,0,configuration([(a,4),(b, 1)])).
```

Finalement, nous avons les faits définissant les paramètres de routage.

pour figure 7.2

```
nb_carreau_en_x(7).
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

pour figure 7.3

```
nb_carreau_en_x(11).
nb_carreau_en_y(11).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

A1.2 FAITS DE L'EXEMPLE 2

Nous présentons maintenant, les faits définissant l'exemple 2 de routage. Voici la liste des faits **composante**, permettant de définir chacune des broches des composante.

```
composante(1,nom(200,550,a),
             dessin(0,650,300,50),
             configuration([(  0,400),
                            (100,400),
                            (200,400),
                            (300,400),
                            (300,700),
                            (200,700),
                            (100,700),
                            (  0,700)])).

composante(2,nom(550,200,b),
             dessin(450,0,650,300),
             configuration([(400,300),
                            (400,200),
                            (400,100),
                            (400,  0),
                            (700,  0),
                            (700,100),
                            (700,200),
                            (700,300)])).
```

Nous définissons aussi les connexions qui seront routées sur le circuit imprimé. La liste des faits **reseaux** permet de définir les points de départ et d'arrivée de chaque connexion.

```
reseaux( 1,0,configuration([(a,5),(b, 8)])).
reseaux( 2,0,configuration([(a,6),(b, 7)])).
reseaux( 3,0,configuration([(a,7),(b, 6)])).
reseaux( 4,0,configuration([(a,8),(b, 5)])).
```

Finalement, nous avons les faits qui paramétrisent le routage.

pour figure 7.6

```
nb_carreau_en_x(7).
nb_carreau_en_y(11).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

pour figure 7.7

```
nb_carreau_en_x(11).
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

A1.3 FAITS DE L'EXEMPLE 3

Voici les faits qui ont permis de définir l'exemple 3 de routage. En premier, nous avons la liste des faits **composante**, permettant de définir chacune des broches de composante. Ensuite, il nous faut aussi définir les connexions qui seront routées sur le circuit imprimé. Les faits **reseaux** permettent de définir les points de départ et d'arrivée de chaque connexion. Finalement, nous avons les faits définissant les paramètres de routage.

```
composante(1,nom(200,550,a),
             dessin(0,650,300,50),
             configuration([(  0,400),
                            (100,400),
                            (200,400),
                            (300,400),
                            (300,700),
                            (200,700),
                            (100,700),
                            (  0,700)])).
```

```
composante(2,nom(550,200,b),
             dessin(450,0,650,300),
             configuration([(400,300),
                            (400,200),
                            (400,100),
                            (400,  0),
                            (700,  0),
                            (700,100),
                            (700,200),
                            (700,300)])).

reseaux( 1,0,configuration([(a,1),(b, 4)])).
reseaux( 2,0,configuration([(a,2),(b, 3)])).
reseaux( 3,0,configuration([(a,3),(b, 2)])).
reseaux( 4,0,configuration([(a,4),(b, 1)])).
reseaux( 5,0,configuration([(a,5),(b, 8)])).
reseaux( 6,0,configuration([(a,6),(b, 7)])).
reseaux( 7,0,configuration([(a,7),(b, 6)])).
reseaux( 8,0,configuration([(a,8),(b, 5)])).

nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

A1.4 FAITS DE L'EXEMPLE 4

Les faits définissant l'exemple 4 sont présentés ci-dessus. Nous avons la liste des faits **composante**, permettant de définir chacune des broches de composante, les faits **reseaux** permettant de définir les points de départ et d'arrivée de chaque connexion et enfin, les faits définissant les paramètres de routage.

```
composante(1,nom(200,550,a),
             dessin(0,650,300,50),
             configuration([(  0,400),
                            (100,400),
                            (200,400),
                            (300,400),
                            (300,700),
                            (200,700),
                            (100,700),
                            (  0,700)])).

composante(2,nom(550,200,b),
             dessin(450,0,650,300),
             configuration([(400,300),
                            (400,200),
                            (400,100),
                            (400,  0),
                            (700,  0),
                            (700,100),
                            (700,200),
                            (700,300)])).

reseaux( 1,0,configuration([(a,1),(b, 5)])).
reseaux( 2,0,configuration([(a,2),(b, 6)])).
reseaux( 3,0,configuration([(a,3),(b, 7)])).
reseaux( 4,0,configuration([(a,4),(b, 8)])).
reseaux( 5,0,configuration([(a,5),(b, 1)])).
reseaux( 6,0,configuration([(a,6),(b, 2)])).
reseaux( 7,0,configuration([(a,7),(b, 3)])).
reseaux( 8,0,configuration([(a,8),(b, 4)])).
```

pour figure 7.13

```
nb_carreau_en_x(7).
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

pour figure 7.14

```
nb_carreau_en_x(5).
nb_carreau_en_y(5).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

A1.5 FAITS DE L'EXEMPLE 5

Voici la liste des faits qui ont permis de définir l'exemple 5 de routage. En premier, nous avons la liste des faits **composante**, permettant de définir chacune des broches de chaque composante.

```
composante(5,nom(150,100,e),
            dessin(-50,-50,350,50),
            configuration([(  0,  0),
                           (100,  0),
                           (200,  0),
                           (300,  0)])).

composante(6,nom(150,600,f),
            dessin(-50,650,350,750),
            configuration([(  0,700),
                           (100,700),
                           (200,700),
                           (300,700)])).
```

Il nous faut aussi définir les connexions qui seront routées sur le circuit imprimé. Les faits **reseaux** permettent de définir les points de départ et d'arrivée de chaque connexion.

```
reseaux( 1,0,configuration([(e,1),(f, 4)])).
reseaux( 2,0,configuration([(e,2),(f, 3)])).
reseaux( 3,0,configuration([(e,3),(f, 2)])).
reseaux( 4,0,configuration([(e,4),(f, 1)])).
```

Finalement, nous avons les faits définissant les paramètres de routage.

```
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(300).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

A1.6 FAITS DE L'EXEMPLE 6

L'exemple 6 est défini par la liste de faits suivants. Premièrement, le **composante**, permettant de définir chacune des broches de la composante.

```
composante(7,nom(350,200,g),
           dessin(-50,250,750,350),
           configuration([(  0,300),
                          (100,300),
                          (200,300),
                          (300,300),
                          (400,300),
                          (500,300),
                          (600,300),
                          (700,300)])).
```

Nous définissons aussi les 4 connexions qui seront routées. Les faits **reseaux** permettent de définir les extrémités de chaque connexion.

```
reseaux( 1,0,configuration([(g,1),(g, 5)])).
reseaux( 2,0,configuration([(g,2),(g, 6)])).
reseaux( 3,0,configuration([(g,3),(g, 7)])).
reseaux( 4,0,configuration([(g,4),(g, 8)])).
```

Enfin, Il y a les faits définissant les paramètres de routage.

```
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(700).
coor_max_y(600).
coor_min_x(0).
coor_min_y(0).
```

A1.7 FAITS DE L'EXEMPLE 7

Voici les faits permettant de définir l'exemple 7. En premier, nous avons les faits **composante**, qui permettent de définir chacune des composantes.

```
composante(3,nom(150,150,c),
             dessin(50,300,250,0),
             configuration([(  0,300),
                            (  0,200),
                            (  0,100),
                            (  0,  0),
                            (300,  0),
                            (300,100),
                            (300,200),
                            (300,300)])).

composante(4,nom(750,150,d),
             dessin(650,300,850,0),
             configuration([(600,300),
                            (600,200),
                            (600,100),
                            (600,  0),
                            (900,  0),
                            (900,100),
                            (900,200),
                            (900,300)])).
```

De plus, il faut définir les connexions qui seront routées. Les faits **reseaux** définissent les points de départ et d'arrivée de chaque connexion.

```
reseaux( 1,0,configuration([(c,1),(d, 1)])).
reseaux( 2,0,configuration([(c,2),(d, 2)])).
reseaux( 3,0,configuration([(c,3),(d, 3)])).
reseaux( 4,0,configuration([(c,4),(d, 4)])).
reseaux( 5,0,configuration([(c,5),(d, 5)])).
reseaux( 6,0,configuration([(c,6),(d, 6)])).
reseaux( 7,0,configuration([(c,7),(d, 7)])).
reseaux( 8,0,configuration([(c,8),(d, 8)])).
```

Les faits suivants définissent les paramètres de routage.

```
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(900).
coor_max_y(300).
coor_min_x(0).
coor_min_y(0).
```

A1.8 FAITS DE L'EXEMPLE 8

Voici la liste des faits qui ont permis de définir l'exemple 8 de routage. Premièrement, les faits **composante**, définissent chaques broches de composante.

```
composante(8,nom(1400,150,h),
             dessin(1150,250,1850,50),
             configuration([(1150, 0),
                            (1250, 0),
                            (1350, 0),
                            (1450, 0),
                            (1550, 0),
                            (1650, 0),
                            (1750, 0),
                            (1850, 0),
                            (1850,300),
                            (1750,300),
                            (1650,300),
                            (1550,300),
                            (1450,300),
                            (1350,300),
                            (1250,300),
                            (1150,300)])).
composante(9,nom(350,550,i),
             dessin(0,650,700,450),
             configuration([(  0,400),
                            (100,400),
                            (200,400),
                            (300,400),
                            (400,400),
                            (500,400),
                            (600,400),
                            (700,400),
                            (700,700),
                            (600,700),
                            (500,700),
                            (400,700),
                            (300,700),
                            (200,700),
                            (100,700),
                            (  0,700)])).
```

Il nous faut aussi définir les connexions qui seront routées sur le circuit imprimé. Les faits **reseaux** permettent de définir les points de départ et d'arrivée de chaque connexion.

```
reseaux( 1,0,configuration([(i, 1),(h,12)])).
reseaux( 2,0,configuration([(i, 2),(h,11)])).
reseaux( 3,0,configuration([(i, 3),(h,10)])).
reseaux( 4,0,configuration([(i, 4),(h, 9)])).
reseaux( 5,0,configuration([(i, 5),(h,16)])).
reseaux( 6,0,configuration([(i, 6),(h,15)])).
reseaux( 7,0,configuration([(i, 7),(h,14)])).
reseaux( 8,0,configuration([(i, 8),(h,13)])).
reseaux( 9,0,configuration([(i, 9),(h, 4)])).
reseaux(10,0,configuration([(i,10),(h, 3)])).
reseaux(11,0,configuration([(i,11),(h, 2)])).
reseaux(12,0,configuration([(i,12),(h, 1)])).
reseaux(13,0,configuration([(i,13),(h, 8)])).
reseaux(14,0,configuration([(i,14),(h, 7)])).
reseaux(15,0,configuration([(i,15),(h, 6)])).
reseaux(16,0,configuration([(i,16),(h, 5)])).
```

Finalement, nous avons les faits définissant les paramètres de routage utilisés par le routeur hiérarchique.

pour figure 7.26

```
nb_carreau_en_x(13).
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(1850).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

pour figure 7.27

```
nb_carreau_en_x(11).
nb_carreau_en_y(5).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(1850).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

A1.9 FAITS DE L'EXEMPLE 9

Pour conclure l'annexe 1, nous présentons la liste des faits qui définissent le banc d'essai de l'exemple 9. En premier, nous avons l'ensemble de faits **composante**, permettant de définir toutes les broches des 4 composantes.

```
composante(1,nom(350,150,a),
           dessin(0,250,700,50),
           configuration([(  0, 0),
                          (100, 0),
                          (200, 0),
                          (300, 0),
                          (400, 0),
                          (500, 0),
                          (600, 0),
                          (700, 0),
                          (700,300),
                          (600,300),
                          (500,300),
                          (400,300),
                          (300,300),
                          (200,300),
                          (100,300),
                          (  0,300)])).


composante(2,nom(1400,150,b),
           dessin(1150,250,1850,50),
           configuration([(1150, 0),
                          (1250, 0),
                          (1350, 0),
                          (1450, 0),
                          (1550, 0),
                          (1650, 0),
                          (1750, 0),
                          (1850, 0),
                          (1850,300),
                          (1750,300),
                          (1650,300),
                          (1550,300),
                          (1450,300),
                          (1350,300),
                          (1250,300),
                          (1150,300)])).
```

```
composante(3,nom(1400,550,c),
             dessin(1150,650,1850,400),
             configuration([(1150,400),
                            (1250,400),
                            (1350,400),
                            (1450,400),
                            (1550,400),
                            (1650,400),
                            (1750,400),
                            (1850,400),
                            (1850,700),
                            (1750,700),
                            (1650,700),
                            (1550,700),
                            (1450,700),
                            (1350,700),
                            (1250,700),
                            (1150,700)])).


composante(4,nom(350,550,d),
             dessin(0,650,700,450),
             configuration([(  0,400),
                            (100,400),
                            (200,400),
                            (300,400),
                            (400,400),
                            (500,400),
                            (600,400),
                            (700,400),
                            (700,700),
                            (600,700),
                            (500,700),
                            (400,700),
                            (300,700),
                            (200,700),
                            (100,700),
                            (  0,700)])).
```

Nous définissons aussi les 28 connexions que le routeur hiérarchique tentera de router sur circuit imprimé. Les faits **reseaux** permettent de définir les extrémités de chaque connexion.

```
reseaux( 1,0,configuration([(d,16),(c, 9)])).
reseaux( 2,0,configuration([(d,15),(b, 7)])).
reseaux( 3,0,configuration([(d,14),(b, 6)])).
reseaux( 4,0,configuration([(d,13),(b, 5)])).
reseaux( 5,0,configuration([(d,12),(c, 6)])).
reseaux( 6,0,configuration([(d,11),(c, 8)])).
reseaux( 7,0,configuration([(d,10),(c,15)])).
reseaux( 8,0,configuration([(d, 9),(c,16)])).
reseaux( 9,0,configuration([(d, 9),(d, 7)])).
reseaux(10,0,configuration([(d, 8),(a, 7)])).
reseaux(11,0,configuration([(d, 5),(d, 2)])).
reseaux(12,0,configuration([(d, 1),(a, 9)])).
reseaux(13,0,configuration([(a,16),(b,16)])).
reseaux(14,0,configuration([(a,15),(b,10)])).
reseaux(15,0,configuration([(a, 8),(b,14)])).
reseaux(16,0,configuration([(a, 6),(c, 1)])).
reseaux(17,0,configuration([(a, 5),(c,10)])).
reseaux(18,0,configuration([(a, 4),(c,11)])).
reseaux(19,0,configuration([(a, 3),(c,12)])).
reseaux(20,0,configuration([(a, 2),(b, 3)])).
reseaux(21,0,configuration([(a, 1),(b, 1)])).
reseaux(22,0,configuration([(b, 2),(b, 8)])).
reseaux(23,0,configuration([(b, 3),(c,14)])).
reseaux(24,0,configuration([(b, 4),(b,11)])).
reseaux(25,0,configuration([(b, 9),(b,15)])).
reseaux(26,0,configuration([(b,16),(c, 7)])).
reseaux(27,0,configuration([(c, 2),(c, 5)])).
reseaux(28,0,configuration([(c,15),(c,13)])).
```

Finalement, nous avons les faits définissant les paramètres de routage.

```
nb_carreau_en_x(13).
nb_carreau_en_y(7).
couche(1,a).
couche(2,a).
grille(25).
nb_de_couche(2).
coor_max_x(1850).
coor_max_y(700).
coor_min_x(0).
coor_min_y(0).
```

ANNEXE 2

Nous présentons dans cette annexe, une décomposition des figures: 6.29, 6.31 et 6.32. La décomposition permet de visualiser les chemins qui se trouvent sur chacune des couches.

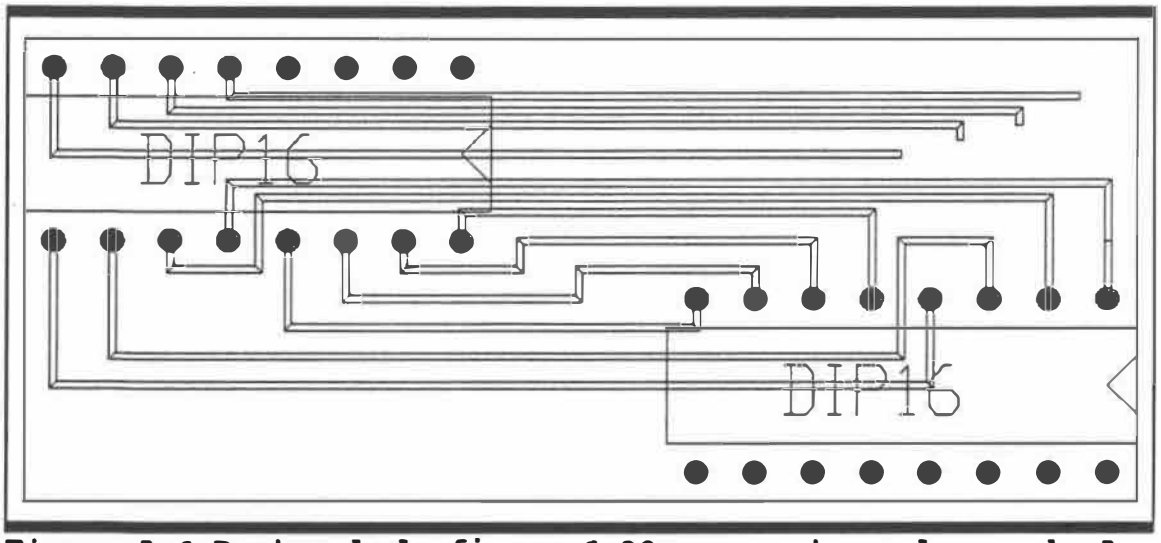


Figure A.1 Routes de la figure 6.29 reposant sur la couche A.

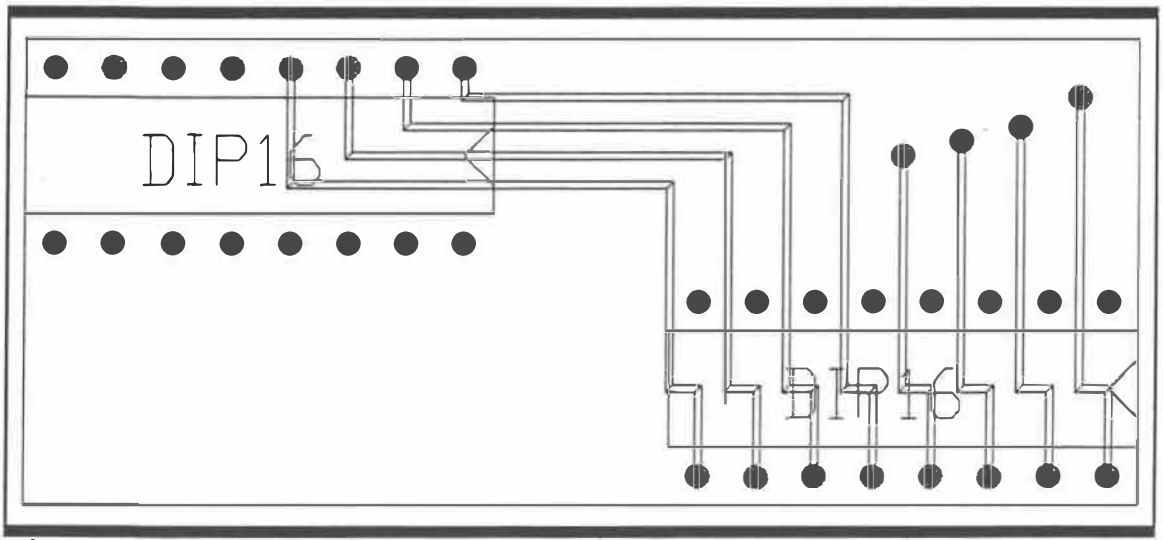


Figure A.2 Routes de la figure 6.29 reposant sur la couche B.

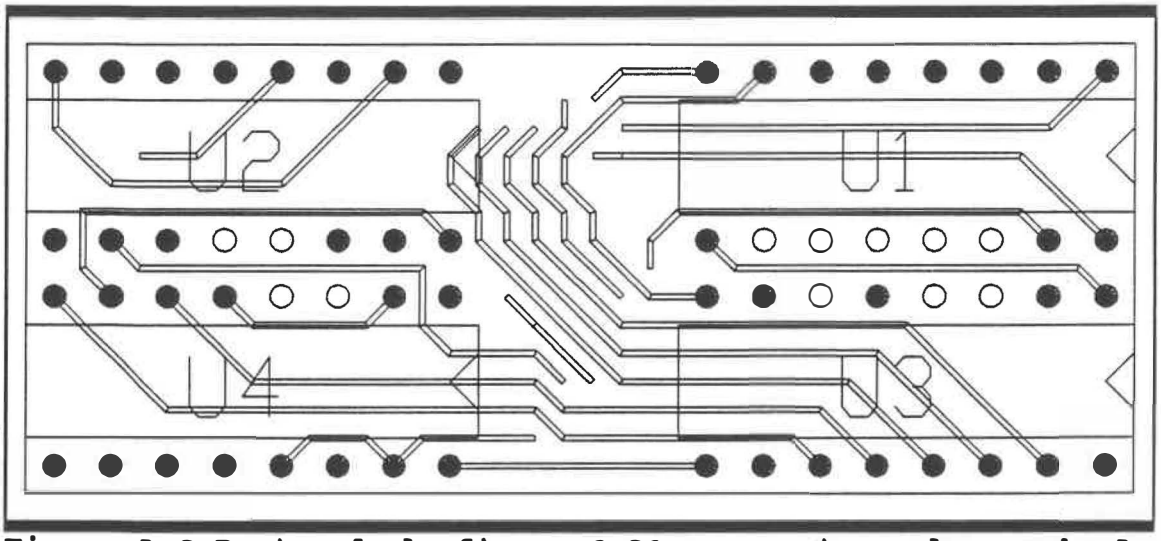


Figure A.3 Routes de la figure 6.31 reposant sur la couche A.

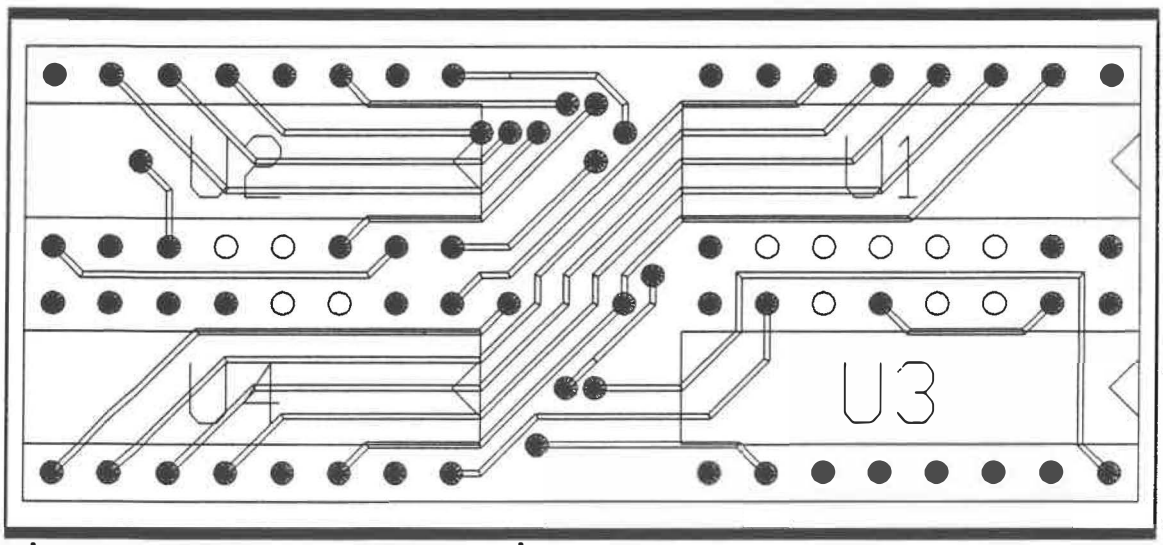


Figure A.4 Routes de la figure 6.31 reposant sur la couche B.

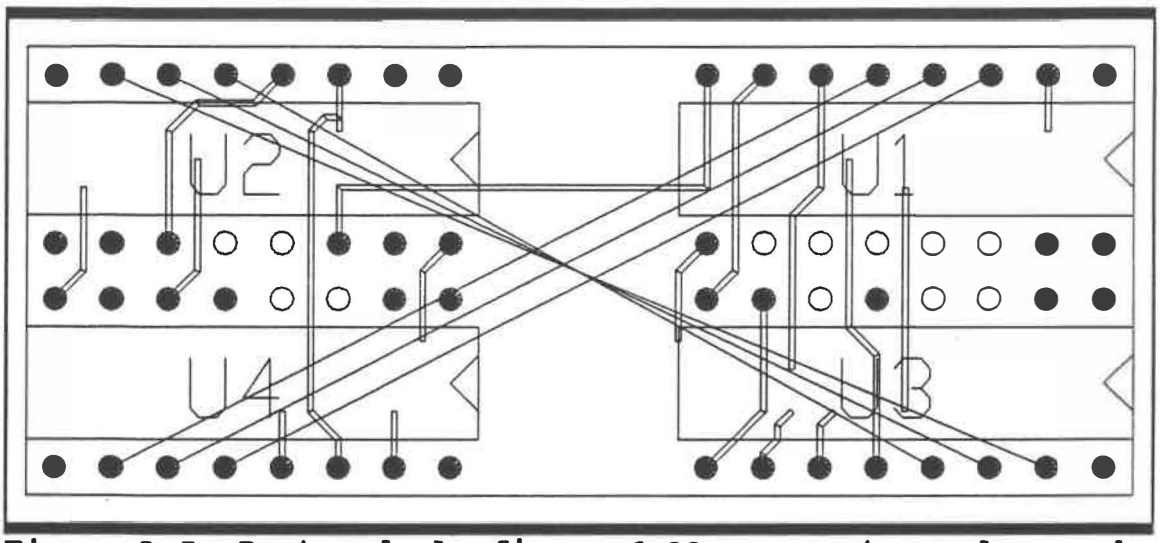


Figure A.5 Routes de la figure 6.32 reposant sur la couche A.

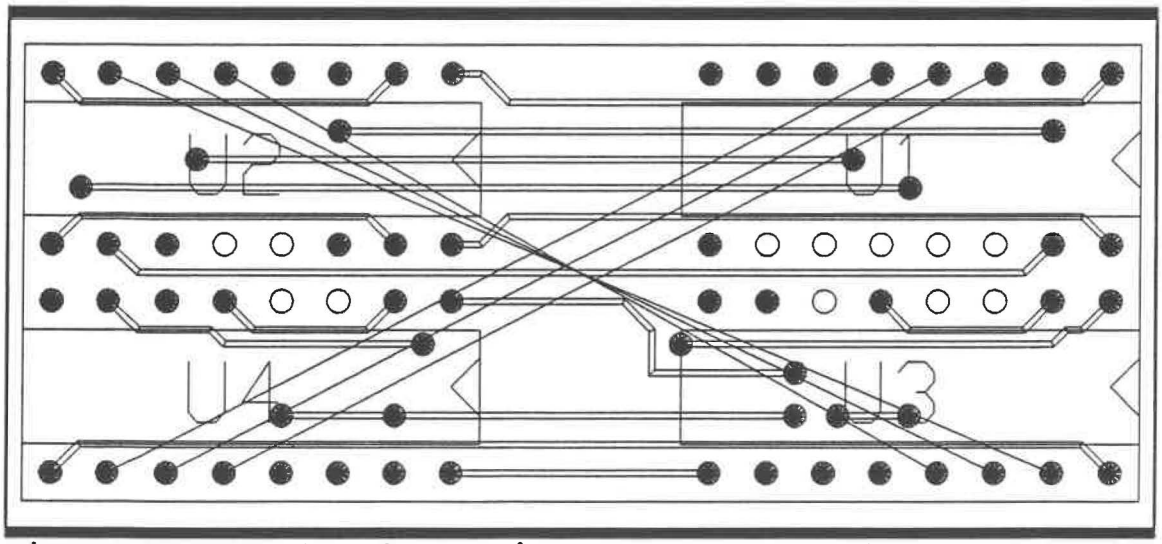


Figure A.6 Routes de la figure 6.32 reposant sur la couche B.

LEXIQUE

AC : Courant alternatif.

Bus de routage : Regroupement de plusieurs fils dont les chemins seront parallèles une fois routés sur le circuit imprimé.

Beigne de soudure : Disque de cuivre entre une traverse et un chemin qui est utilisé afin de souder une broche de composante et un chemin.

Canal (channel) : Région de routage avec deux frontières fixes, opposée l'une à l'autre et espacée d'une distance fixe. La dimension dans l'autre sens est maximale par rapport a la dimension du circuit imprimé.

Coin (corner) : Point d'intersection d'une ligne verticale et horizontale, mais qui ne forme pas de croix (+) ou un "T" (T).

Connexion : Couple de coordonnée ou d'identification de deux ports ou terminaux qui doivent être reliés par un fil.

Couche (layer) : Plan indépendant ou isolé électriquement sur lequel les fils peuvent être routés.

DC : Courant continu.

DIP : Configuration du boituer d'une composante électronique ayant deux rangées parallèles de broches.

Échangeur (switchbox) : Région de routage avec quatre frontières fixes. Habituellement région carrée.

Fil (wire) : Interconnexion entre des objets, c'est une connexion physique qui produit un contact électrique entre deux objets ou deux points d'un même objet et qui transmet un certain signal. Les fils qui se croisent, doivent avoir le même

signal et sont considérés comme un seul fil.

Frontière (edge) : Segments de droite parallèles ou perpendiculaires qui définissent un certain espace.

Grille de base : Le plus petit quadrillage de l'espace de routage dont chacun des carrés est adressable individuellement. On parle habituellement d'une grille de 25 ou 50 Mil.

Grille de travail : Région de routage contenant des endroits prédéfinis pour les routes, les ports, les traverses... Synonyme de grille de base.

Mil : Un millième de pouce.

Monde graphique : Base de données regroupant un ensemble d'objets graphiques.

Objet : Une cellule (cell) ou un circuit intégré qui sont des items

fonctionnels. Le but d'un routeur est d'interconnecter des objets fonctionnels afin d'en faire un objet plus fonctionnel ou plus gros (complexe).

Obstacle fixe : Obstacle au routage qui ne peut pas être violé et qui ne sera pas changé. Ce sont des fils placés par l'usager, les ports de composantes après le placement, les bords de la plaque du circuit imprimé...

Obstacle mobile : Obstacle qui peut être modifié par le programme tout en conservant les contraintes demandées comme les connexions...

Port ou Point(pin) : Points auxquels sont connectés les fils sur un objet.

Région de routage : Aire à l'intérieur de frontières rectilinéaires qui est utilisable pour le routage.

Réseau (net) : Ensemble de terminaux à être connectés par des fils. Est constitué de ports, ou routes ou fils ou de traverses qui voyagent un signal commun.

Route : Ensemble de segments de ligne horizontale et/ou verticale qui produisent tout ou une partie d'un réseau. Ce sont des fils mais avec des directions prédéfinies.

Signal commun : Signal dans des fils qui sont reliés ou connectés ensemble.

SIP : Configuration du boitier d'une composante électronique ayant une seule rangée de broches.

Terminal : Ports sur la frontière d'une région de routage.

Traverse (vias) : Fil qui traverse normalement (direction normale) deux ou plusieurs couches et qui permet d'interconnecter deux ou plusieurs points sur des couches différentes.