Mechanization of Spatial Reasoning for Automatic Pipe Layout Design

by
David Zhu and Jean-Claude Latombe

TECHNICAL REPORT
Number 39

October, 1990

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 95305-4020

Mechanization of Spatial Reasoning for Automatic Pipe Layout Design

David Zhu and Jean-Claude Latombe
Robotics Laboratory
Department of Computer Science, Stanford University
Stanford, CA 94305, USA

Abstract

Artificial Intelligence has been very active in developing high-level symbolic reasoning paradigms that have resulted in practical expert systems. However, with a few exceptions, it has paid little attention to the automation of spatial reasoning. On the other hand, spatial reasoning has attracted the interest of several researchers in Robotics. One of the important problems that have been investigated is motion planning, and very significant results have been obtained. This paper describes an implemented system for designing pipe layouts automatically using motion planning techniques. It introduces a new approach to pipe layout design automation in which pipe routes are treated as trajectories left behind by rigid objects ("robots"). We have implemented this approach in a basic Pipe Router that is described in detail in this paper. We have extended this router in order to make it capable of treating a variety of other constraints which are typical of practical pipe layout design problems. These constraints relate to the process carried out in the pipes, to the design of their mechanical support, and to the constructability and the ease of operation and maintenance of the designed pipe systems.

Acknowledgements: This research project was partially funded by a grant from CIFE (Center for Integrated Facility Engineering) at Stanford and by Darpa Contract N00014-88-k-0620 (office of Naval Research). The authors would like to thank Prof. P. Teicholz for suggesting the topic of the project and for providing continuous encouragements. They also thank the R&D group at Bechtel National Inc. and Dr. A. Riitahuhta of Tampela Inc. (Finland) for providing technical advice throughout the project.

1 Introduction

Artificial Intelligence (AI) has been very active in developing high-level symbolic reasoning paradigms that have resulted in practical expert systems. However, with a few exceptions in the area of "qualitative reasoning", it has paid little attention to the automation of spatial reasoning. Consequently, AI has been less successful in domains that require involved spatial reasoning capabilities such as design and process planning. In design, one must reason about the relationship between function and shape. In process planning, it is necessary to reason about accessibility, stability, and tolerances. The integration of design and process planning to ensure constructability also relies on sophisticated spatial reasoning capabilities. The geometrical and physical knowledge needed in these domains is not conveniently captured in logic-oriented constructs such as frames and production rules which are typical in knowledge-based expert systems.

On the other hand, during the past ten years or so, spatial reasoning has attracted the interest of several researchers in Robotics. One of the important problems that have been investigated is *motion planning* [Schwartz, Sharir and Hopcroft, 1987] [Latombe, 1990]. The basic motion planning problem is to compute a collision-free path for a given object among obstacles from an initial configuration of the object to a goal configuration. Very significant results have been obtained over the last few years. Though theoretical results indicate that motion planning is intrinsically difficult (PSPACE-hard), experimental results with implemented planners are more encouraging and suggest that the problem is tractable in many practical situations [Barraquand and Latombe, 1989a] [Faverjon and Tournassoud, 1989].

A recent stream of effort has been devoted to exploring the application of spatial reasoning techniques developed in Robotics to other areas, especially mechanical design and process planning [Latombe, 1988] [Donald, 1989] [Natarajan, 1989] [Wilson and Rit, 1990]. The research reported in this paper participates in this effort. It has resulted in the design and implementation of an operational system for designing pipe layouts automatically using motion planning techniques. Pipe layout design (PLD) is similar to motion planning in that both problems consist of finding geometric paths satisfying input constraints. In this paper we exploit this similarity and propose a new approach to PLD automation in which pipe routes are treated as trajectories left behind by rigid objects.

PLD is a problem of major significance in chemical, refinery, and power plant design [Gunn and Al-Asadi, 1987], and in ship and submarine design [Sheridan, 1976]. It is complex and time consuming due to the large number of pipes and constraints involved. It often requires many iterations before a satisfactory design can be obtained. Therefore, any progress toward automating PLD will benefit these industries in terms of higher quality, reduced cost, and shorter turn-around time. Several CAD systems have been built to help engineers solving this problem (e.g. [Bechtel, 1986] [Mitsuta et al., 1986] [Kobayashi et al., 1986]), but none of them is truly satisfactory. Either they only consider peripheral issues of the PLD problem (e.g. they are essentially graphical display systems) or they do not actually address the geometrical issues raised by the problem (e.g. these issues are hidden in a numerical function to be optimized or in rules of thumb believed to mimick human expertise). In contrast, our approach takes pipe routing for what it actually is, that is, a geometric planning problem. By treating geometric issues in a systematic fashion, it can deal with a wide variety of constraints that are typical of practical pipe layout design problems, including constraints aimed at facilitating pipe construction, operation, and maintenance. Our approach allows one to have better control over the achievement of each individual constraint than the optimization approach. It makes our system more robust than pure expert systems which tend to work poorly, if at all, when they face situations not anticipated in their knowledge. On the other hand, our system could certainly benefit from the other approaches. For example, one could run optimization techniques to improve further the layouts it produces, and apply expert rules to increase its efficiency in stereotyped situations.

In another domain, VLSI routing, there has been a great deal of effort aimed at developing efficient routing systems (e.g. [Sechen, 1988]). However, though VLSI routing and pipe routing certainly have external similarities, the constraints to be satisfied by the routes in the two problems are quite different. These differences make the transfer of routing methods from one problem to the other more difficult (if at all possible) than one would expect. Another related routing problem is the design of hydraulic manifold blocks [Chambon et al., 1987].

Although the approach presented in this paper is new, a few routers have already been proposed which apply techniques closely related to motion planning techniques. But this relation was not explicitly mentioned, nor exploited in depth. In particular, a "visibility graph" technique was presented in [Wangdahl, 1974] to route pipes in ships. A "potential field" technique was proposed in [Hasan and Liu, 1987] for PCB routing. On the other hand, the application of motion planning techniques to automated structure design, a problem with some similarities with pipe routing, was suggested in [Donald, 1983].

This paper is organized as follows. Section 2 provides the context for the rest of the paper. We present the goal of our research in more detail, we give an overview of our motion-planning-based approach, and we describe the system architecture deriving from this approach. This architecture is organized around a core module called the *Pipe Router*. In the other sections we mainly focus on the technical issues which underlie the development of this module. In Section 3 we give the necessary background in motion planning. In Section 4 we describe a basic Pipe Router embedding our motion-planning-based approach. In Section 5 we present various extensions of the basic router which allow us to deal with additional constraints in PLD problems. In Section 6 we present our current research in pipe layout design.

2 Overview

The PLD problem consists of finding the layout of pipes in a two-dimensional (2D) or three-dimensional (3D) workspace. The pipes must connect nozzles (or terminals) of given locations and avoid collision with obstacles (machines, walls, structure) of given locations and shapes. The layout must often satisfy additional constraints related to the process carried out in the pipes (e.g. a water pipe should not be hanged above any electrical equipment) and to the design of mechanical support for the pipes (e.g. pipes should traverse designated structures providing support), as well as to the constructability and the ease of operation and maintenance of the pipe system (e.g. valves should be accessible). Furthermore, there usually are optimality criteria regarding the length and number of turns of the pipes which should be taken into account in order to produce a satisfactory layout. The goal of our research has been to develop an automatic PLD system capable of addressing this variety of constraints in a systematic fashion.

Our approach to automatic PLD is based on using spatial reasoning techniques originally developed for robot motion planning. In this approach, we regard each pipe as the trace left behind by a rigid object moving in the pipe workspace. Using this metaphor, we transform the PLD problem into the problem of planning the motion of a collection of robots. Among the various approaches that have been proposed for solving the robot motion planning problem [Latombe, 1990], we have selected the so-called *approximate cell decomposition* approach [Brooks and Lozano-Pérez, 1982] [Zhu and Latombe, 1989]. Basically, this approach consists of representing the subset of collision-free configurations of the robot as a collection of cells having the same predefined shape. The connectivity graph representing the adjacency relation among these cells is then constructed and searched for a channel (sequence of adjacent cells) connecting the initial configuration to the goal configuration of the robot. If the search terminates successfully, a path is extracted from the generated channel. We have selected this approach because it has a unique set of attractive features. It is rather easy to implement, relatively efficient and resolution-complete [Zhu and Latombe, 1989]. It also generates "channels" (sets of paths) rather than single paths, thus enabling a lesser-commitment approach to pipe routing.

Our PLD system has evolved from this approach. The pipes are planned one at a time. Planning a pipe corresponds to generating the path of a robot among the

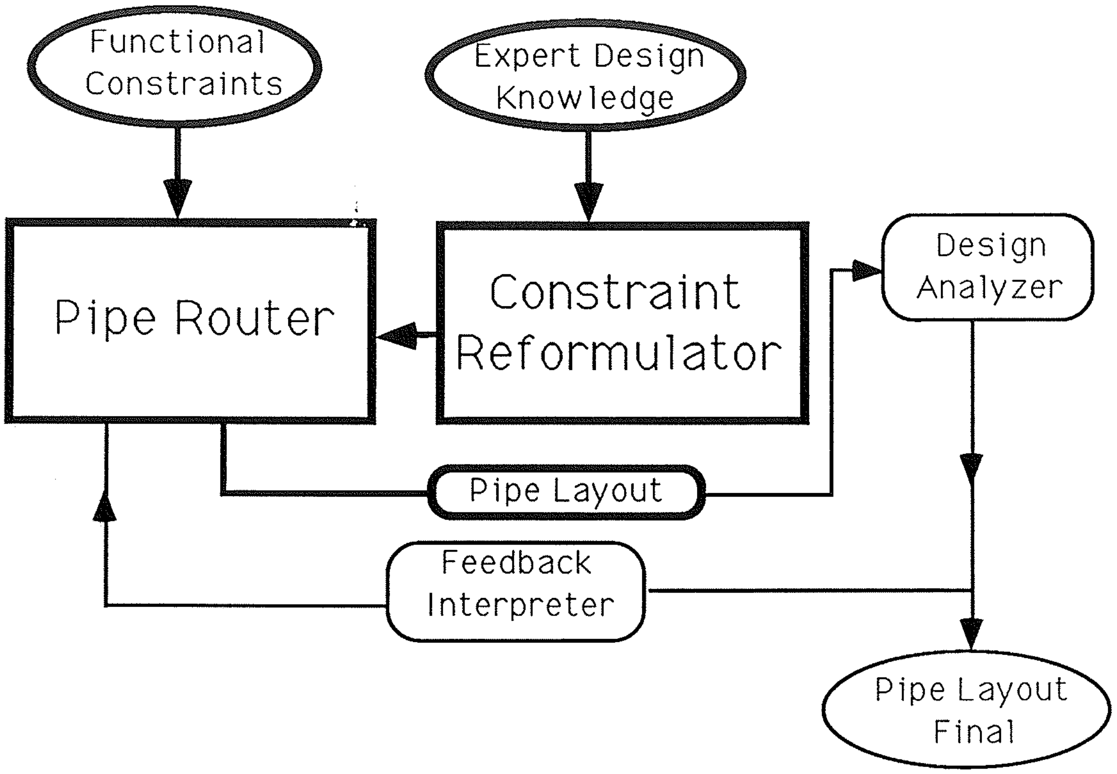


Figure 1: Architecture of Our PLD System

obstacles and the previously planned pipes. In order to deal with the various design constraints applying to the pipe layout to be generated, we reformulate them into geometric constraints applying to the paths of the "robots" tracing the pipes. Some of these constraints restrict the possible *locations* of the pipes; they are essentially treated as additional obstacles to be avoided. The other constraints put requirements on the *shapes* of the pipes; they restrict the set of valid channels in a connectivity graph and are treated using *history-dependent search* techniques.

The PLD system deriving from this approach consists of four main modules depicted in Figure 1: the *Pipe Router*, the *Constraint Reformulator*, the *Design Analyzer*, and the *Feedback Interpreter*. The inputs to the system are divided into two parts: the *Functional Constraints* and the *Expert Design Knowledge*. The output of the system is the *Pipe Layout (Final)*.

The Functional Constraints give the locations of the terminals of the pipes and the shape and locations of the obstacles in the workspace. They are extracted from a document known as the Piping and Instrument Diagram (P&I D) in the construction industry. They are directly input into the Pipe Router.

The Functional Constraints alone do not completely specify a pipe layout. As indicated above, we have identified three types of additional constraints: process constraints, structural constraints, and accessibility constraints. A pipe layout must also satisfy these constraints (not all of them are strict constraints). These constraints can be quite complex to express and to verify (e.g. they may require extensive stress analysis using finite-element methods). An experienced pipe designer typically abstracts them into simplified constraints which we call Expert Design Constraints. These constraints cannot be used directly by the Pipe Router. The purpose of the Constraint Reformulator is to rewrite them into geometric constraints usable by the Pipe Router. In Section 5 we will give various examples of Expert Design Constraints and we will discuss their transformation into geometric constraints.

The Pipe Router attempts to generate a pipe layout that satisfies the input Functional Constraints and the reformulated Expert Design Constraints. Since the latter are abstractions of more complex constraints, the generated layout, if any, must be submitted to a post-design analysis in the *Design Analysis* module. This may involve detailed process and stress analysis, to ensure that the real process and structural constraints are actually satisfied, and simulation of construction, operation and maintenance plans, to verify that they are feasible. If the layout fails in this analysis, for example if the pipe support designer fails to come up with a detailed design of the pipe supports, then information related to the shortcomings of the layout is fed back to the Pipe Router through the *Feedback Interpreter*. The layout is then modified by taking this information into consideration.

At the current stage of our research, only the subset of Figure 1 shown in bold lines, which include the Pipe Router and the Constraint Reformulator, have been designed and implemented. Therefore, in this paper we focus our attention on the Pipe Router (Section 4) and the Constraint Reformulator (Section 5). Prior to that, we give some background in robot motion planning.

3 Background in Motion Planning

The basic robot motion planning problem can be stated as followed: Given a set of obstacles $\mathcal{B}_i$ ($i = 1, ..., q$) in a workspace $\mathcal{W}$, find a collision-free path for a robot $\mathcal{A}$ from a given initial configuration $\mathbf{q}_{init}$ to a given goal configuration $\mathbf{q}_{goal}$. Figure 2 illustrates a typical example of the basic motion planning problem where the robot is a rigid object (shown grey) allowed to translate and rotate among obstacles (shown black) in a 2D workspace. It displays a collision-free path followed by this robot.

We model $\mathcal{W}$ as the Euclidean space $\mathbf{R}^2$ (2D case) or $\mathbf{R}^3$ (3D case). A configuration $\mathbf{q}$ of the robot $\mathcal{A}$ is a specification of the position of every point in $\mathcal{A}$ with respect to a coordinate system embedded in $\mathcal{W}$. The **configuration space** $\mathcal{C}$ of $\mathcal{A}$ is the set of all the possible configurations of $\mathcal{A}$ [Lozano-Pérez, 1983]. For example, if $\mathcal{A}$ is

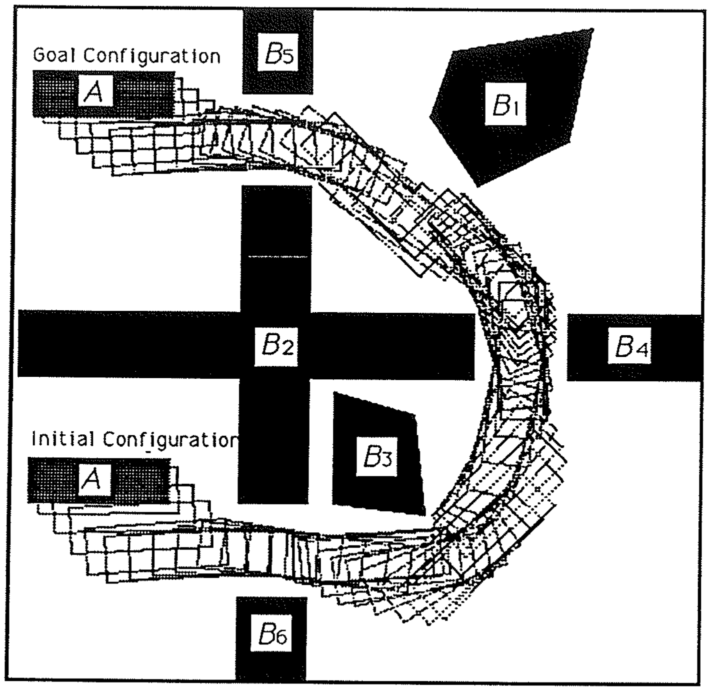


Figure 2: Motion Planning Example

a disc in $\mathbf{R}^2$ or a ball in $\mathbf{R}^3$ (the two cases of interest in the rest of this paper), a configuration $\mathbf{q}$ can be represented as the coordinates (x, y) or (x, y, z) of the center of the disc/ball. Then the workspace and the configuration space are both copies of $\mathbf{R}^N$, with $N = 2$ or 3.

Every obstacle $\mathcal{B}_i$ $(i \in [1, q])$ is mapped into $\mathcal{C}$ as a region $\mathcal{CB}_i$ called a **C-obstacle** which is defined by:

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C} \ / \ \mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\}$$

where $\mathcal{A}(\mathbf{q})$ designates the region occupied by $\mathcal{A}$ at configuration $\mathbf{q}$. The region:

$$\mathcal{C}_{free} = \mathcal{C} \setminus \bigcup_{i \in [1,q]} \mathcal{CB}_i$$

is called the **free space.** A **collision-free path** (more simply, a **free path**) between two configurations $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ is any continuous map $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = \mathbf{q}_{init}$ and $\tau(1) = \mathbf{q}_{goal}$. The basic motion planning problem is to find such a free path.

Figure 3 illustrates the construction of C-obstacles when the robot is a disc of radius r and the obstacles are polygons. The C-obstacles are obtained by growing the workspace obstacles isotropically by the radius r.

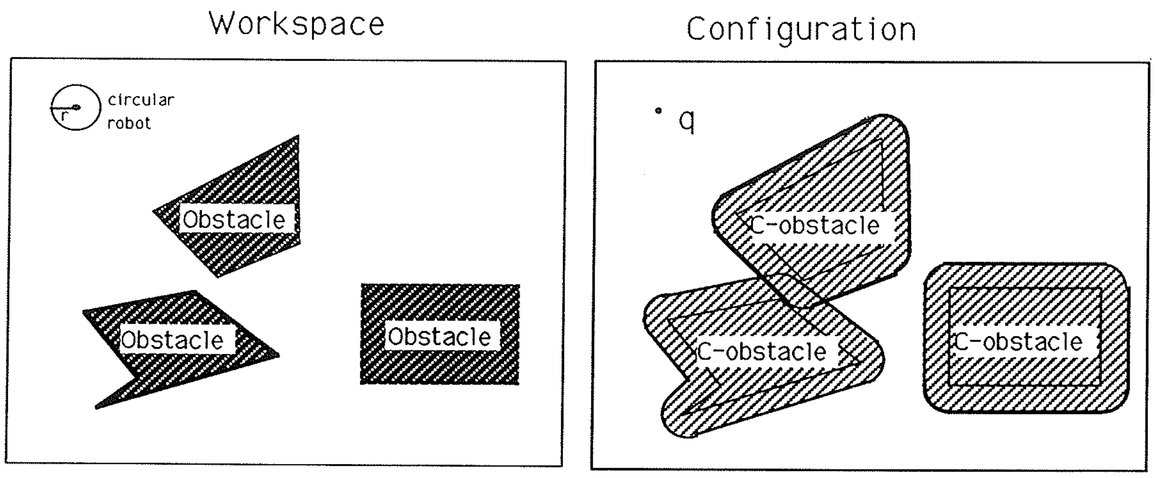


Figure 3: Construction of C-Obstacles

Several general approaches to robot motion planning have been developed [Latombe, 1990]:

- The *roadmap approach* consists of capturing the connectivity of the free space in a network of one-dimensional curves lying in the free space or its closure [Ó'Dúnlaing and Yap, 1982]. This network is then used as a set of standardized paths for connecting the initial to the goal configuration.
- The *cell decomposition approach* consists of representing (exactly or approximately) the free space as a collection of cells of rather simple shapes and searching the connectivity graph representing the adjacency relation among these cells [Schwartz and Sharir, 1983].
- The *potential field approach* regards the robot represented as a point in its configuration space as a particle under the influence of a potential field combining an attractive potential generated by the goal and a repulsive potential generated by the C-obstacles [Khatib, 1986].

The cell decomposition methods can be divided further between *exact* and *approximate* methods. The former represent the free space exactly, while the latter represent it approximately as a collection of cells of predetermined shape.

The approximate cell decomposition approach (used in our system) was introduced in [Brooks and Lozano-Pérez, 1983], with subsequent contributions by other researchers. In its most typical form, it consists of decomposing the robot's configuration space into rectangloid cells at successive levels of approximation. Cells are classified as EMPTY or FULL, depending on whether they lie entirely outside or entirely inside the C-obstacles. If they are neither EMPTY, nor FULL, they are labeled as MIXED. At each level of approximation, the planner searches the connectivity graph of these cells for a channel of adjacent EMPTY cells connecting the initial configuration of the robot to its goal configuration. If no such sequence is found, it iteratively decomposes some MIXED cells into smaller cells, label them appropriately, and searches the updated connectivity graph for a channel. The process ends when a solution has been found, or it is guaranteed that no solution can be found, or MIXED cells have become smaller than some prespecified size.

In our PLD system, we use a non-hierarchical version of the approximate cell decomposition approach. This means that we build a single decomposition of the configuration space. If we fail to find a path, this decomposition is not refined into a finer one. Indeed, when the robot is a disc or a ball, the particular geometry of the C-obstacles makes it possible to generate a rather precise conservative approximation of the free space in a single pass, with a reasonable number of cells. In addition, the cross-section of a pipe is usually small relative to the size of the obstacles. In any case, the parameters specifying the precision of this approximation can be set by the system's user as desired.

The motion planning approaches listed above can be generalized to the case where there are multiple robots $\mathcal{A}_1, ..., \mathcal{A}_p$ moving in the same workspace. This extension is important since we want to generate the layout of many pipes, each being traced out by its own "robot". One way to proceed is to consider the various robots as the components of a single, multi-bodied robot $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_p\}$. The configuration space of $\mathcal{A}$ is $\mathcal{C} = \mathcal{C}_1 \times ... \times \mathcal{C}_p$, with $\mathcal{C}_k$ ($k \in [1,p]$) denoting the configuration space of $\mathcal{A}_k$. The obstacles are mapped into $\mathcal{C}$ and a free path is planned among the C-obstacles. Such a path determines a set of p *coordinated* free paths for the robots $\mathcal{A}_k$ in their respective configuration spaces.

The above approach to multi-robot motion planning is called the *centralized planning* approach. Although it solves the problem in theory, it suffers, however, from a major drawback: the dimension of $\mathcal{C}$ is equal to p times the dimension of each individual space $\mathcal{C}_k$ (assuming all these spaces have the same dimension), and it has been shown that finding a free path in a configuration space of dimension m requires exponential time in m [Reif, 1979]. Hence, the approach tends to be intractable. In the particular case of the approximate cell decomposition method, it yields a number of cells that is exponential in the dimension of $\mathcal{C}$. Another approach, which we use in our PLD system, is called the *prioritizing approach* [Erdmann and Lozano-Pérez, 1986]. It consists of planning the paths for the robots, one at a time. For each robot, the considered obstacles are the workspace obstacles and those robots whose motions have been already planned. This approach assumes implicitly that the interactions among the robots are relatively weak. If this assumption is roughly satisfied, this approach tends to run in time exponential in the maximal dimension of the $\mathcal{C}_k$'s (i.e.

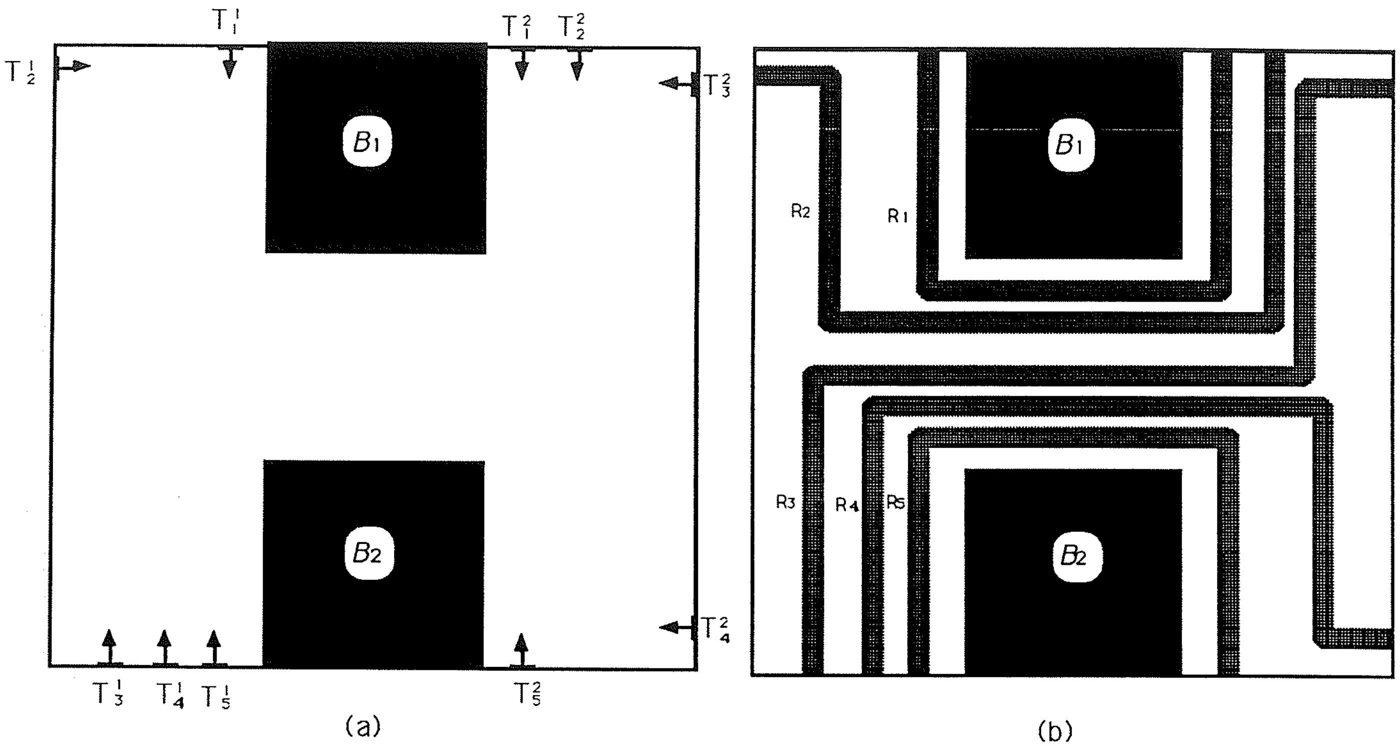


Figure 4: Example of the Basic PLD Problem

2 or 3 in the application to PLD), which is more tractable. At the other extreme, if the assumption is truly not satisfied, the approach may need a prohibitively large number of backtracking operations.

4 Pipe Routing

4.1 Basic Routing Problem

Let $\mathcal{W}$ be a bounded, connected subset of $\mathbf{R}^N$, with $N = 2$ or 3, representing the pipe **workspace**. A **pipe specification** $\mathbf{P}_k$ is a triple (T_k^1, T_k^2, r_k), where T_k^1 and T_k^2 are the two distinct points in $\mathcal{W}$ representing the **terminals** that the pipe should connect, and r_k is the radius of the pipe. A **route** for $\mathbf{P}_k$ is the region $\mathcal{R}_k$ in $\mathcal{W}$ that is swept out by a disc (in 2D) or a ball (in 3D) of radius r_k when its center moves along a curvilinear line L_k connecting T_k^1 and T_k^2. This line is called the **path** of the pipe route.

We formulate the **basic PLD problem** as follows:

Given a collection of obstacles $\mathcal{B}_i$ $(i = 1, ..., q)$ in $\mathcal{W}$ and a set of pipe specifications $\mathbf{P}_k$ ($k = 1, ..., p$), compute routes $\mathcal{R}_k \subset \mathcal{W}$ ($k = 1, ..., p$), so that:

- no route intersects an obstacle, i.e. : $\forall i \in [1, q], \forall k \in [1, p] : \mathcal{B}_i \bigcap \mathcal{R}_k = \emptyset$;

- no two routes intersect each other, i.e. : $\forall k, k' \in [1, p], k \neq k' : \mathcal{R}_k \bigcap \mathcal{R}_{k'} = \emptyset$.

(Since the routes must lie in $\mathcal{W}$, the boundary of $\mathcal{W}$ is treated as the boundary of an additional obstacle enclosing the workspace. We denote this "obstacle" by $\mathcal{B}_0$.)

Figure 4 shows an example of the basic PLD problem in a 2D rectangular workspace. In this example, there are two polygonal obstacles $\mathcal{B}_1$ and $\mathcal{B}_2$ (shown dark) and five pipe specifications (Figure 4.a). A solution to this problem is shown in Figure 4.b.

In the sequel, we make the following assumptions:

- Both the workspace $\mathcal{W}$ and the obstacles $\mathcal{B}_i$ are modeled as polygons or polyhedra.

- All the terminals are located in the boundary of the region $\mathcal{E} = \mathcal{W} \setminus \cup_i \mathcal{B}_i$. At every terminal the path of the corresponding pipe is perpendicular to the boundary of $\mathcal{E}$.

- The path of every pipe is a sequence of straight and circular segments. All the circular segments in the same path L_k have the same radius ρ_k specified as an additional parameter in the pipe specification.

- A Cartesian coordinate system $\mathcal{F}_{\mathcal{W}}$ is embedded in $\mathcal{W}$. Every straight segment in the path of a pipe is parallel to one of the axes of $\mathcal{F}_{\mathcal{W}}$ and (in the 3D case) every circular segment is parallel to one of the planes determined by two major axes.

These assumptions are all reasonable in practice. In most cases, $\mathcal{W}$ is a parallelepiped bounded by "walls". The axes of $\mathcal{F}_{\mathcal{W}}$ are selected parallel to the edges of this parallelepiped. "Good" pipe design tends to avoid slanted pipes.

4.2 Outline of the Algorithm

Our algorithm for solving the basic PLD problem is an adaptation of the approximate cell decomposition method developed in robot motion planning. It considers the pipes $\mathbf{P}_k$, $k \in [1, p]$, in sequence. For each pipe $\mathbf{P}_k = (T_k^1, T_k^2, r_k)$, it plans a collision-free path for a ball of radius r_k from T_k^1 to T_k^2 among the obstacles $\mathcal{B}_0, ..., \mathcal{B}_q$, on the one hand, and the pipe routes $\mathcal{R}_1, \ldots, \mathcal{R}_{k-1}$ already constructed, on the other hand. If such a path L_k is found, it determines a route $\mathcal{R}_k$. Otherwise, the algorithm must backtrack, i.e. it must change some of the routes $\mathcal{R}_1, \ldots, \mathcal{R}_{k-1}$ in order to make room for $\mathcal{R}_k$. As mentioned in the previous section, this corresponds to applying a prioritizing planning approach. A general prioritizing heuristics to reduce backtracking is to route bigger pipes before smaller ones.

We let $\mathcal{A}_k$ designate the "robot" tracing out the route $\mathcal{R}_k$ and $\mathcal{C}_k$ denote its configuration space. The algorithm that is executed at each iteration in order to compute the path L_k of $\mathcal{R}_k$ consists of the following three steps:

1. Compute the C-obstacles due to the obstacles $\mathcal{B}_0, ..., \mathcal{B}_q$ and the existing routes $\mathcal{R}_1, ..., \mathcal{R}_{k-1}$ by growing them by the radius r_k of $\mathbf{P}_k$. Approximate the free space in $\mathcal{C}_k$ as a conservative collection of (EMPTY) rectangloid cells.
2. Construct the connectivity graph of these cells and search this graph for a channel, i.e. a sequence of cells such that the boundary of the first (resp. the last) cell contains T_k^1 (resp. T_k^2) and any two successive cells are adjacent.
3. If the search terminates sucessfully, extract a path L_k from the channel. Otherwise return failure.

In order to facilitate the subsequent extraction of paths, the rectangloid cells generated at Step 1 have all their sides parallel to axes of $\mathcal{F}_\mathcal{W}$. Two cells are adjacent if the intersection of their boundaries is a segment of non-zero length (in $\mathbf{R}^2$) or a rectangle of non-zero area (in $\mathbf{R}^3$).

Figure 5 illustrates the operations performed by this algorithm in a 2D workspace for a single pipe $\mathbf{P}_1$. Figure 5.a shows the construction of the C-obstacles due to $\mathcal{B}_1$ and $\mathcal{B}_2$ in the configuration space $\mathcal{C}_1$. Figure 5.b displays an approximation of the free subspace in $\mathcal{C}_1$ in the form of a collection of rectangular cells. Figure 5.c shows both a channel between the two terminals T_1^1 and T_1^2 and a path L_1 in this channel. Figure 5.d presents the corresponding route $\mathcal{R}_1$ in the workspace.

The operations carried out by the above algorithm are described in more detail in the following subsections. We also present the backtracking mechanism that is activated when the path planning algorithm fails to construct a path L_k.

4.3 Cell Decomposition

The first step of the above algorithm generates an explicit representation of the space that is available for the pipe $\mathbf{P}_k$ — i.e. the free space in $\mathcal{C}_k$ — in the form of a collection of rectangloid cells. To this purpose, it first maps the obstacles $\mathcal{B}_i$ ($i = 0$ to q) and the routes $\mathcal{R}_j$ ($j = 1$ to $k-1$) to C-obstacles in $\mathcal{C}_k$. We denote these C-obstacles by $\mathcal{CB}_i$ and $\mathcal{CR}_j$, respectively. Next, it computes a bounding approximation of these C-obstacles in the form of a collection of rectangloids whose edges are oriented along $\mathcal{F}_\mathcal{W}$'s axes. The complement in $\mathcal{C}_k$ of all these rectangloids is easily computed in the form of a collection of rectangloids also oriented along $\mathcal{F}_\mathcal{W}$'s axes. This second collection of rectangloids are the cells of the conservative representation of the free space in $\mathcal{C}_k$. This approximation technique has been shown to be more efficient

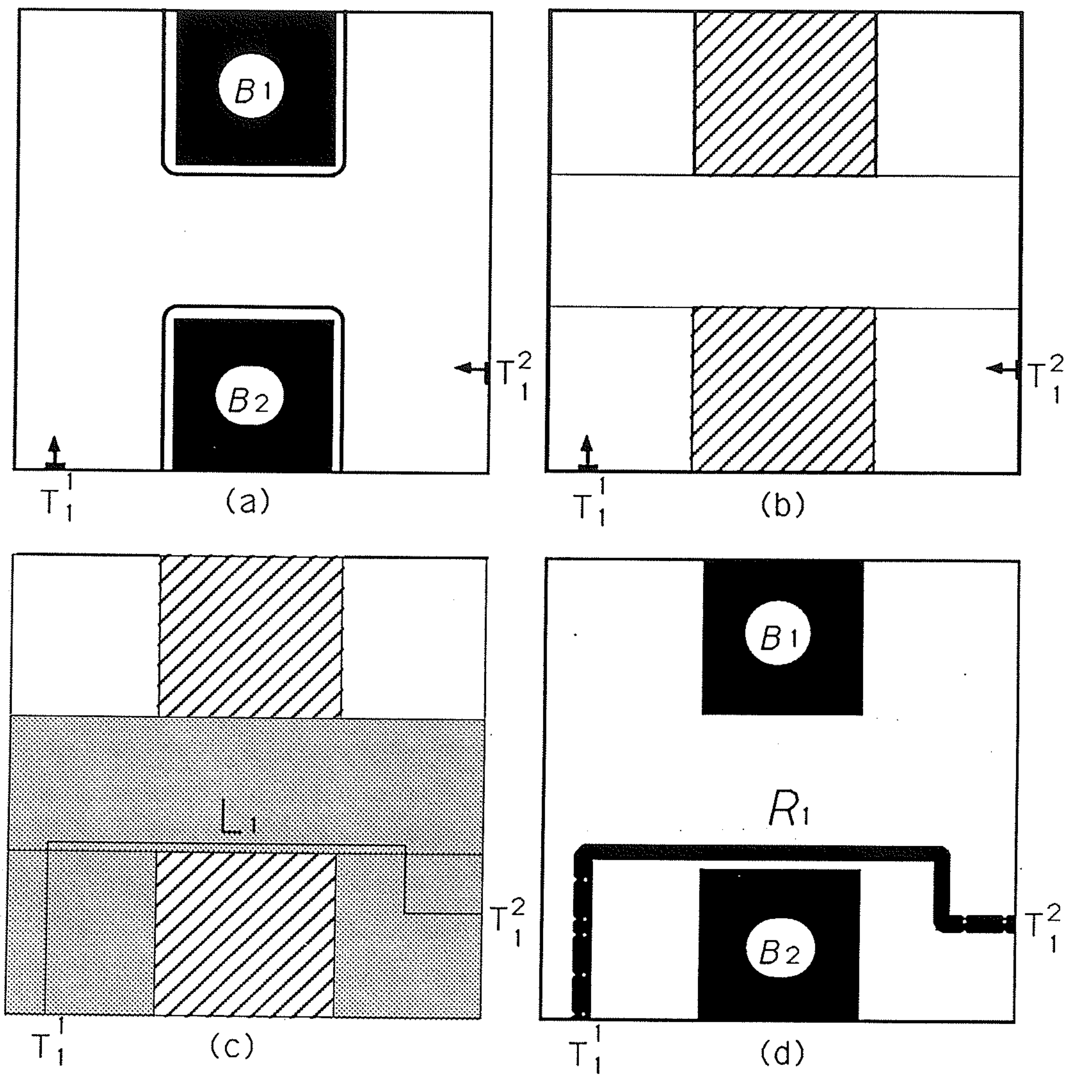
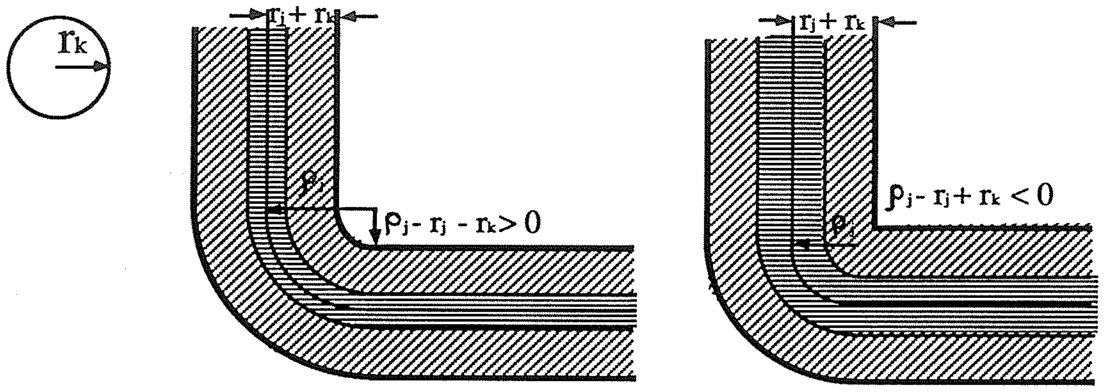


Figure 5: Pipe Routing Example

than the more classical quadtree/octree decomposition, in the sense that it usually generates a more precise approximation of the free space with a much smaller number of cells (which subsequently facilitates graph searching) [Zhu and Latombe, 1989].

4.3.1 2D Case

$\mathcal{CB}_i$, for $i = 0$ to q, is obtained by growing $\mathcal{B}_i$ isotropically by the radius r_k of the pipe to be routed. Since we model $\mathcal{B}_i$ as a polygon, $\mathcal{CB}_i$ is a generalized polygon whose boundary is a sequence of straight and circular edges (see Figure 3). Each straight edge of $\mathcal{CB}_i$ is the locus of the center of the disc $\mathcal{A}_k$ when it moves with its boundary sliding in contact along an edge of $\mathcal{B}_i$. It is obtained by translating



(a) (b)

Figure 6: C-obstacle Due a Pipe Elbow

the corresponding edge of $\mathcal{B}_i$ by r_k along its outgoing normal direction. Each circular edge is the locus of the center of $\mathcal{A}_k$ while it moves with its boundary being in contact with a convex vertex V of $\mathcal{B}_i$. It is a circular arc of radius r_k centered at V. Let n be the number of vertices of $\mathcal{B}_i$. Algorithms compute $\mathcal{CB}_i$ in $O(n)$ time, if $\mathcal{B}_i$ is convex, and in $O(n^2 \log n)$ time, otherwise [Latombe, 1990]. In the second case, if the intersection of $\mathcal{B}_i$ with any disc of radius r_k contains a small number of vertices (that is then considered to be constant), a reasonable situation where $\mathcal{B}_i$ is said to have *bounded local complexity* [Sifrony and Sharir, 1987], the time complexity reduces to $O(n \log n)$.

$\mathcal{CR}_j$, for $j = 1$ to $k - 1$, is also obtained by growing $\mathcal{R}_j$ isotropically by the radius r_k. It has the form of a route of radius $r_j + r_k$ following the same path L_j as $\mathcal{R}_j$. The shape of an elbow of $\mathcal{CR}_j$ corresponding to a circular segment of L_j (of radius ρ_j) depends on the relative values of r_j, r_k, and ρ_j. If $\rho_j > r_j + r_k$, then the elbow of $\mathcal{CR}_j$ is bounded by two circular arcs of respective radii $\rho_j + r_j + r_k$ and $\rho_j - r_j - r_k$ (see Figure 6.a). Otherwise, it is bounded by a circular arc of radius $\rho_j + r_j + r_k$ on one side and by a right corner on the other side (Figure 6.b).

We compute a rectangular approximation of $\mathcal{CB}_i$ by first constructing the bounding box of $\mathcal{CB}_i$ whose edges are parallel to $\mathcal{F}_\mathcal{W}$'s axes. We next cut this box into slices parallel to its shortest side at a specified resolution (which determines the precision of the approximation). Assume that these slices are parallel to the x-axis. For each slice S, we compute the projection of the intersection of $\mathcal{CB}_i$ and S on the x-axis, and we lift this projection back into the slice, thus obtaining a rectangloid approximation of $\mathcal{CB}_i$ within S (see Figure 7).

Every grown route $\mathcal{CR}_j$ is treated in a similar fashion. However, we take advantage

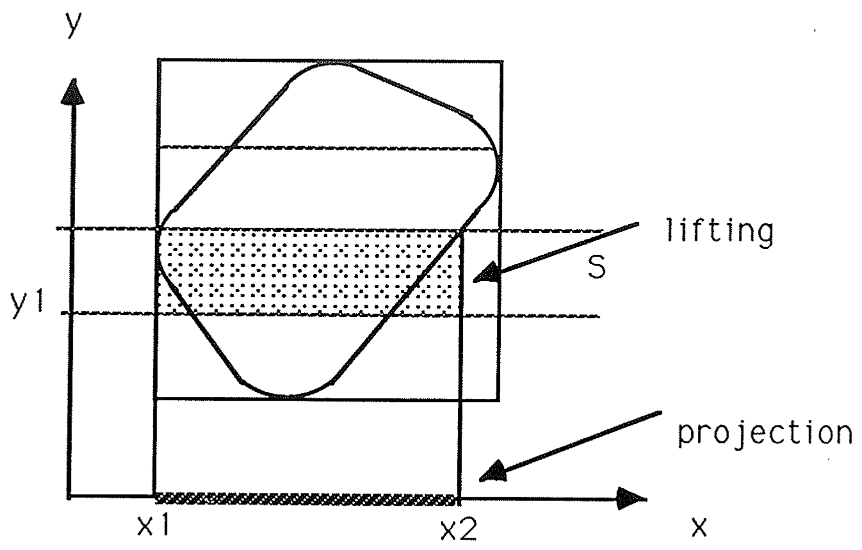


Figure 7: Computation of the Bounding Approximation of a C-obstacle

of the special shape of the routes to simplify this process. In particular, since the straight segments of a route are rectangles oriented along $\mathcal{F}_{\mathcal{W}}$'s axes, they do not have to be approximated.

4.3.2 3D Case

The isotropic growth by r_k of a polyhedral obstacle $\mathcal{B}_i$ yields a generalized polyhedral C-obstacle $\mathcal{CB}_i$, i.e. a volume whose boundary is made of pieces of planar, cylindrical and spherical surfaces. Each planar face corresponds to a face F of $\mathcal{B}_i$ and is obtained by translating F by r_k along its outgoing normal direction. Each cylindrical face corresponds to a convex edge E of $\mathcal{B}_i$ and is part of the surface of a cylinder having E for its midline and r_k for its radius. Each spherical face corresponds to a convex vertex V of $\mathcal{B}_i$ and is part of the sphere of radius r_k centered at V.

In order to compute a bounding approximation of $\mathcal{CB}_i$, we apply a recursive project-and-lift technique [Zhu and Latombe, 1989], which we sketch below. Assume that the longest dimension of the bounding box of $\mathcal{CB}_i$ is along $\mathcal{F}_{\mathcal{W}}$'s z-axis. We slice the z-axis into intervals at some prespecified resolution and we compute the projection $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ of the subset of $\mathcal{CB}_i$ contained in every slice $[z_1, z_2]$ into the xy-plane. Next, we proceed recursively by slicing, say, the y-axis and projecting the subset of $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ contained in every slice $[y_1, y_2]$ into the x-axis. The latter projection consists of one or several intervals. We lift each such interval $[x_1, x_2]$ up to a rectangloid $[x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$ in $\mathcal{C}_k$.

We could compute the projection $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ from the explicit representation of $\mathcal{CB}_i$'s boundary. Although the construction of this representation is not really difficult, it is not needed as shown below (if we assume that $\mathcal{B}_i$ is homeomorphic to a sphere, hence is not traversed by any hole.) For every face F of $\mathcal{B}_i$, we let $\mathcal{CF}$ denote the face F translated by r_k along its outgoing normal. For every convex edge E of $\mathcal{B}_i$, we let $\mathcal{CE}$ denote the truncated cylinder of radius r_k having E as its midline. For every convex vertex V of $\mathcal{B}_i$, we let $\mathcal{CV}$ denote the sphere of radius r_k centered at V. Since both the cylinders $\mathcal{CE}$ and the spheres $\mathcal{CV}$ are entirely contained in $\mathcal{CB}_i$, the projection $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ is obtained by projecting the subset of every element $\mathcal{CF}$, $\mathcal{CE}$, and $\mathcal{CV}$ contained in $[z_1, z_2]$ into the xy-plane, and computing the external boundary of the union of all the projections. These computations are straightforward (though, in the case of a cylinder $\mathcal{CE}$, several different cases have to be carefully considered). If n in the number of vertices of $\mathcal{B}_i$, the overall computation takes $O(n^2 \log n)$ time in the worst case, using a sweep-line technique [Preparata and Shamos, 1985] to trace out the external boundary of $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$. This boundary consists of straight, circular, and elliptical edges.

The isotropic growth of a route R_j by r_k yields a route of radius $r_j + r_k$ following the same path L_j as R_j. The straight segments and the elbows of the grown route are treated separately. Each straight segment is parallel to a plane of $\mathcal{F}_\mathcal{W}$, say the xy-plane. It is sliced into intervals along the z-axis. Each slice projects into the xy-plane as a rectangle which is lifted up into a rectangloid. Each elbow of the grown route is also parallel to a plane, say the xy-plane. Its cross-section by a plane at any constant z has one of the two forms shown in Figure 6. Hence, each elbow can be sliced into intervals along the z-axis, with each slice projecting into the xy-plane as a 2D route. Each projection is approximated as in the 2D case and the resulting rectangular cells are lifted up into 3D rectangloid cells.

4.3.3 Cashing in on Previous Computation

If the current route has the same radius as a previously generated route (a frequent case in practice), we can reuse the approximations of the $\mathcal{CB}_i$'s computed for planning the previous route. If two routes $\mathcal{R}_{k-1}$ and $\mathcal{R}_k$ having the same radius are planned consecutively, we can even save more work. Indeed, most of the cell decomposition of the free space in $\mathcal{C}_{k-1}$ (and consequently, most of the connectivity graph) remains valid in $\mathcal{C}_k$. One can proceed as follows. The decomposition built to represent the free space in $\mathcal{C}_{k-1}$ is copied into $\mathcal{C}_k$ and, since the route $\mathcal{R}_{k-1}$ is an additional obstacle to $\mathcal{R}_k$, the cells of the copied decomposition which are intersected by the route $\mathcal{R}_{k-1}$ grown by $r_k = r_{k-1}$ (and only these cells) are refined into smaller cells. Since this can result in considerable saving of computation, it is generally desirable to plan routes with the same radius consecutively. This is compatible with the general prioritizing heuristics that suggests bigger pipes to be routed before smaller ones.

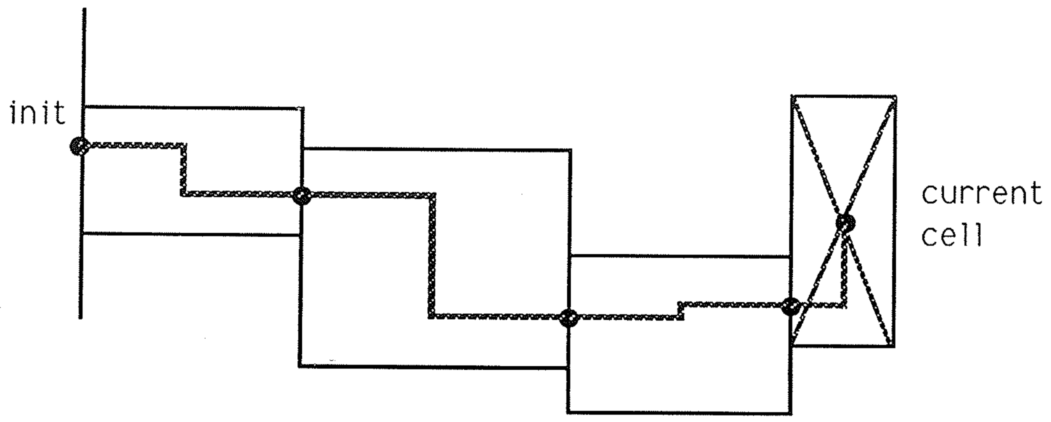


Figure 8: Estimation of the Length of a Path in a Partial Channel

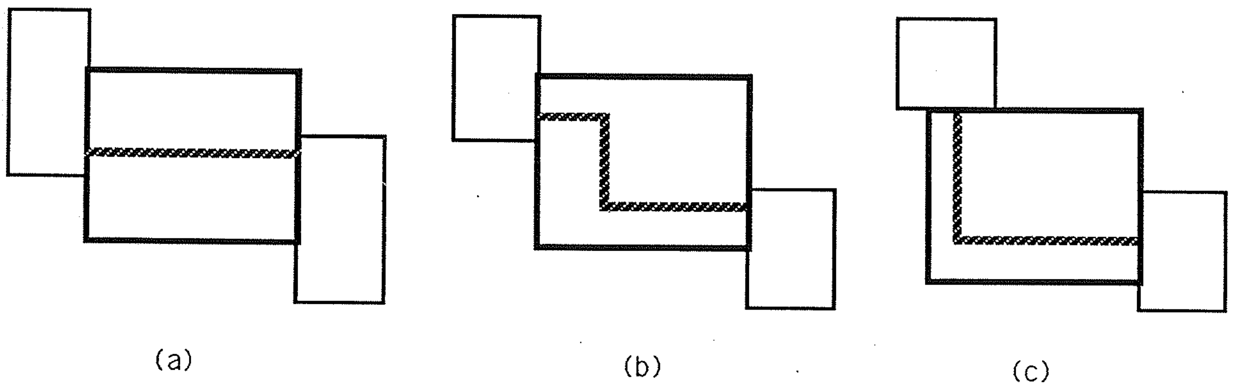


Figure 9: Expected Number of Turns in a Cell

4.4 Channel and Route Generation

Once we have decomposed the free space in C_k, we construct the **connectivity graph** G_k representing the adjacency relation among the generated cells. The node corresponding to the cell whose boundary contains the terminal T_k^1 (resp. T_k^2) is called the **initial** (resp. **goal**) node (the initial and goal nodes coincide if the two terminals are contained in the boundary of the same cell). A channel is constructed by searching G_k for a path connecting the initial node to the goal node.

Various techniques could be used to search G_k. In our implementation, we use an A* algorithm [Nilsson, 1980]. The search is guided by an evaluation function $f(N) = g(N) + h(N)$ defined over the set of nodes in G_k. If there exist one or several channels between the initial and the goal cells, the A* algorithm generates one channel which minimizes the function f evaluated at the goal node. Since it is desirable to plan paths of minimal length and minimal number of turns in order to generate satisfactory layouts, we use the evaluation function f to encode this desire.

Hence, $g(N)$ is defined as a weighted sum of the length $l(N)$ of the path L_k constructed so far and its number of turns $n(N)$. However, since we construct L_k only after a complete channel has been generated, $l(N)$ and $n(N)$ are only estimates derived from the geometry of the channel. We compute $l(N)$ as the Manhattan length of the line connecting the terminal T_k^1 to the center of the last cell generated so far and passing through the midpoint of the intersection of every two successive cells (Figure 8 shows this line in the 2D case). We compute $n(N)$ by associating an expected number of turns in every cell of the partial channel constructed so far, except the last cell. We define the *entrance* of a cell κ as the intersection of its boundary with that of the previous cell (the entrance of the first cell is the point T_k^1) and its *exit* as the intersection of its boundary with that of the following cell (the exit of the last cell is T_k^2). In 2D, the expected number of turns in κ is: 0, if the entrance and the exit are contained in parallel sides of the cell and their projections overlap (Figure 9.a); 2, if the entrance and the exit are in parallel sides and their projections do not overlap (Figure 9.b); and 1, if the entrance and the exit are not in parallel sides (Figure 9.c). In 3D, the expected number of turns in κ is 0, 1, 2, or 3 depending on the relative orientation and position of the entrance and the exit of the cell.

The function $h(N)$ is simply computed as the Manhattan distance between the center of the cell corresponding to the node N and the terminal T_k^2.

Let us assume that the search of G_k succeeds in producing a channel $(\kappa_1, ..., \kappa_s)$ of s adjacent cells. We construct a path L_k within this channel which has minimal length and minimal number of turns over all the paths lying in this channel. This is done by selecting a point Q_i in the exit of every cell κ_i ($i \in [1, s-1]$) along the channel, except the last one. L_k is constructed as the concatenation of s subpaths successively connecting T_k^1 to Q_1 in κ_1, Q_i to Q_{i+1} in κ_{i+1}, for $i = 1, ..., s-1$, and Q_{s-1} to T_k^2 in κ_s. The choice of the points Q_i is done by applying the following constraint propagation technique. Let β_i denote the exit of the cell κ_i and $\beta_0 = T_k^1$. For $i = 1$ to $s-1$, we reset β_i to its intersection with the projection of β_{i-1} into the line or the plane containing β_i, if this intersection is non-empty; otherwise, we leave it unchanged. In a second pass, for $i = s-1$ to 1, we reset β_i to its intersection with the projection of β_{i+1}, if this intersection is non-empty. In choosing Q_i, we let $Q_0 = T_k^1$, and for $i = 1, ..., s-1$, we set Q_i to the projection of Q_{i-1} into β_i if the projection lies within β_i; otherwise we set Q_i to the point in β_i that is closest to the projection of Q_{i-1} subject to the constraints described below concerning the turn radius of the pipe. This selection guarantees to generate a path with minimum length and number of turns within the channel.

The algorithm described above generates a pipe path L_k in the form of a polygonal line connecting two terminals. The intermediate vertices of this line must be changed into circular arcs of radius ρ_k as illustrated in Figure 10. It may, however, happen

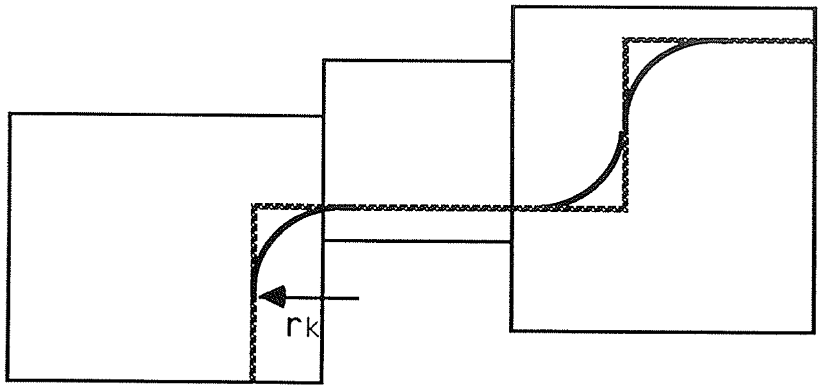


Figure 10: Fitting Circular Arcs into a Path

that this transformation is impossible (for instance, two consecutive vertices of the path are closer to each other than $2\rho_k$), or that a circular arc intersects a C-obstacle. If either of these two events happens, we can either relax the optimality requirement discussed above or resume the search of G_k at the cell of the current channel where the problem was detected. (Remark: This algorithm is not complete. The problem of finding a path with a lower-bounded curvature radius is directly related to the path planning problem for a nonholonomic car-like robot with a limited steering angle [Laumond, 1987] [Fortune and Wilfong, 1988] [Jacobs and Canny, 1989] [Barraquand and Latombe, 1989b] [Latombe, 1990]. This problem is known to be very difficult. Fortunately, in pipe layout design, the radius ρ_k is usually small, and the two events mentioned above happen relatively rarely.)

After a path L_k has been constructed, we update the layout description by adding the newly created route. This route becomes an obstacle to the pipes that remain to be routed.

Remark: In the basic routing problem we assume that every pipe connects two terminals. Though this is the most common situation in practice, there may exist some pipes, called *nets*, which have more than two terminals. The channel and path generation algorithms implemented in our PLD system have been extended to handle the case of these multi-terminal nets. The extension works as follows. The algorithm selects two terminals of the net and constructs a channel connecting these terminals, regardless of the other terminals. It then selects an other terminal among the remaining ones and search for a channel connecting this terminal to any of the cells in the channel connecting the first two terminals. This process is repeated for each of the remaining terminals in the net yielding a multi-branch channel. A path is extracted from this channel by connecting the terminals in the same order as for the construction of the channel. Figure 11 shows a route generated for a four-terminal net.

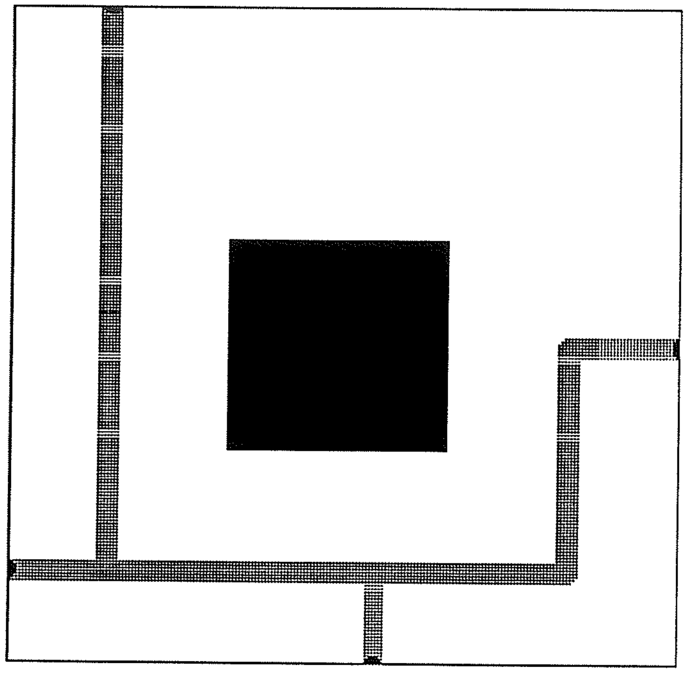


Figure 11: Example of a Multi-Terminal Net

4.5 Backtracking

If the search of G_k fails to produce a channel in the decomposition of the free space in C_k, the algorithm must backtrack. That is, it must change the routes of some of the previously routed pipes to make room for the current pipe $\mathbf{P}_k$. Backtracking requires the Pipe Router to decide *which* routes to change and *how* to change them. One must also avoid running iteratively through the same cycle of failures.

A simple approach consists of applying a *chronological backtracking strategy* by changing routes one by one in the reverse chronological order of their generation. However, this approach is quite inefficient, because it often leads to changing routes that are not relevant to the failure of finding a route for $\mathbf{P}_k$. This drawback is illustrated by the example shown in Figure 12.a. In this example, $\mathbf{P}_1$ and $\mathbf{P}_2$ have been routed in sequence. Then the algorithm fails to find a route for $\mathbf{P}_3$. A chronological backtracking strategy would first change $\mathbf{P}_2$, while it is obvious that the current route of $\mathbf{P}_1$ is the only one that is responsible for the failure.

Inspired by dependency-directed backtracking techniques [Latombe, 1976 and 1979] [Stallman and Susman, 1977], we have developed a more sophisticated backtracking strategy. If the search of G_k fails to find a route for $\mathbf{P}_k$, we augment the connectivity graph G_k by adding those rectangloid cells which are occupied by the bounding approximations of the previously generated routes. (As we will see in a moment, some

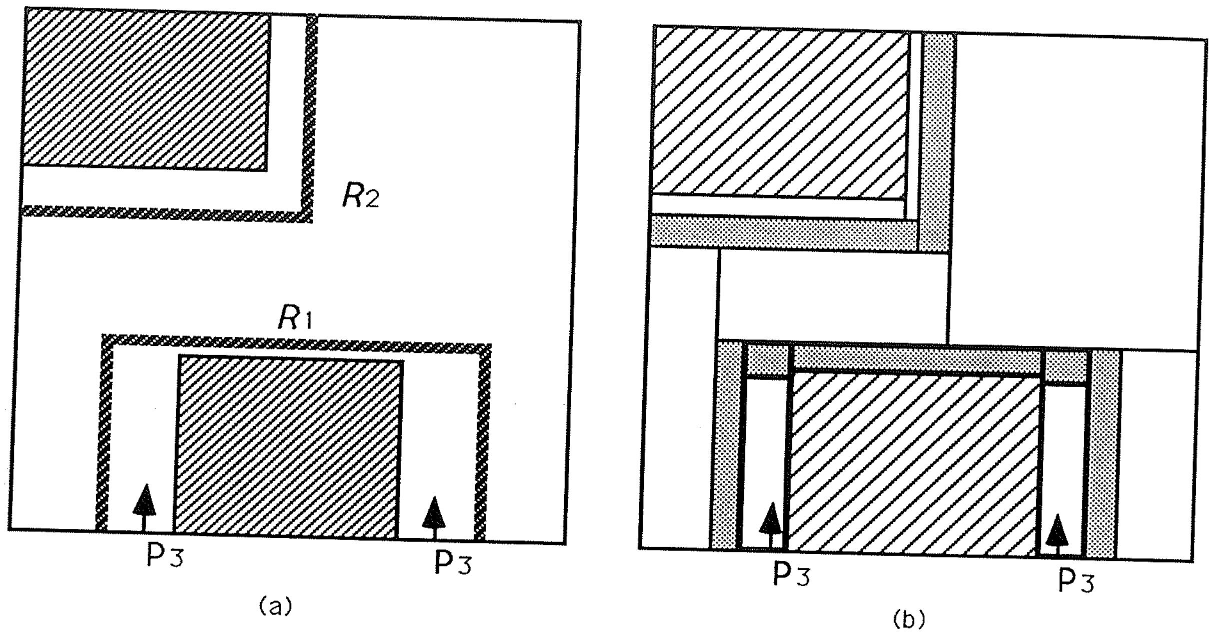


Figure 12: Identification of Relevant Routes to Rip Off

routes may be "protected"; only those cells corresponding to the other, un-protected routes are included in the augmented graph.) We search the augmented connectivity graph G'_k for a channel. The cells of the generated channel that were not in the previous graph allows us to identify the routes that are relevant to the failure. (If no channel can be built in G'_k, it simply means that, at the resolution of the approximation of the free space, there is no route for $\mathbf{P}_k$, regardless of the other routes.) The number of planned routes to be changed in order to make room for $\mathbf{P}_k$ depends on the channel generated by the search of G'_k (assuming that one can be generated). We can reduce this number by searching G'_k with an A* algorithm using an appropriate evaluation function. We then only change those routes that occupy cells in the channel generated by the search of G'_k. Figure 12.b illustrates the application of this backtracking approach. Both the empty cells (shown white) and the cells occupied by the routes $\mathcal{R}_1$ and $\mathcal{R}_2$ (shown grey) are in G'_3. The channel whose cells are shown with bold contours is produced by the search of G'_3. The grey cells included in this channel are traversed by the route $\mathcal{R}_1$ only. The route $\mathcal{R}_2$ has no intersection with the channel and hence need not be changed.

Once the algorithm has decided which routes to change, it still has to decide how to change them. One approach would consist of locally modifying routes to free up all the cells that are included in the channel generated by searching G'_k. This approach could work well if there were available empty space around these routes for them to go. Otherwise the local modification of a route would require other routes to be changed and controlling the propagation of changes could be difficult to manage. Therefore, in

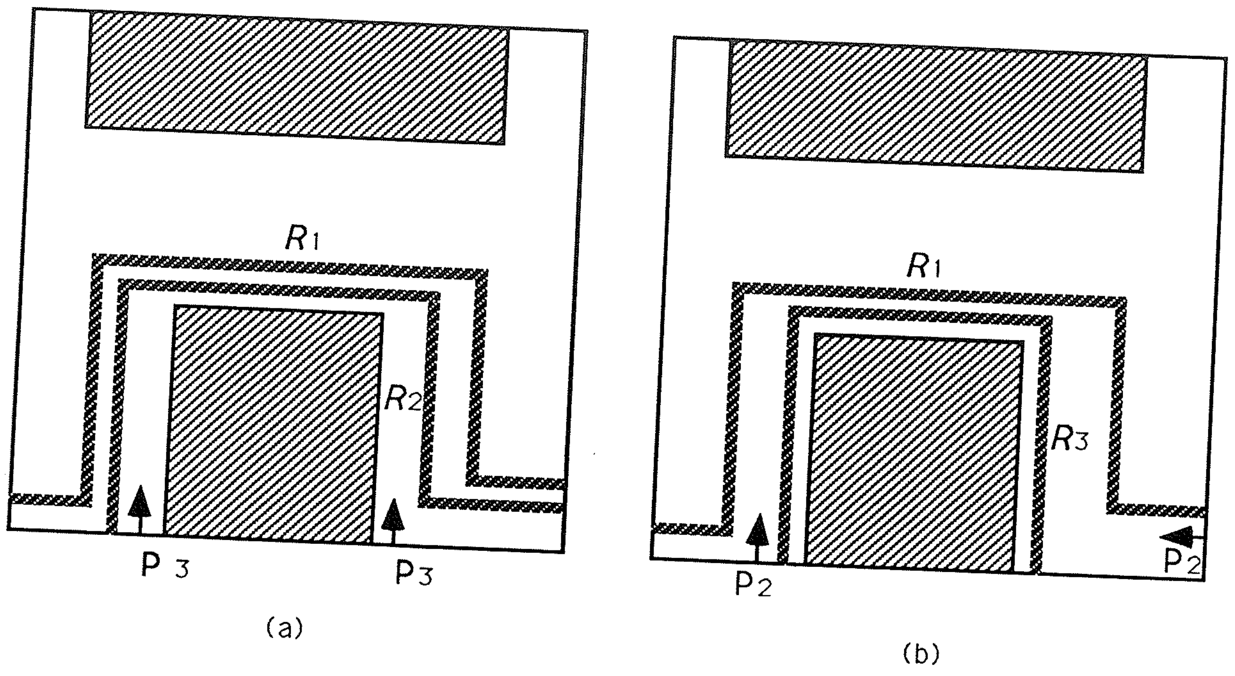


Figure 13: Interaction Among Routes

our PLD system, we adopted a more radical, but incomplete, approach which consists of ripping off all the routes that should be changed and re-routing them after a route has been planned for the current pipe $\mathbf{P}_k$. This essentially corresponds to routing some of the pipes in a different order.

Ripping off routes increases the free space in $\mathcal{C}_k$. Rather than recomputing a new decomposition of the free space, however, we consider the cells forming the bounding approximation of the C-obstacles $\mathcal{CR}_j$ corresponding to the ripped routes and we include them in the cell decomposition of the new free space together with the cells that were previously computed with all the routes $\mathcal{R}_1$ through $\mathcal{R}_{k-1}$ being present. (Actually, this step is slightly more complicated and takes into account the fact that each of these cells may have a non-zero intersection with cells approximating the C-obstacles $\mathcal{CB}_i$ and the C-obstacles $\mathcal{CR}_j$ corresponding to non-ripped pipes.) To reduce free space fragmentation into too many cells (and hence ultimately reduce the cost of the search), we apply simple heuristics to merge adjacent cells together into larger ones. The new decomposition yields a new graph G_k which contains the channel found for $\mathbf{P}_k$ in G'_k.

In order to make the approach work properly, however, several issues still have to be addressed. First, after we generated $\mathcal{R}_k$, we need to re-plan the ripped routes. Re-planning these routes may require other routes to be retracted as illustrated in Figure 13. In this example, $\mathbf{P}_1$ and $\mathbf{P}_2$ have been routed. Assume that $\mathcal{R}_2$ is then ripped off in order to make room for the pipe $\mathbf{P}_3$ and that the route $\mathcal{R}_3$ is generated as shown in Figure 13.b. Planning a new route for $\mathbf{P}_2$ then causes either $\mathcal{R}_1$ or $\mathcal{R}_3$

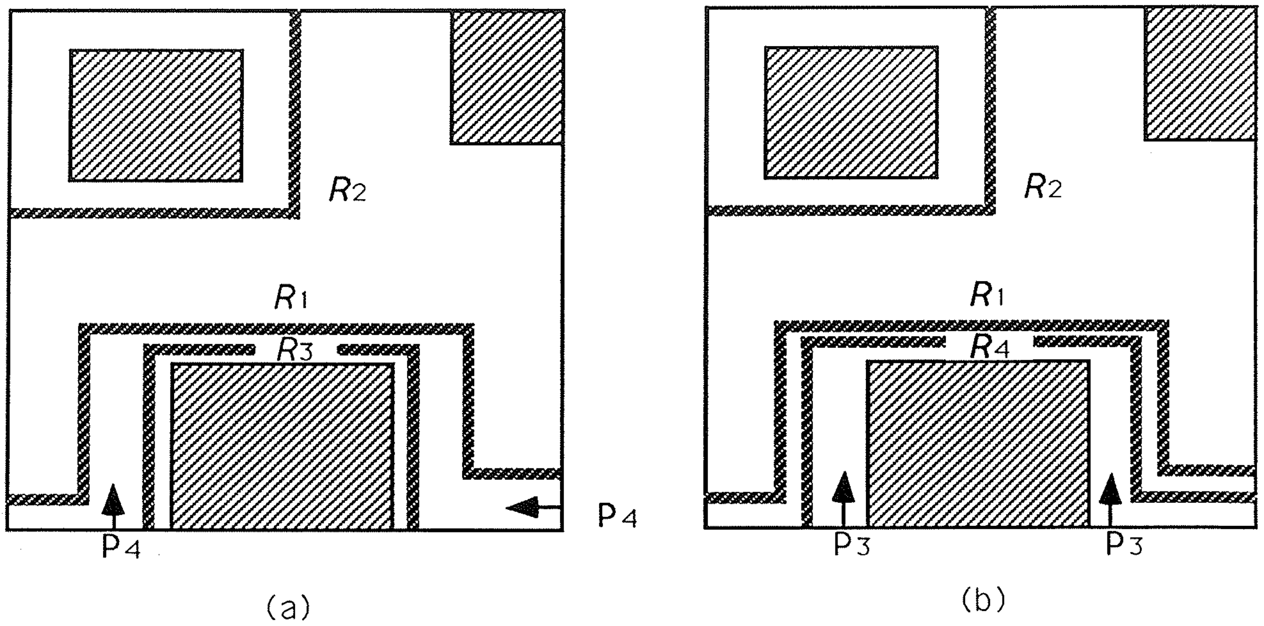


Figure 14: A More difficult Example

to be removed, which may cause infinite looping if $\mathcal{R}_3$ is chosen rather than $\mathcal{R}_1$. To avoid infinite looping, we make use of the **route protection** mechanism described below.

When the route of a pipe $\mathbf{P}_j$ is ripped off to make space for routing $\mathbf{P}_k$, $\mathbf{P}_j$ remembers $\mathbf{P}_k$ in a *protection list* PL_j. When a new route $\mathcal{R}_j$ is generated for $\mathbf{P}_j$, the current routes of the pipes in PL_j are *protected* against $\mathbf{P}_j$. This means that if the generation of $\mathcal{R}_j$ fails, they should not be ripped off to make room for $\mathbf{P}_j$. If the algorithm ultimately succeeds to re-route all the pipes $\mathbf{P}_j$ ($j < k$) whose routes were ripped off, $\mathbf{P}_k$ is removed from the corresponding lists PL_j. To illustrate how protection works, consider the previous example (Figure 13). When we remove $\mathcal{R}_2$ to route $\mathbf{P}_3$, we store $\mathbf{P}_3$ in PL_2. When we re-route $\mathbf{P}_2$, we must remove either $\mathcal{R}_1$ or $\mathcal{R}_3$. Since $\mathcal{R}_3$ is protected, we decide to rip $\mathcal{R}_1$ off. This allows us to plan a new route $\mathcal{R}_2$, and next a new route $\mathcal{R}_1$. $\mathbf{P}_3$ is then removed from PL_2.

However, the channel found by searching G'_k may not be unique. If this is the case, different subsets of routes could have been ripped off in order to make room for $\mathbf{P}_k$, and perhaps the algorithm chose the wrong subset. This possibility requires the above protection mechanism to be completed. As an illustration, consider Figure 14. Suppose that having failed to plan a route for $\mathbf{P}_4$ (Figure 14.a), the algorithm decides to rip $\mathcal{R}_3$ off (hence, $\mathbf{P}_4$ is stored in PL_3). After $\mathbf{P}_4$ has been routed as shown in Figure 14.b, the algorithm tries to re-route $\mathbf{P}_3$ and it fails. It then identifies the route

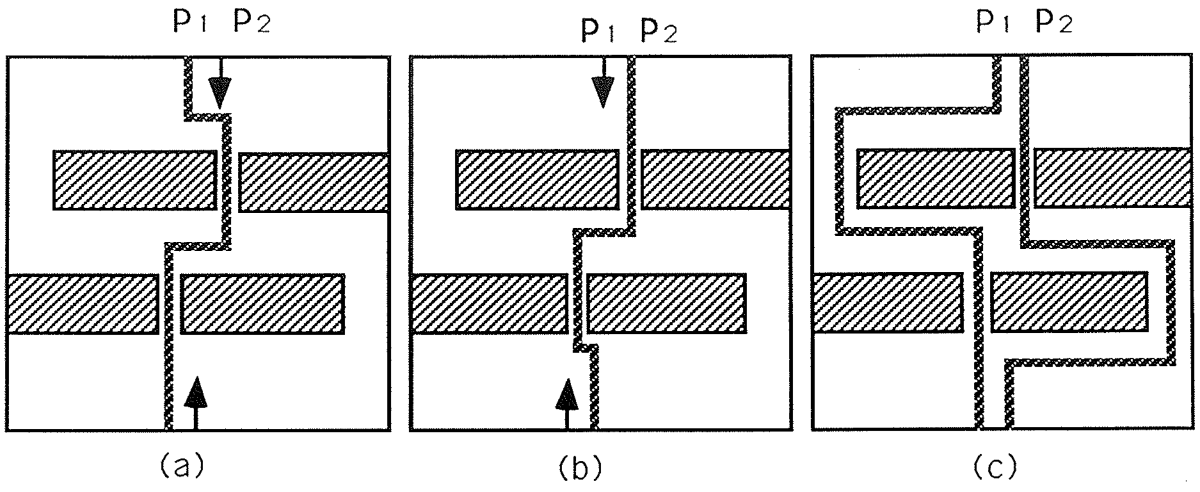


Figure 15: An Example Where the Protection Mechanism Fails

of $\mathbf{P}_4$ as the only responsible for this failure. Since $\mathbf{P}_4$ is protected, its only remaining alternative is to withdraw the previous decision to rip $\mathcal{R}_3$ off for routing $\mathbf{P}_4$. That is, it re-establishes the situation shown in Figure 14.a and attempts to route $\mathbf{P}_4$. But, this time, it rips $\mathcal{R}_1$ off, rather than $\mathcal{R}_3$.

The above bactracking algorithm allows the Pipe Router to eventually consider all the possible orderings on the pipes, without looping. It has been implemented in our PLD system and works well for most reasonable routing problems. However, it is not complete, and there are two major causes to that. First, the above protection mechanism is too strict, as illustrated in the example of Figure 15. Routing $\mathbf{P}_2$, after $\mathbf{P}_1$ has been routed as shown in Figure 15.a, causes $\mathcal{R}_1$ to be ripped off and therefore $\mathbf{P}_2$ to be remembered in PL_1. When the algorithm attempts to re-route $\mathbf{P}_1$ after it has routed $\mathbf{P}_2$ as shown in Figure 15.b, it fails. Recovering from the failure would require to rip the current route of $\mathbf{P}_2$ off, but it is protected. On the other hand, the algorithm cannot withdraw its anterior decision to rip the route of $\mathbf{P}_1$ off since there was no other possible responsible for the failure to route $\mathbf{P}_2$. Nevertheless, there exists a solution for the routing problem shown in Figure 15.c. The basic reason why our backtracking algorithm fails in this example is that the protection mechanism identifies pipes as being responsible for the failures, while pipes routed in a certain way are the actual causes for the failure. In Figure 15.a, the failure to find a route for $\mathbf{P}_2$ is caused by $\mathbf{P}_1$ with its current route. In order to be more complete, the backtracking algorithm should keep open the possibility of changing the route of $\mathbf{P}_1$ in order to find a route for $\mathbf{P}_2$. Modifying the algorithm accordingly would not be very difficult, but it would result in a much larger solution space to explore, and therefore a much less efficient PLD system. In fact, the backtracking algorithm implemented in our system is even more radical. In the example of Figure 14, after it has re-established the situation shown in Figure 14.a and before it attempts to route $\mathbf{P}_4$ again, it stores $\mathbf{P}_3$ in a *permanent* protection list attached to $\mathbf{P}_4$. This list prevents the routing of $\mathbf{P}_4$ to ever cause the removal of the route of $\mathbf{P}_3$ in the rest of the routing process. Permanent protection slightly worsens the incompleteness of the Pipe Router, but it definitely increases its efficiency.

The other cause of incompleteness is that the algorithm does not consider the possibility of modifying the path of a route *within* a channel. This second cause will be handled by a new algorithm, called *parallel routing*, that we are currently developing (see Section 6).

The backtracking algorithm described above is also applicable to 3D workspaces. The main difference in 3D is that mis-routing a pipe is more likely to cause unnecessary turns in some of the routes planned later rather than search failures. For example, in the example of Figure 12.a, if the workspace was three-dimensional, $\mathbf{P}_3$ could be routed, but it would have at least two extra turns. In our PLD system, we use a heuristic function to activate the backtracking algorithm when it detects unnecessary turns or recognizes pipes with unrealistic numbers of turns. We expect that the parallel routing algorithm under development will considerably reduce the number of backtracking operations in the 3D case.

4.6 Implementation and Experimentation

This basic Pipe Router described above has been implemented in Allegro Common Lisp running on a Macintosh II for 2D and 3D workspace. In our current implementation, a 3D workspace is discretized into multiple 2D layers. Figure 16 shows an example where two pipes are routed one after the other in a 3D workspace with two layers. The window "Workspace" shows the projection of the pipe routes into the xy-plane of the workspace. The first pipe lies entirely in the first layer; the second pipe lies in part in the first layer, and in part in the second layer (darkest portion). The windows "Layer 1" and "Layer 2" show the first and the second layers of the corresponding configuration space. Figure 17 gives another example with more pipes.

5 Extensions

If we look to Figure 1 again, we see that the basic Pipe Router can generate pipe layouts that satisfy input Functional Constraints. However, as mentioned in Section 2, a pipe layout must also satisfy a variety of other constraints input as Expert Design Constraints. In this section we describe how these constraints are processed in our PLD system. The constraint classifications and constraint treatments discussed in

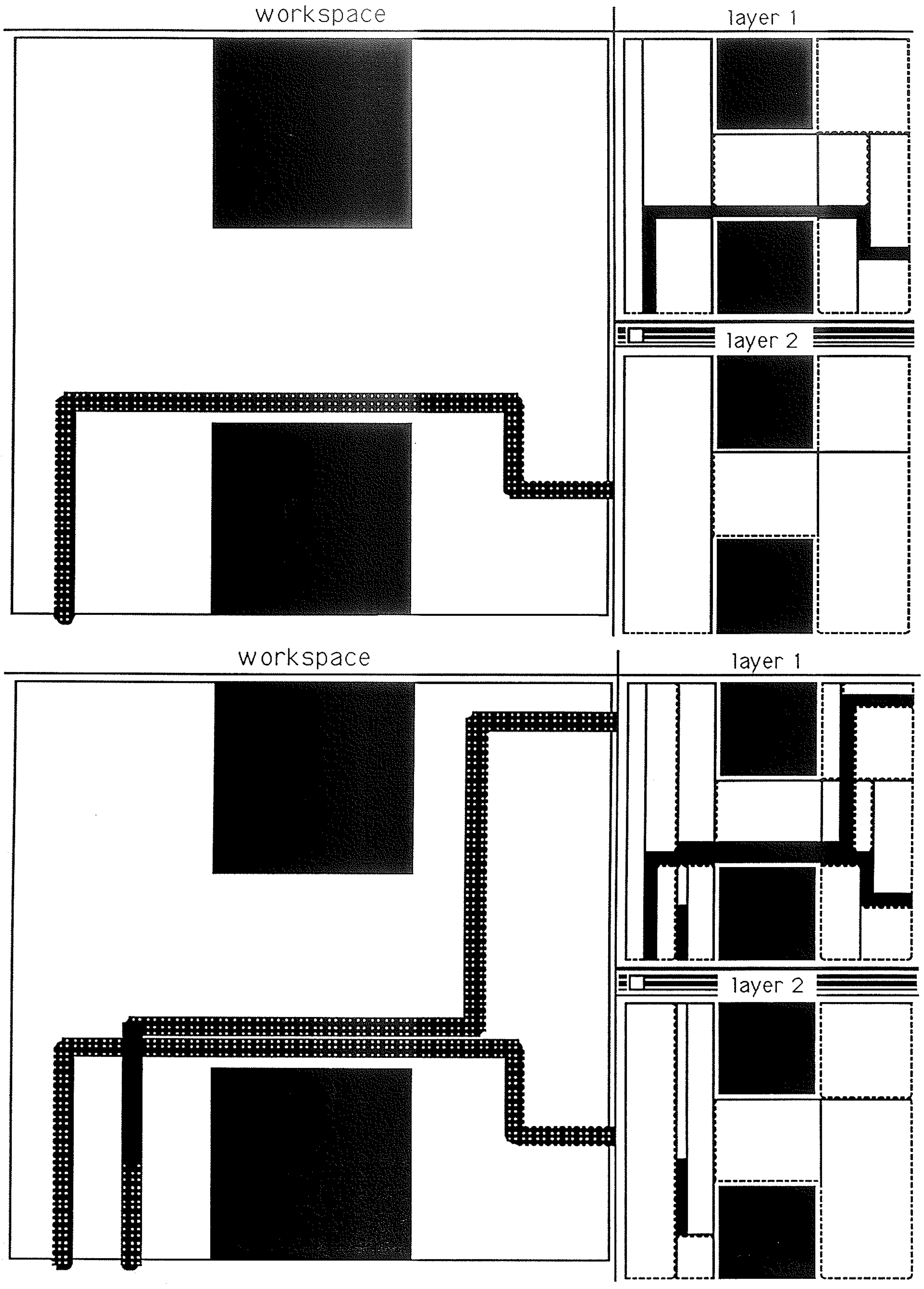


Figure 16: Output of the Basic Pipe Router

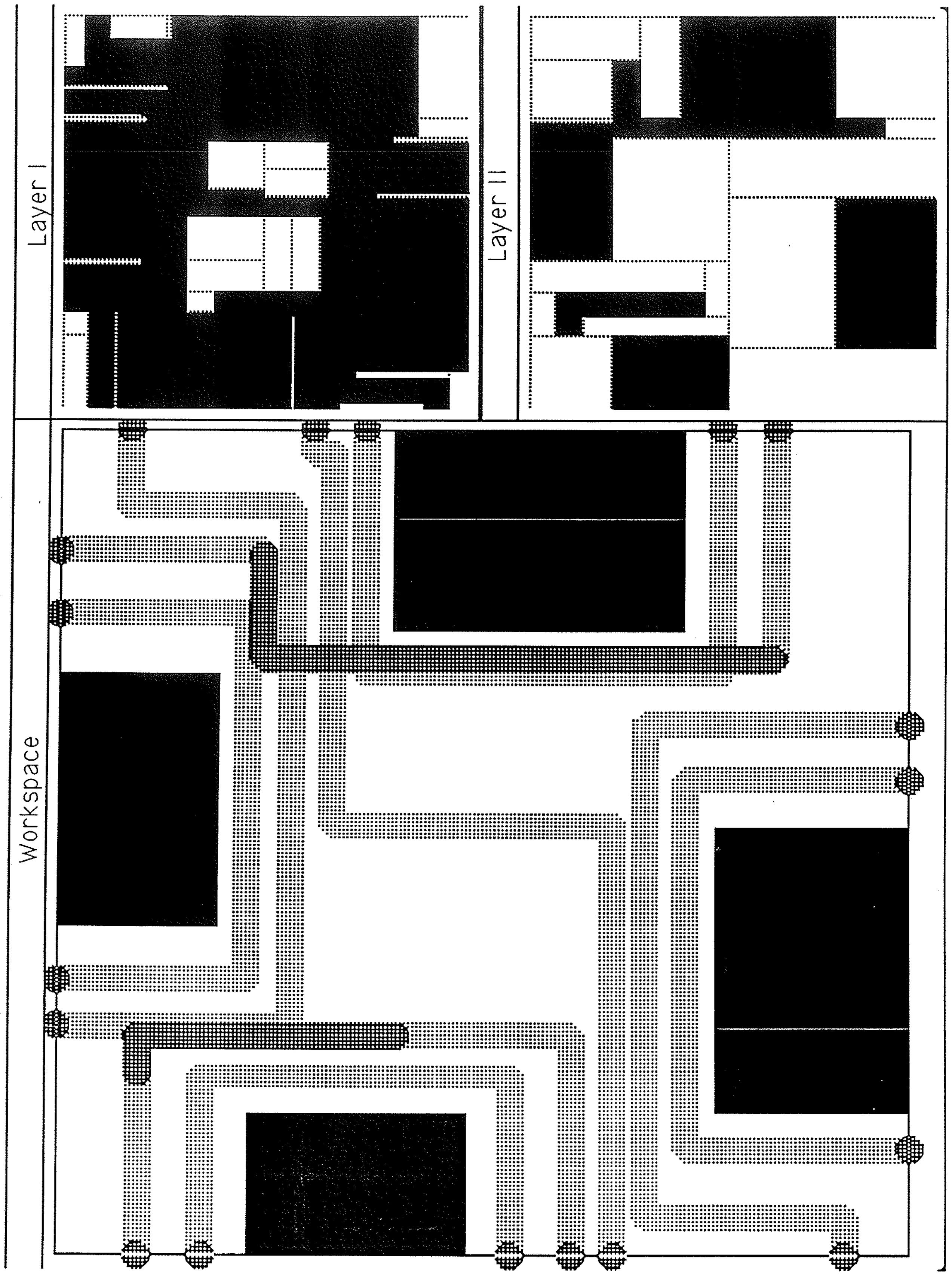


Figure 17: Pipe Layout Produced by the Basic Pipe Router

this section have been inspired by [Mitsuta et al., 1986].

5.1 Expert Design Constraints

Expert Design Constraints are abstractions of more involved constraints whose accurate verification requires running complex programs. We have identified three types of such constraints, which we call **process constraints**, **structural constraints**, and **accessibility constraints.**

Process constraints relate to the process that is carried out in the pipes. Examples of such constraints are the following:

- A high-temperature pipe should have an expansion loop to insure thermal flexibility.
- A pipe carrying water should not be hanged above any electrical equipment.
- Two pipes carrying chemicals α and β, respectively, should be distant of each other by more than d feet.
- A drainage pipe should be non-ascending.
- A heat-sensitive pipe should be kept sufficiently away from high-temperature equipment.

Structural constraints relate to the mechanical properties of the pipes, more precisely their capacity to stand without falling down. Examples of such constraints are the following:

- No large portion of any pipe should be far away from a major support structure such as a wall or a beam.
- A pipe should go as much as possible through existing pipe racks.
- A pipe should not have a vertical drop of more than d_{max} feet to avoid being overstressed.

Accessibility constraints relate to the constructability of the pipe layout, and its ease of operation and maintenance. Examples of such constraints are the following:

- There must be access paths for removing all major equipments for off-site repair.
- If a frequently used valve is to be installed along a pipe, there must be at least one point along the pipe that is less than 3 feet above the ground or a designated platform so that the valve can be installed at that point.

- Free paths should remain to access pipes subject to frequent maintenance, both for humans and machines.

Expert Design Constraints, even in the more codified and more quantitative form required by our system, cannot be used directly by the Pipe Router. They must first be reformulated into geometric constraints. So far, we have found that Expert Design Constraints translate into only two kinds of geometric constraints:

- *Location constraints*, which specify preferred location, undesirable location, and forbidden location for a pipe. They are derived from Expert Design Constraints such as "a heat-sensitive pipe should be kept sufficiently away from high-temperature equipment" and "a pipe should go as much as possible through existing pipe racks."
- *Shape constraints*, which apply to the shape of the pipe routes. They are derived from Expert Design Constraints such as "a pipe should not have a vertical drop of more than d_{max} feet to avoid being over-stressed" and "a drainage pipe should be non-ascending."

The translation is carried out by the Constraint Reformulator. Basically, we have identified a list of schemas for Expert Design Constraints, and the Constraint Reformulator consists of one translation program for each schema.

However, many of the generated location and shape constraints still cannot be processed by the basic Pipe Router as described in the previous section. In the following two subsections we describe extensions of the basic Pipe Router which have been implemented for treating these constraints.

5.2 Location Constraints

The location constraints specify the preferable, undesirable, or forbidden regions for a pipe route to go through. They are conceptualized as *virtual obstacles* which repulse pipes and *virtual sinks* which attract pipes. These virtual obstacles and virtual sinks may be directional, i.e. they may only repulse (or attract) pipes running along a certain direction.

A virtual obstacle can be *hard* or *soft.* Forbidden regions are protected by hard virtual obstacles, which then act as real obstacles, while undesirable regions are protected by soft virtual obstacles which can be traversed by pipes, but at some additional cost. Pipes and virtual obstacles can be given various types such that a pipe of a certain type only "sees" virtual obstacles of certain types. For example, the Constraint Reformulator creates a virtual obstacle of type "high-temperature" surrounding every high-temperature equipment (e.g. a furnace). This virtual obstacle is only visible to pipes classified as "heat-sensitive". This means that the Pipe Router will treat the "high-temperature" virtual obstacle as a regular obstacle when routing a heat-sensitive pipe, and will ignore this virtual obstacle when routing another pipe. This notion of virtual obstacle was previously suggested in [Mitsuta et al., 1986].

A non-directional hard virtual obstacle that is visible to a pipe $\mathbf{P}_k$ is treated as a regular obstacle when the free space of $\mathcal{C}_k$ is decomposed into cells. Hence, no cells enclosing the corresponding C-obstacle are included in the connectivity graph G_k. If the hard virtual obstacle is directional, then the cells corresponding to the C-obstacle are included in the connectivity graph G_k, but the links corresponding to the non-traversable directions are removed from the graph.

On the other hand, the existence of soft virtual obstacles visible to $\mathbf{P}_k$ is taken into account by modifying the decomposition of the free space and the evaluation function $f(N) = g(N) + h(N)$ used by the A* algorithm searching G_k. The C-obstacles corresponding to these virtual obstacles are computed as described in Subsection 4.3 and they are also approximated by rectangloids. The free space in $\mathcal{C}_k$ is decomposed into cells of two types, those which do not contain soft C-obstacles and those which do contain soft C-obstacles. A penalty is associated with each cell containing a soft C-obstacle (the penalty may depend on the type of the corresponding virtual obstacle). This penality is included in the computation of the function $g(N)$, e.g. we may multiply the basic cost (estimated length and number of turns) by the penalty factors (> 1) associated with the traversed cells. If the virtual obstacle is directional, then this penalty is included only if the cell is traversed in the undesirable directions.

The decomposition of the free space in $\mathcal{C}_k$ treats virtual sinks much in the same way as soft virtual obstacle, but with a bonus associated to the corresponding cells. If the virtual sink is directional, the bonus applies only if these cells are traversed in the preferred directions.

Figure 18 illustrates the routing of a heat-sensitive pipe around a furnace. The striped area around the dark obstacle representing the furnace depicts a (non-directional) hard virtual obstacle. The heat-sensitive pipe avoids traversing this area, while a normal pipe goes through it.

Figure 19 shows an example where the workspace contains a pipe rack modeled as a directional virtual sink (along the vertical direction as shown in the figure) and a directional hard virtual obstacle (along the horizontal direction). One of the two routes does not traverse this sink since this would have caused a considerable increase of its length. The other route traverses the sink along the preferred direction.

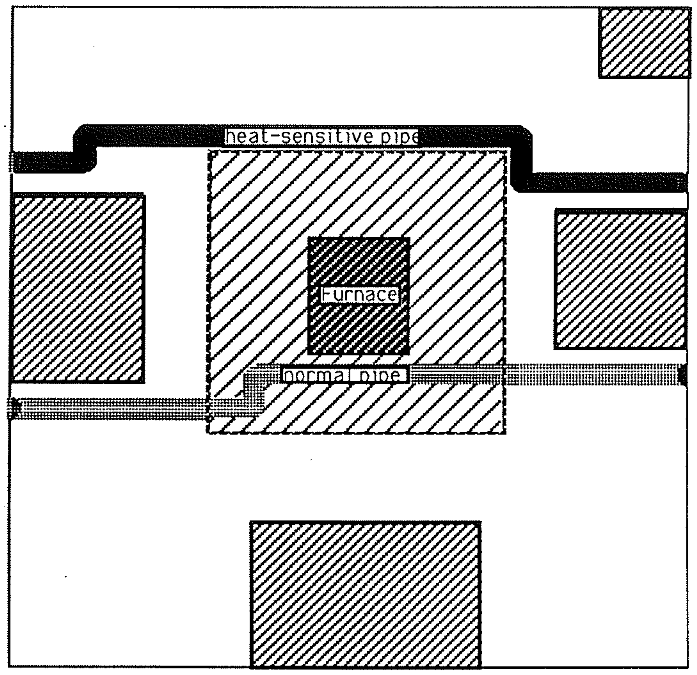


Figure 18: Routing a Heat-Sensitive Pipe Around a Furnace

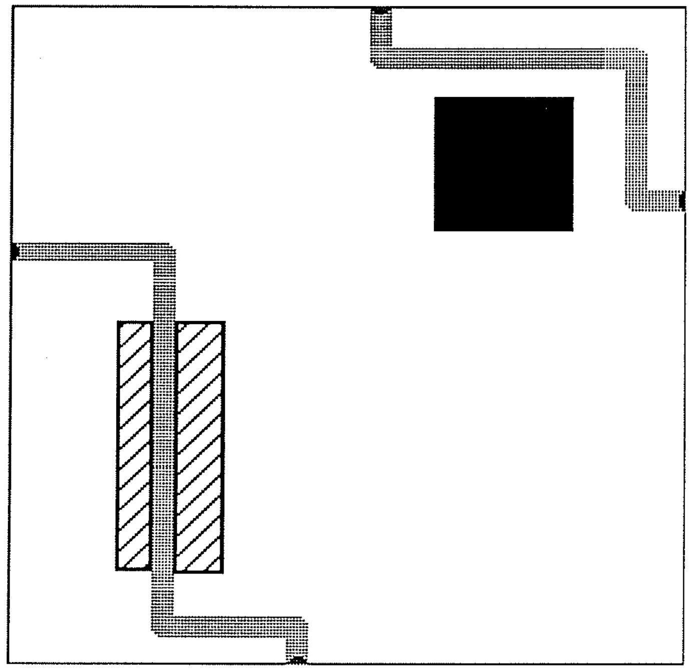


Figure 19: Example of a Pipe Routed Through a Pipe Rack

5.3 Shape Constraints

The other type of geometric constraints applies to the shapes of the pipes. The Pipe Router deals with these constraints at two levels: at the channel generation level and at the path generation level. At the channel generation level, it restricts the search of G_k so that the generated channel, if any, contains at least one path that satisfies the shape constraints. In order to produce a channel containing paths satisfying the shape constraints, the A* search algorithm of the Pipe Router has been modified so that the cost of including a cell in a channel depends on the channel generated so far. We call the search now carried out by the algorithm *history-dependent search.*

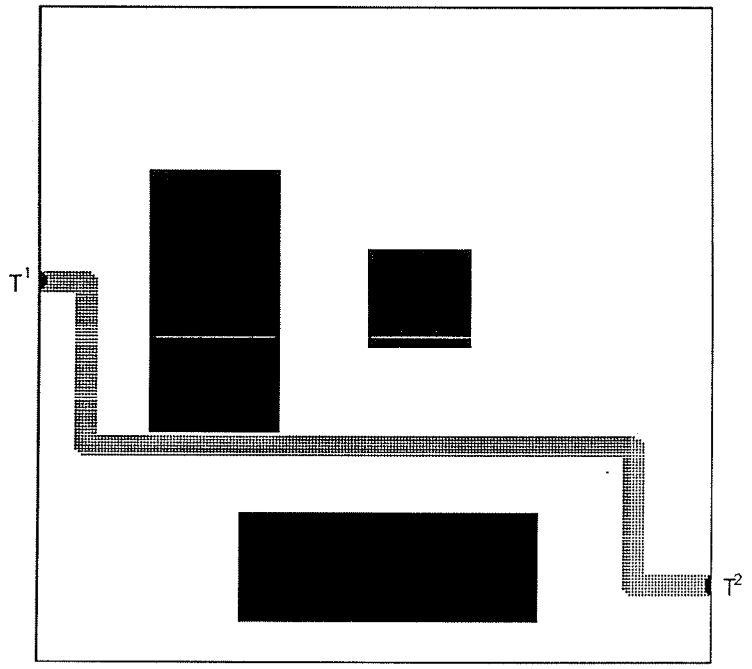


Figure 20: Drainage Pipe

Let us first illustrate history-dependent search with an example. The goal is to route a drainage pipe connecting terminal T^1 to terminal T^2 as shown in Figure 20. In order to simplify the presentation, we assume that the workspace shown in the figure is a vertical 2D workspace. The route of the pipe must be non-ascending between the initial and the goal terminals. The following history-dependent search technique allows the Pipe Router to find a channel that contains at least one route path that is non-ascending.

1) The history information stored in the last cell of a partial channel is the maximum elevation h_{max} of a path at the exit of that cell. In the initial cell, h_{max} is set to the elevation of the initial terminal T^1. In any subsequent cell κ, it is set to the h_{max} of the

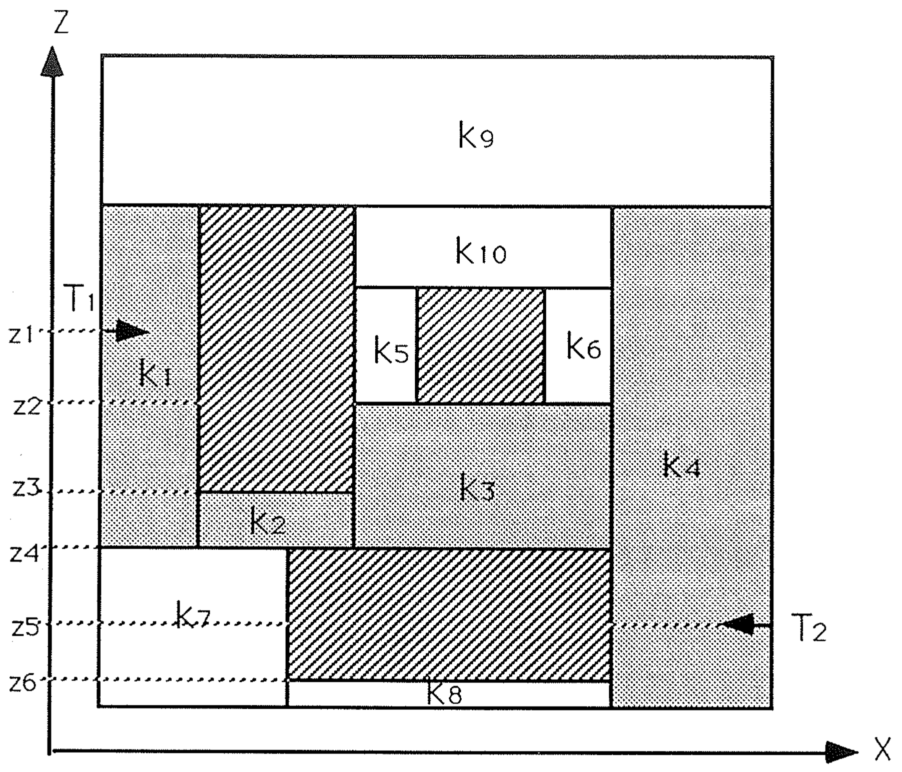


Figure 21: Decomposition of the Configuration Space for the Drainage Pipe

previous cell of the channel, or to the maximum elevation of the points in κ, whichever is smaller. For example, consider Figure 21, which shows the decomposition of the free space in the configuration space of the drainage pipe. The history information h_{max} is set to z_1 in the initial cell κ_1, to z_3 in the next cell κ_2, as well as in the following two cells κ_3 and κ_4 (assuming that the channel $(\kappa_1, \kappa_2, \kappa_3, \kappa_4)$ is being built).

2) The history information h_{max} is used as follows in the cost function $g(N)$. The extra-cost derived from the values of h_{max} is 0 if the minimum elevation of the entrance of the cell corresponding to N is not greater than the value of h_{max} at the previous node, and $+\infty$ otherwise. Thanks to h_{max}, in Figure 21, the only way to expand the partial channel $(\kappa_1, \kappa_2, \kappa_3)$ is by adding κ_4, since the extra-cost of adding κ_5 or κ_6 would be infinite.

3) The history information is also used to estimate the remaining difficulty of completing a partial channel into a channel connecting the initial terminal T^1 to the goal one T^2, and incorporate this estimate in the heuristic function $h(N)$. For example, we may estimate the difficulty to 0 if the value of h_{max} at the current node N is greater than or equal to the elevation of T^2, and to $+\infty$ otherwise. Then the difficulty of finding a channel traversing κ_8 is ∞. No channel including this cell would be considered for expansion. This difficulty estimate is admissible, i.e. it guarantees the optimality of the channel found by the A* algorithm. We may also use a less conservative estimate to improve search efficiency. For example, if the difference Δh between h_{max} at

the current node and the elevation of T^2 is positive, we could estimate the difficulty of expanding the partial channel to the ratio of the horizontal distance from the current cell to T^2 by Δh, rather than 0. This non-admissible heuristic would allow us to anticipate routing difficulties earlier.

Note that this history-dependent search technique is exactly the one used by some motion planners to generate a collision-free trajectory for a robot among moving obstacles. Indeed, such a trajectory is planned as a path in the configuration $\times$ time space of the robot [Erdmann and Lozano-Pérez, 1986] [Latombe, 1990]. Just as the route of the drainage pipe must be non-ascending, this path must not go backward along the time axis.

In history-dependent search, we propagate non-local information along the partial channels while they are constructed. This information may be used both in the cost function $g(N)$ and in the heuristic function $h(N)$ to guide the search. However, it is not always easy to determine which information needs to be propagated and how to compute it efficiently. In that respect, a difficult shape constraint is one which requires the route of a pipe to include a certain number of expansion loops. One way to deal with such a constraint is to generate a channel regardless of the constraint, then to extract an appropriate path from the channel, and if the second step fails (which would be most likely) then generate another channel. However, even for such a difficult constraint, there exists pertinent history information to propagate through the partial channels. For this particular constraint, this information is the number of turns n having appropriate dimensions — i.e. the number of expansion loops — that are possible in the channel generated so far. The information n can then be used to estimate the difficulty of completing a partial channel as the ratio of the number of turns that remain to be made by the distance between the current cell and the goal terminal.

5.4 Implementation and Experimentation

The extensions described above (and others not described here) have been implemented in our Pipe Router. Figure 22 shows how several different constraints can work together in a more complex environment. There are four obstacles $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$ and a pipe rack. $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_4$ are regular obstacles. $\mathcal{B}_3$ is a hot obstacle (e.g. a furnace). Pipes $\mathbf{P}_1$ and $\mathbf{P}_2$ are heat-sensitive and are therefore routed away from $\mathcal{B}_3$. Pipes $\mathbf{P}_3$, $\mathbf{P}_4$ and $\mathbf{P}_5$ are drainage pipes (to make the illustration more easily understandable, we assume the draining direction to be along the y rather than the z-axis. In fact the system allows us to specify any of the three major axes as the draining direction.)

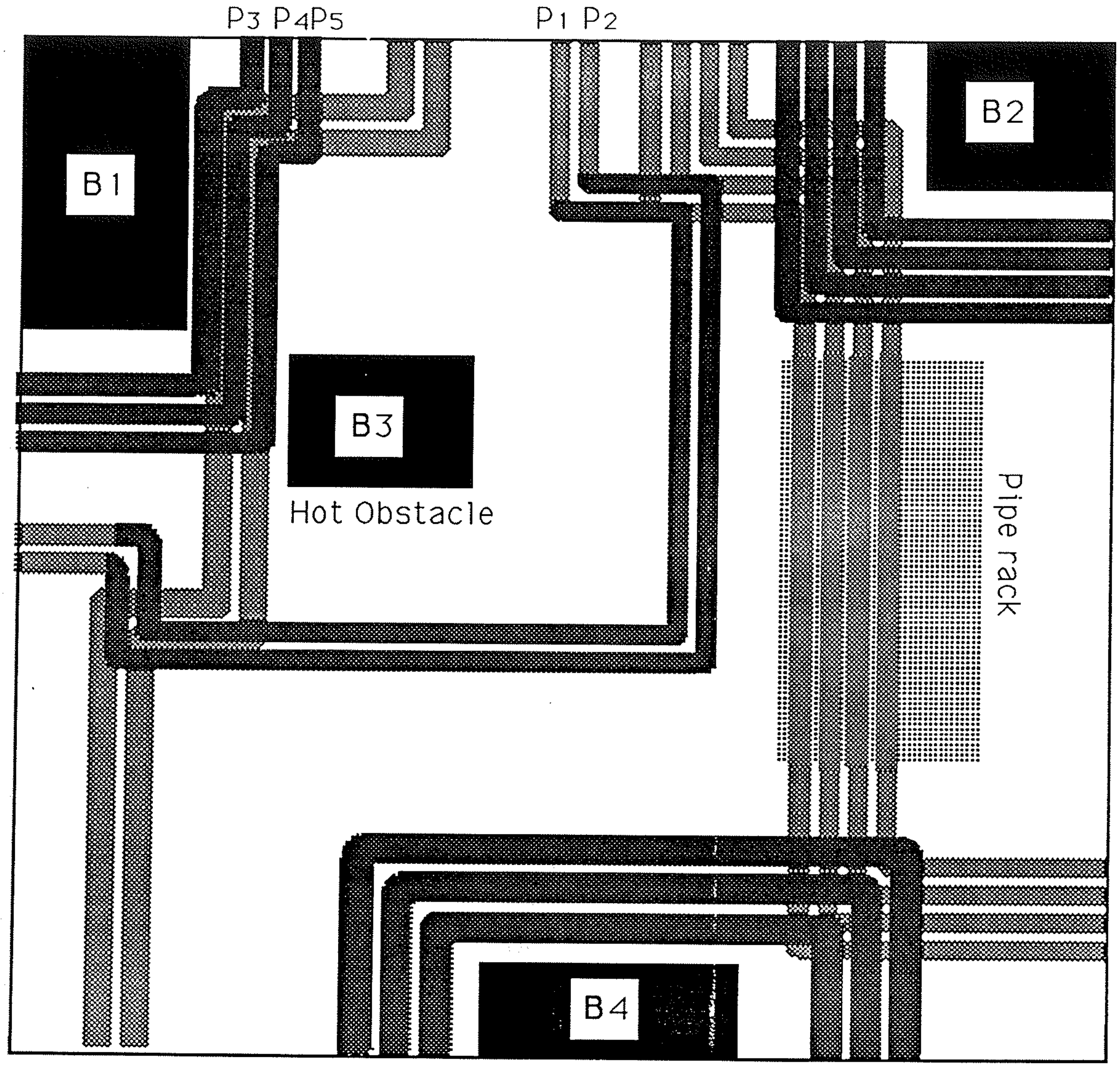


Figure 22: A more complex example

6 Current Work

In this paper we have described a novel approach to pipe layout design in which pipes are treated as the paths left behind by robots moving in a 2D or 3D workspace. We have described a basic Pipe Router based on a well-known motion planning approach, approximate cell decomposition. We have extended this router to handle various process, structural, and accessibility constraints. The extended router described in this paper has been completely implemented in a PLD system. Experimentation has shown that this system can handle intricate constraints and generate satisfactory pipe layouts.

So far, we have paid relatively little attention to the efficiency of the system, though the implementation is reasonably fast. Better efficiency could obviously be achieved by using a faster computer, but we doubt that it would be sufficient to solve problems involving several hundred pipes as is the case in a chemical plant or a submarine. We are now attacking the efficiency problem with two different and complementary approaches: *hierarchical design* and *parallel routing.*

Regarding hierarchical design, our goal is to develop methods for decomposing the pipe workspace into *modules* and for identifying collections of pipes that can be grouped into *macro-pipes.* Workspace decomposition into modules is based on the observation that most complex pipe systems consist of several subsets (the modules) whose layouts are relatively independent of each other. Pipe grouping is based on the observation that in many pipe systems there are several pipes starting at terminals located close to each other and ending at terminals located close to each other. We can take advantage of this situation by treating these pipes as one "macro-pipe". The notions of modules and macro-pipes are actually used by human experts.

Regarding parallel routing, our goal is to reduce the time spent in backtracking operations when pipes strongly interact in their workspace. The inherent drawback of the routing method described in this paper is that it commits every pipe to a specific path within a channel too early, without anticipating the interaction of this pipe with the remaining unrouted pipes. Such an early commitment may yield poor choices preventing the routing of some remaining pipes and causing the backtracking algorithm to be activated. The more crowded the space, the more likely such poor choices. The best way to completely avoid backtracking would be to consider all the pipes simultaneously in the Cartesian product of their configuration spaces, using the centralized planning approach developed in multi-robot motion planning (see Section 3). But this approach would lead us to work in a very high-dimensional configuration space and we know that planning algorithms are exponential in the dimension of the configuration space. These considerations led us to start developing a lesser-commitment approach to pipe routing which proceeds mostly as described in this paper, but delay the construction of a path in a channel until sufficient information is available. In this approach under development, we allow multiple paths to traverse the same cells and we use local planning techniques for "coordinating" these paths.

We realize that the treatment of some of the constraints, especially the shape constraints, is still rather ad hoc. But after we have established the relationship between pipe routing and motion planning, we expect to improve our routing system by further exploit the research done in motion planning, especially in the area of motion planning with constraints (e.g nonholonomic and dynamic constraints).

References

[1] Barraquand, J. and Latombe, J.C., (1989a) *Robot Motion Planning: A Distributed Representation Approach,* Report No. STAN-CS-89-1257, Department of Computer Science, Stanford University. (To appear in *International Journal of Robotics Research.*)

[2] Barraquand, J. and Latombe, J.C., (1989b) "On Non-Holonomic Mobile Robots and Optimal Maneuvering," *Revue d'Intelligence Artificielle,* 3(2), Hermes, Paris, 77-103.

[3] Bechtel (1986) "Expanding CAD Applications on Petroleum Projects," *Bechtel CAE Bulletin.*

[4] Brooks, R.A., and Lozano-Pérez, T. (1982) *A Subdivision Algorithm in Configuration Space for Findpath with Rotation*, AI Memo 684, AI Laboratory, MIT, Cambridge, MA.

[5] Chambon, R. et al. (1987) "An Expert System for Objects Placing in Three-Dimensional Space," *KBES in Engineering: Planning and Design*, 447-459

[6] Donald, B.R. (1983) "The Mover's Problem in Automated Structural Design," *Proceedings of the Harvard Computer Graphics Conference,* Cambridge, MA.

[7] Donald B.R. and Pai, D.K., (1989) *On the Motion of Compliantly-Connected Rigid Bodies in Contact, Part II: A system for Analyzing Design for Assembly,* TR 89-1048, Department of Computer Science, Cornell University, Ithaca, NY.

[8] Erdmann, M. and Lozano-Pérez, T. (1986) *On Multiple Moving Objects*, AI Memo No 883, Artificial Intelligence Laboratory, MIT.

[9] Faverjon, B. and Tournassoud, P. (1989) "A practical Approach to Motion Planning for Manipulators with Many Degrees of Freedom," *Preprints of the Fifth International Symposium of Robotics Research,* Tokyo, 65-73.

[10] Fortune, S. and Wilfong, G.(1988) "Planning Constrained Motion," *Procveedings of the Fourth ACM Symposium on Computation Geometry,* 445-459.

[11] Gunn, D.J. and Al-Asadi, H. D. (1987) "Computer-aided Layout of Chemical Plant: a Computational Method and Case Study," *Computer Aided Design*, Volume 19 number 3, 131-140.

[12] Hasan, H. and Liu, C.L. (1986) "A Force-Directed Global Router", *Proceedings of the Stanford Conference on VLSI Design*, 135-150.

[13] Jacobs, P. and Canny, J. (1989) "Planning Smooth Paths for Mobile Robots,"*Proceedins of the International IEEE Conference on Robotics and Automation*, Scottsdale, AZ, 2-7.

[14] Khatib, O. (1986) "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, 5(1), 90-98.

[15] Kobayashi, Y. et al. (1986) "Knowledge Representation and Utilization for Optimal Route Search," *IEEE Transactions on Systems, Man, and Cybernetics*, 454-462.

[16] Latombe, J.C., (1976) "Artificial Intelligence in Computer-Aided Design: the Tropic System" *IFIP Working Conference,* Austin, TX, edited by J.J. Allan, North Holland, 61-120.

[17] Latombe, J.C. (1979) "Failure Processing in a System for Designing Complex Assemblies," *Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, Tokyo, Japan.

[18] Latombe, J.C. (1988) "Spatial Reasoning: From Robotics to Engineering," *Second Toyota Conference on Organization of Engineering Knowledge for Product Modeling in Computer Integrated Manufacturing*, Nagoya, edited by T. Sata, Elsevier, 83-105

[19] Latombe, J.C. (1990) *Robot Motion Planning*, Kluwer Academic Publishers, Boston.

[20] Laumond, J.P. (1987) "Finding Collision-Free Smooth Trajectories for a Non-Holonomic Mobile Robot," *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Milan, Italy, 1120-1123.

[21] Lozano-Pérez, T. (1983) "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, C-32(2), 108-120.

[22] Mitsuta, Toru et al. (1986) "A Knowledge-Based Approach to Routing Problems in Industrial Plant Design", *6th International Workshop on Expert Systems and Their Applications*, Avignon, France, 237-255.

[23] Natarajan, B.K. (1989) "Some Paradigms for the Automated Design of Parts Feeders," *International Journal of Robotics Research*, 8(6), 98-109.

[24] Nilsson, Nils J. (1980) *Principles of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., CA.

[25] Ó'Dúnlaing, C. and Yap, C.K., (1982) "A Retraction Method for Planning the Motion of a Disc," *Journal of Algorithms*, 6, 104-111.

[26] Preparata, F.P. and Shamos, M.I. (1985) *Computational Geometry: An Introduction,* Springer-Verlag, New York.

[27] Reif, J.H., (1990) "Complexity of the Mover's Problem and Generalizations," *Proceedings of the 20th IEEE Symposium of Foundations of Computer Science*, 144-154.

[28] Sechen, C. (1988) *VLSI Placement and Global Routing Using Simulated Annealing,* Kluwer Academic Publishers.

[29] Schwartz, J.T. and Sharir, M. (1983) "On the Piano Movers' Problem: I. The Case if a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Communications on Pure and Applied Mathematics*, 36, 345-398.

[30] Schwartz, J.T., Sharir, M. and Hopcroft, J. (1987) *Planning, Geometry, and Complexity of Robot Motion,* Ablex, Norwood, NJ.

[31] Sheridan, H.C. (1976) "An overview of a CASDAC subsystem-Computer-Aided Piping Design And Construction (CAPDAC)," *Naval Engineers Journal*, 87-98.

[32] Sifrony, S. and Sharir, M. (1987) "A New Efficient Motion Planning Algorithm for a Rod in Two-Dimensional Polygonal Space," *Algorithmica*, 1, 367-402.

[33] Stallman, R.M. and Sussman, G.J. (1977) "Forward Reasoning and Dependency-Directed Backtracking in a System for Computed-Aided Circuit Analysis," *Artificial Intelligence*, 9(2), 135-196.

[34] Wangdahl, G.E. et al. (1974) "Minimum-Trajectory Pipe Routing," *Journal of Ship Research*, Vol. 18, No.1, 46-49.

[35] Wilson, R.H. and Rit, J.F. (1990) "Maintaining Geometric Dependencies in an Assembly Planner," *Proceedings of the IEEE International Conference on Robotics and Automation,* Cincinnati, OH, 890-895.

[36] Zhu, D. J. and Latombe, J. C. (1989) *New Heuristic Algorithms for Efficient Hierarchical Path Planning*, Tech. Report STAN-CS-89-1279, Computer Science Department, Stanford University. (To appear in *IEEE Transactions of Robotics and Automation.*)