USING THE INTERFERENCE GRAPH FOR THE DYNAMIC ORDERING OF VISION PROCESSING TASKS

by

Kiriakos N. Kutulakos and Charles R. Dyer

Computer Sciences Technical Report #977

October 1990

Using the Interference Graph for the Dynamic Ordering of Vision Processing Tasks

Kiriakos N. Kutulakos Charles R. Dyer

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

Abstract

Most computer vision algorithms consist of a number of tasks that share a large amount of geometrical information. In this paper we address the problem of using this shared information to order the algorithm's tasks efficiently. We focus our attention on hypothesis-guided algorithms, describing them as a collection of interdependent, constrained search tasks. The dependencies between these tasks determine which task orders produce correct results, and provide the conditions under which one task order is more efficient than others. We introduce an *interference graph representation* for hypothesis-guided algorithms that models explicitly the dependencies between tasks, and hence, contains the information necessary to select efficient task orderings. Even though most previous approaches to vision processing assume a static ordering of tasks, we show that the efficiency of a particular task ordering is image-dependent and therefore no static ordering can be efficient, on average. This result makes the interference graph representation an essential tool for designing efficient, dynamic, task-ordering strategies.

1 Introduction

While many low-level image processing algorithms require uniform processing over the entire image, most intermediate-level and high-level algorithms perform variable types of computations that are dependent on the image data. This is because parts of an image contain more information than others, and results of these types of dependencies from early processing stages can be used in the form of predictions to guide later stages. As a result, many algorithms in computational vision can be naturally characterized using the hypothesize-and-test paradigm, including the following:

- Hypotheses are used to focus attention on important or informative portions of the image data in problems such as scene analysis [11], road detection from aerial images [6], and in dynamic image analysis problems [4].
- Hypotheses can model the results of predictions in active vision: In problems like the computation of shape-from-x, an active vision system can generate a number of hypotheses for the possible shapes of parts of object surfaces. The system subsequently tries to verify one or more of these hypotheses by focusing the camera lens on a particular region of the object's surface, or by altering other imaging system parameters [1].
- Object recognition can be performed by creating some initial hypotheses and then incrementally verifying or refining them [16], by verifying predictions that are the result of partial or inexact matches [14], or by verifying predictions that are created using geometrical constraints [3].

One important reason for using hypothesis-guided, focus-of-attention methods for vision stems from the combinatorial nature of the above problems [18, 8, 9, 7]. For example, point-to-point matching, one of the most basic operations required for computing shape-from-x and estimating object motion, has a solution space of size N^d, where N is the number of points and each point can be matched with up to d points. Thus in most real-world problems only a small portion of the total solution space can be searched. In such settings, an algorithm is more efficient if it examines, on average, a smaller part of the solution space before finding a solution.

Even though the particular hypothesize-and-test mechanisms used for examining a problem's solution space can vary, the underlying algorithms share an important common property: They all perform a number of tasks that produce information (i.e., make predictions) and a number of tasks that can use it to constrain their search. Low-level image processing tasks such as edge detection and segmentation have traditionally been viewed as basic sources of information about the images to be analyzed, whereas higher-level object recognition tasks can make use of that information in the form of constraints [7, 3]. A similar situation occurs in game-playing using alpha-beta search, where information produced in the early stages of the search (i.e., values for α and β) can be used to restrict the search and improve performance over exhaustive minimax search. It is thus evident that if a task $\mathcal{A}$ can generate predictions constraining the solution space of some other task $\mathcal{B}$, $\mathcal{B}$ can be performed more efficiently if it is performed after $\mathcal{A}$.

In this paper we claim that the efficiency of hypothesis-based algorithms is largely dependent upon the order in which their component vision-processing tasks are performed. We also claim that

The support of the National Science Foundation under Grant No. IRI-8802436 and the University of Wisconsin Graduate School is gratefully acknowledged.

this dependency exists only when there are search-limiting relationships between the algorithm's tasks. The main result of the paper is a new framework for representing such hypothesize-and-test algorithms. The framework, which makes explicit the relationships between an algorithm's component tasks, can be used to improve the algorithm's efficiency by indicating orders for performing its tasks efficiently. The importance of our framework lies in the fact that most vision processing algorithms can be naturally decomposed into a number of related tasks. We show that these relationships are due to the geometric nature of the problems and can be used to improve the efficiency of the algorithms.

1.1 Related Work

The assumption that the tasks in the decomposition of a vision processing algorithm are dependent on each other comes into sharp contrast with many previous approaches to vision processing. Relaxation techniques [5] perform a single operation at all points in an image, in parallel. These techniques therefore assume that the operations performed on each point are completely independent. Similarly, many image-processing algorithms employing data-level parallelism in massively-parallel machines take for granted this operation independence. In [19], for example, object recognition is performed by first estimating the generalized Hough transform of the image for each of the object models available. The approach implicitly uses the assumption that each region of the Hough space is independent from the regions already computed. The algorithm proceeds by partitioning the Hough space into regions and computes each one of them separately. Usually, however, partial results from already-computed regions can be used to determine whether a particular region of Hough space needs to be computed or not.

The idea of establishing a partial ordering on a set of computational tasks such that the dependencies between them are utilized, is not new. In [6], the tasks of detecting road features were divided into two categories, the ones that could produce an accurate feature identification once the feature's approximate position was known, and those that could detect the existence of a feature in a particular region without being able to identify it. The authors noted the way information was created and used by the various tasks and suggested that the order in which they are performed should depend on what kind of information they produce or on how they use that information. The usage of these producer-consumer relationships is also implied in many vision processing algorithms that follow top-down, coarse-to-fine approaches for object recognition [16] or shape-from-x computation [12].

A first approach to the task ordering problem is to order the tasks statically. This approach is followed in most cases where the same image processing task must be performed on all parts of the image. The problem with this approach is that in many cases, static orderings are not very efficient. For example, iterative approaches to optical flow calculation [20] perform the tasks of calculating the values of the optical flow field corresponding to each point in the image in parallel. If we have prior information about the positions of the objects in motion, we could instead calculate the flow field only for those regions. This approach is more efficient because we calculate the values of the optical flow only in image regions that contain useful information. On the other hand, the order in which parts of the image are processed will depend on the image itself and domain knowledge that is available. Thus, in order to increase processing efficiency by using prior knowledge we must dynamically order the tasks to be performed.

The concept of dynamically ordering the tasks to be performed by an algorithm has also been

used in other domains. Task scheduling is used in the description of the blackboard model [17]. The model implies the existence of a scheduler that, given a set of knowledge sources and a set of objects, selects an object and a knowledge source that will be applied to it at each step of the computation. Task scheduling has also been an issue in database query optimization [13], where the algorithm corresponds to an access plan and the tasks correspond to database queries. In this domain the scheduler must find the most efficient order to perform the set of queries (i.e., find an efficient access plan).

1.2 Overview

In the following section we develop a computational framework where an hypothesize-and-test algorithm is decomposed into a number of vision processing tasks. The tasks are modeled as search problems in the solution space of the hypothesize-and-test algorithm. The execution of the original algorithm is thus modeled as an attempt to find a set of consistent solutions to a collection of search problems. We show that this framework is very useful in determining how the algorithm's processing tasks can interact and we define a compact graph representation that makes explicit all of these interactions.

Even though it would seem that focusing only on hypothesize-and-test algorithms is very restrictive, we show in Section 3 through the use of two examples that many algorithms that are not traditionally viewed as such can be formulated using our framework. The main result of Section 3 is the construction of representations for two intermediate-level vision processing algorithms, a feature detector that uses a multidimensional Hough transform technique and an algorithm that computes shape-from-stereo by integrating techniques for feature matching, surface fitting and contour detection [12]. More importantly, we show that our framework can be very useful in determining how we can improve the efficiency of these and other algorithms. There are many other types of algorithms that could also have been chosen, but we use these two to demonstrate that our framework is not inherently restricted to a small set of vision processing modules.

In our framework, the two algorithms are decomposed into a number of tasks. As originally defined, the algorithms used a static ordering of those tasks. We analyze under which conditions this static ordering can prove inefficient, motivating alternative execution strategies that dynamically order the tasks. Finally, in Section 4 we generalize the results of Section 3 and discuss the usefulness and applicability of the framework.

2 Formalizing Hypothesis-Guided Algorithms

2.1 Hypothesis-Guided Visual Processing as a Constrained Search Problem

Algorithms that use the hypothesize-and-test paradigm share a number of general characteristics: (a) The control of these algorithms is guided by predictions, (b) predictions are formed by examining the image data, considering any domain knowledge available and taking into account the results of the algorithm's execution up to that point, and (c) predictions are used to select hypotheses that must be subsequently verified. Consequently, an algorithm can be decomposed into a number of components, each of which has a specific task, to verify selected hypotheses and to form predictions based on the results of those verifications.

To illustrate this approach, consider an hypothesize-and-test algorithm that addresses the following problem from intermediate-level processing. Given a set of line segments in a image, we want

to group them using perceptual organization techniques. For example, in [14, 15] these techniques are employed to group points and line segments based on geometrical properties such as proximity, collinearity and parallelism; the groupings are used at a later stage to guide high-level model matching tasks.

Initially, the only knowledge we have about the image to be processed is a number of problem-specific assumptions, such as the assumption that all the curves in the image are smooth. An hypothesize-and-test algorithm can use this assumption by trying to verify only hypotheses for segment groupings that approximate smooth curves. This set of hypotheses that the algorithm may need to verify constitutes the set of initial hypotheses of the algorithm. At any point in time during the execution of the algorithm, there will be a number of hypotheses from this initial set whose validity or falsity has not been established. The algorithm must then focus attention on one of these hypotheses, such as a particular grouping of lines in a region of the image, and try to verify it. In order to determine whether the chosen hypothesis is valid, the significance of the grouping is quantified using perceptual organization rules. Based on the outcome of the verification process and the available domain knowledge, the algorithm can make predictions: If the hypothesis is verified, we can reject all the hypotheses that imply different groupings for the line segments mentioned in the verified hypothesis. On the other hand, if the hypothesis is proven false, all other groupings that imply the false grouping must be false too. Therefore, each hypothesis (i.e., segment grouping) corresponds to a pair of (sound) constraints on the solution space of the problem; one constraint is applied to the space when the hypothesis is verified, whereas the other is applied in the case that the hypothesis is proved false. The algorithm will terminate when every segment is included in some group of size at least 1, each group of segments approximates a smooth curve, and the groupings are mutually consistent (e.g., two segments are not grouped both on the basis of proximity and collinearity).

It is easy to see from the above example that the objective of hypothesis-guided algorithms is to identify a set of hypotheses that are contained in the algorithm's initial set of hypotheses, can be verified in the image data, satisfy all the domain knowledge constraints, and constitute a complete solution to the target problem. The task performed by such algorithms is directly analogous to constrained search tasks that use the *interpretation tree* [7]: Each node in the tree represents an hypothesis. Search begins by selecting and testing an hypothesis at the first level of the tree. Depending on the outcome of the verification process a suitable constraint is applied. At the next level of the tree, a search is conducted to find an hypothesis that is consistent with the already-formed constraints. Search terminates when an hypothesis is found that constitutes a complete solution to the target problem. Note that the search process, as described, does not need to backtrack up the interpretation tree as in [7] but on the other hand, the height of the tree is exponential. The perceptual organization example uses hypotheses that build groupings based on parallelism, collinearity and proximity relationships and whose validity can be determined using distinct perceptual organization rules. In general we can think of hypothesis-guided algorithms as using hypotheses that belong to a number of different classes, depending on the techniques used to verify them.

During the execution of an hypothesize-and-test algorithm several hypotheses from a given class will be examined. We can think of this process of examining hypotheses from a particular class as being a search conducted in a search space associated with that class. The space will contain states for all hypotheses that (a) belong to the class, and (b) are contained in the initial set of hypotheses of the algorithm. For example, one such space is the set of all possible segment groupings based

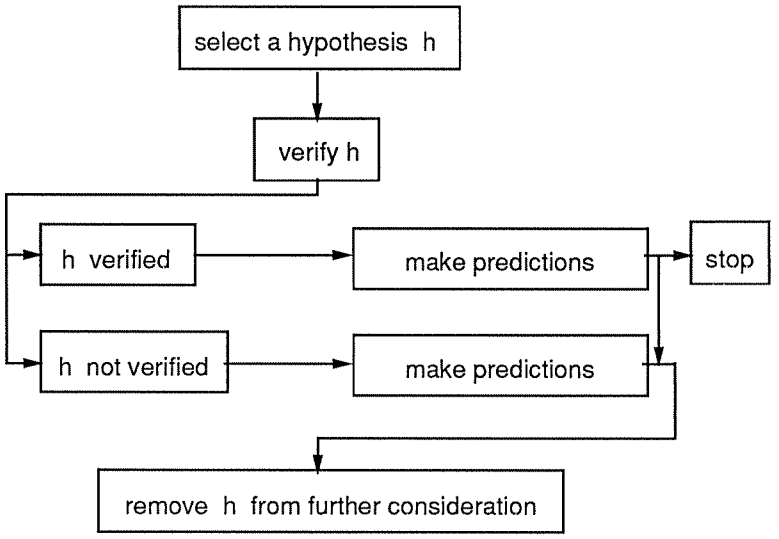

Figure 1: A graphical representation of the steps followed by a search process during the exploration of its associated search space.

on collinearity that in addition give rise to smooth curves. Each time a hypothesis is tested, the search is restricted by the constraints associated with the hypothesis. If a hypothesis is verified and no more constraints can be applied (i.e. search has reached a leaf of the interpretation tree) the hypothesis constitutes a goal state for that space. Since in general the hypotheses selected by the algorithm will belong to different classes, the execution of the algorithm can be thought as a collection of constraint-guided search processes, each of which takes place in a different search space (i.e., each process searches a private interpretation tree). In addition, the complete solution to the target problem dictates that all the hypotheses verified must be mutually consistent. This in turn implies that even though the search spaces are explored independently, the goal states found in each of these search spaces must be mutually consistent. If no such set of goal states exists, there is no solution to the target problem and the hypothesize-and-test algorithm will eventually fail.

2.2 A Model for Hypothesize-and-Test Algorithms: Search Space Dependencies and the Interference Graph

The search subproblems in the decomposition of the original hypothesize-and-test algorithm discussed earlier are solved incrementally, by repeatedly performing a standard sequence of steps. Figure 1 summarizes this sequence. Each time the steps are followed, new constraints are applied to the acceptable goal states of the problem space, reducing the size of the space that needs to be explored. We can think of this sequence of steps as an operator application to the problem space that results in its reduction.

The above observation allows us to build a basic model for hypothesize-and-test algorithms in two steps. First, we organize the hypotheses into classes depending on the type of image-processing operations needed to verify them, and assign a search space to each class. These spaces represent

the search subproblems that must be solved before the algorithm can terminate. Initially they will contain states corresponding to all possible solutions of the subproblems. Having defined a decomposition of the algorithm's problem into subproblems, the next step is to define an operator for each of the subproblems that (a) selects an hypothesis corresponding to that problem space, (b) uses a verification procedure to check if the hypothesis is correct, (c) makes predictions indicated by domain knowledge in order to reduce one or more problem spaces, and (d) if a goal state is found, determines if it is consistent with the goal states found in the other problem spaces. Thus we assume that an operator that is assigned to a problem space can verify hypotheses associated with its space only, but it can affect other problem spaces. Furthermore, we assume that termination detection is distributed across all of the operators since it can take place only when all operators agree to terminate.

Our basic model gives a precise description of how each one of the defined search spaces is explored, but fails to show how a search in one space can interact with the search in some other space. The search processes in two spaces can interact when an operator corresponding to one of the spaces is applied and makes decisions that can affect the other. In our perceptual organization example, if two segments A and B can be grouped using a collinearity rule (see Figure 2(a)), this grouping will be part of a goal state g in the space $\mathcal{S}$ of collinearity groupings (Figure 2(b)). If g is part of the original problem's solution then an acceptable goal state g' in the space $\mathcal{S}'$ of groupings based on parallelism must not have A and B grouped together (Figure 2(c)). This follows from the uniqueness constraint that two segments cannot be grouped in more than one way (e.g. A and B cannot be collinear and parallel). Thus, if an operator decides to select g as its goal state, it automatically affects the states in $\mathcal{S}'$, since the number of its acceptable goal states is reduced.

This situation means that the search processes in the two spaces cannot be completely independent from each other, but must instead be coordinated by controlling the order in which they are performed. One way of ordering them is to find a goal state in one of the spaces and then start the search process in the other space (i.e., the space reductions are not interleaved). Such an approach may also imply a fixed order for solving the two search problems (see Section 3.2). Another way to force consistency among goal states is to ensure that both spaces are not reduced independently. Instead, if one of the spaces is reduced, information about which states were removed from the space can be communicated to the other one. This strategy will ensure that only globally-consistent states are considered by each search process. We call this form of dependency between spaces a *strong dependency* since their goal states must always be consistent with each other. The notion of strong dependencies can also be illustrated using the terminology of Grimson's constrained search paradigm [7, 9]: A strong dependency between two search spaces $\mathcal{S}$ and $\mathcal{S}'$ exists if we can arrive at a leaf in the interpretation tree of $\mathcal{S}$ that is inconsistent with the constraints applied to $\mathcal{S}'$ or vice versa (i.e., the constraints applied to $\mathcal{S}$ are not sound with respect to the states of $\mathcal{S}'$). Depending on whether a fixed order for searching the spaces is implied, strong dependencies can be directed or undirected.

The predictions formed by an operator can also affect the search processes in other spaces. For instance, assume that each state is assigned a probability of being a goal state. If we have no prior knowledge about the search problems involved, we must assume that all states are equally likely to be goal states. If segments A and B are found to be parallel and an operator assigned to the space of collinearity groupings finds segments B and C to be collinear, the operator can predict that segments A and C are parallel by increasing the probability that this state/hypothesis is also a goal state. Dependencies of this sort do not restrict the form of the acceptable goal states in the

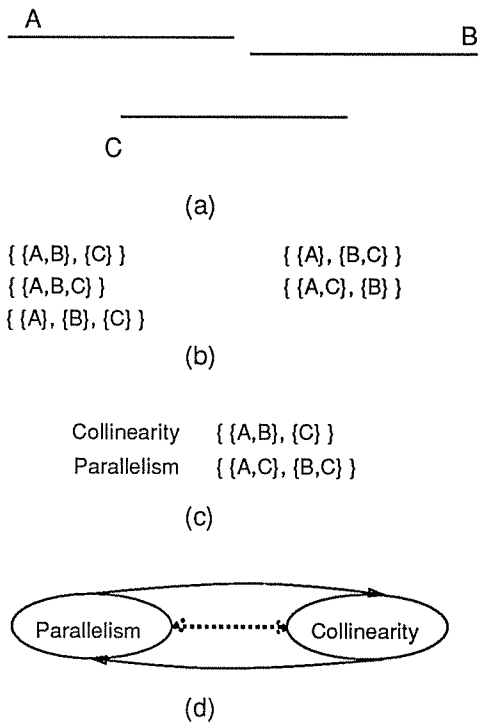

Figure 2: (a) Three line segments to be grouped using parallelism and collinearity rules. (b) A state space containing all the possible groupings of the three segments based on collinearity. (c) A set of consistent goal states (g and g') for the collinearity and parallelism segment grouping problems. (d) The interference graph representing the dependencies between the two grouping problems. Directed, solid arcs represent weak dependencies. Dotted arcs represent strong dependencies.

two spaces, but give hints on their possible forms. These hints can make the search processes more efficient by first considering the best forms suggested by those hints.

Hints are essentially weaker forms of constraints: They can be used to restrict the search by reducing their associated search space, but they need not be sound. Therefore, if a goal state cannot be found in the space reduced by such constraints, the search process must backtrack, revoking these constraints. If an operator of some space $\mathcal{S}$ forms predictions that can affect other spaces and these effects are taken into account when the spaces are reduced, information about the state of the search process in $\mathcal{S}$ is transferred to the search processes in the other spaces. Thus there is a direction for these dependencies. We will refer to these dependencies as *weak dependencies* to contrast them with the strong dependencies described earlier.

In our model, the *interference graph* represents all the strong and weak dependencies between tasks: Each node in the graph represents a search space. Each arc represents either a strong or a weak dependency between two spaces, thus showing how the search processes between two spaces can affect one another (Figure 2(d)). The interference graph will be used in Sections 3 and 4 to determine operator application orderings that produce consistent solutions to a given problem and

allow search space reductions to take place between search processes.

3 Hypothesis-Guided Vision Algorithms

The existence of strong and weak dependencies between the tasks of a hypothesis-guided vision algorithm is the key factor in determining an efficient order for performing these tasks. To illustrate this point we consider two example problems from the vision literature. The first example, Hough transform processing, is a relatively low-level image processing problem that can be used for line or general curve detection. The second example addresses the problem of calculating surface from stereo, a shape-from technique for more intermediate-level processing. The purpose of this section is to show by example that our methods are applicable across a wide variety of vision-processing problems that vary in generality, complexity and the amount of domain-dependent information they use.

3.1 The Hough Transform for Feature Detection

3.1.1 Overview of the Algorithm

The Hough transform and the Generalized Hough transform [2] are common image transformations used to detect clusters of image points in lines, parameterized curves, or arbitrary shapes. The Generalized Hough transform has also been used for object/shape recognition in settings where both noise and partial occlusion are present [16].

Shape recognition using the Hough transform is usually performed by (a) parameterizing the shapes or features to be detected using a parameter vector $\overline{P}$, (b) developing a voting scheme consisting of a space of buckets for the possible values of $\overline{P}$ (the Hough space), where each point in the image votes for a number of buckets, and (c) designing a peak detector that can accurately determine which buckets contain peaks that reflect an actual image property. Since each vote essentially increases the confidence of a match between an object model and the image data, peaks in the space correspond to high-confidence matches.

We will use the Hough transform to identify objects in a given image. We assume that the image can contain projections of a number of objects, and that we are given a collection of 2D object models represented by graphs. A simple object and its corresponding model are shown in Figure 3. For simplicity we will assume that the objects are composed of identical parts. Thus our algorithm will need to search a single Hough space associated with the shape of the object's parts.

As pointed out in [10], there are two problems with using the Hough transform for feature detection. The first problem involves the size of the space; given a 512×512 image, three orientation parameters having 64 possible values each and a scaling parameter having 16 possible values, the size of the Hough space, 2^{40} buckets, makes it impossible to fully construct the space and then search for the peaks it contains. For this reason we will assume that our algorithm builds the space incrementally, searching for peaks in the small region of the space that has already been constructed. The second problem involves the robustness of the Hough space peak detector with respect to noise and occlusion. It is argued in [10] that false peaks can be very frequent in images where there are large amounts of noise or occlusion. In order to distinguish between real peaks and false ones we will assume that we have available additional verification techniques that can reliably decide whether a peak found by our algorithm corresponds to a feature in the image.

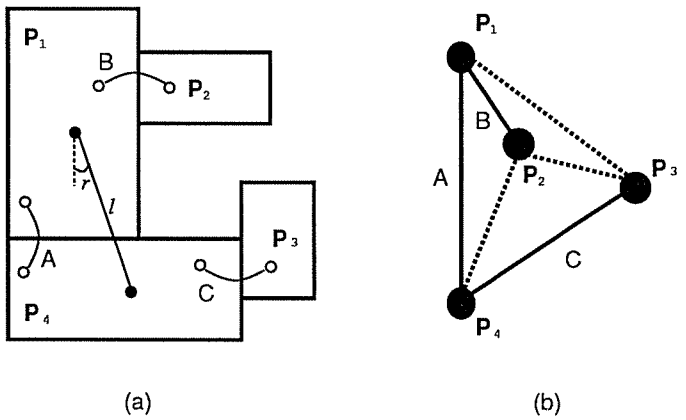

Figure 3: (a) A sample object containing four parts. (b) The corresponding 2D object model. Vertices denote user-defined shapes, edges represent geometric relations between the parts. Edge A represents the geometric relation between the centers of the parts P_1 and P_4. Dotted edges in the graph represent relations between non-adjacent parts.

3.1.2 In Search of an Efficient Task Ordering Strategy

In general, given an image and a feature description, there is no reason to assume that the peaks in the associated Hough space will be uniformly distributed. This implies that some parts of the Hough space will contain more information than others. Let us now assume that the Hough space is two-dimensional and that it is divided into four areas $H_1, \ldots, H_4$. A naive approach to the peak detection problem would be to conduct a search for peaks in H_1 first, and then in $H_2, \ldots, H_4$. This approach involves the risk of computing and searching a whole area of the Hough space without finding a single peak. A second approach would be to interleave the search processes in the four areas of the space. In this case the search subproblems associated with each of the areas $H_1, \ldots, H_4$ are considered independent and the original problem is decomposed into four identical subproblems. There are two questions that arise from such an ordering. The first one is whether this order is correct, i.e, whether we can consider the subproblems as independent. The second question is whether there is a more efficient way of solving the subproblems. Both of these questions can be readily answered when we describe the tasks involved using our framework. For simplicity we will assume that the given image contains a single instance of the object.

Each part of the object appearing in the image will give rise to a single peak in the Hough space. Conversely, since all object parts have the same shape, each peak in the Hough space will have one of many possible causes, depending on which part of the object gave rise to the peak. A hypothesis in our model will predict that a point in Hough space is a peak and will identify the object and the particular part that caused it. We can think of these hypotheses as belonging to four classes associated with the areas $H_1, \ldots, H_4$. A state in our model will correspond to one of the above hypotheses, and thus each of the areas $H_1, \ldots, H_4$ is assigned a search space S_i, $i = 1, \ldots, 4$. Search

is conducted by applying operators to the spaces S_i: When an operator is applied to a space it selects a state (i.e., computes the value of a point in H_i) and checks if that point is a valid peak. Hypothesis verification involves first determining whether that point is a local maximum in the Hough space. If it is a local maximum, verification proceeds by checking whether the peak corresponds to an image feature. If the point is a valid peak but the operator cannot determine which part of which object it corresponds to, the search continues in a reduced space; the fact that a peak is found at a particular point restricts the possible interpretations for the object. This in turn restricts the possible locations for peaks in the Hough space, and thus restricts the set of states in the search space that can be goal states.

We can show that the method of interleaving the searches in the four areas is correct using the following argument. Assume that parts of the object give rise to peaks in more than one area of the Hough space. When an operator verifies that a given peak corresponds to a real feature in the image, the search space associated with that operator is reduced by considering the possible interpretations for that peak. Search in that space then necessarily terminates since no further progress can be achieved. By definition, when all search processes terminate, the solution to the object identification problem is given by taking the intersection of the interpretations associated with the reduced search spaces. Therefore we do not get wrong results when we search the four spaces independently. Using the terminology of our model, there are no strong dependencies between these spaces since the sets constraints applied to each one of the search spaces are independent and complete [7].

Even though the search spaces can be explored independently by performing a number of constrained search tasks, there is a way for the tasks to communicate results from their partial searches. When a search space is reduced, the reduction is caused by a constraint on the possible interpretations for the object in the image. This constraint can also be applied to the other search spaces, since the final result of the object recognition algorithm will be the intersection of the interpretations given by the four search tasks. Thus, when an operator reduces its corresponding space it can also reduce the other spaces too, in order to restrict the other search processes. In our terminology, there are weak dependencies between the four search spaces. Moreover, the constraints giving rise to these dependencies are sound, so the search processes will never need to backtrack.

Assume that the time needed to apply an operator to its corresponding space is constant. This is reasonable, since the operators in this example are essentially peak detectors that, in addition, can generate constraints based on the outcome of the verification. One would expect that the time needed to check if a point in Hough space is a peak does not depend on the image. Furthermore, the number of constraints generated will depend only on whether the verification succeded or not, and not on any already-existing constraints (for simplicity we will assume that no constraints are generated if an hypothesis is not verified). Under the above assumptions, we can determine the efficiency of a given approach by counting the number of times the operators are applied to their spaces. If N_i is the number of operator applications before search terminates in space S_i, then the method of interleaving the operator applications (as well as the naive approach) needs a total of $N_1 + \cdots + N_4$ applications. On the other hand, if we use the weak dependency information and let the operator that terminates first reduce the other three spaces, we need at most $4\,(\min_i \{N_i\}) + M$ applications, where M is the total number of points in the Hough spaces restricted by that operator. This is because the other operators must verify the peaks implied by those predictions. The performance of this approach can be significantly better than the first one if most of the peaks are contained in a single space, S_i. In such a case a peak is likely to be found in S_i much sooner than in the rest of

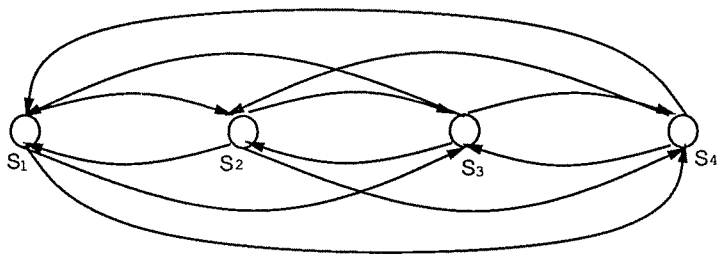

Figure 4: An interference graph for the operators associated with spaces $S_1, \ldots, S_4$. Arcs denote weak dependencies created using the geometric relations A, B and C of Figure 3. The interference graph is complete, since any operator can be the source of predictive information affecting the other spaces. There are no strong dependencies in the graph.

the spaces (i.e., $N_i \ll N_j$ for $i \neq j$). The exact difference will depend on the way the operators are implemented as well as on the distributions of the peaks in the four spaces.

If we assume that not all peaks are needed to describe the object with a small error probability, then the number of interpretations will be small too (i.e., $M \ll N_j$) [8]. Under these conditions it is evident that $4\,(\min_i \{N_i\}) + M \ll N_1 + \cdots + N_4$. Note that the operator applications performed by the other operators do not give any useful results and do wasted work since their spaces are eventually restricted to a small set of states by an operator from another space. Had we known a priori the area containing most of the information, we would apply only the operator corresponding to that area and wait until it reduced the other spaces. This implies a lower bound on the number of operator applications of $min_i\{N_i\} + M$.

We can make better use of the weak dependency information by not restricting the search process to follow a static ordering. We can start the search by interleaving the operator applications, but when one of the operators finds peaks and starts reducing its space at a higher rate, we can focus attention on that operator. The search can be focused by scheduling that operator's applications more often with the hope that the information gathered by that search process will eventually be greater. In addition, there is no reason for the operator to reduce the other spaces only after it has terminated search in its own space. Rather, when the number of constraints applied to its space becomes large, these constraints can be used to effectively reduce the search in the other spaces too.

All of the above heuristics can be effective only if there are weak dependencies between the search spaces. If we do not assume in the original problem description that any domain information is available, then there are no weak dependencies between the four search spaces, and the method of statically ordering the operator applications is as good as any other one. Finally, note that if there is a weak dependency from space S_1 to space S_2 but not one from S_2 to S_1, it would be more efficient to apply the operator associated with S_1 first, since there is a possibility that the operator will also reduce S_2, making the search process in S_2 more restricted. This direct relationship between the weak dependencies and the ordering of the operator applications makes the interference graph a very important tool for identifying efficient task orderings. Figure 4 shows the interference graph describing these dependencies between the spaces $S_1, \ldots, S_4$.

3.2 Surface from Stereo

3.2.1 Overview of the Algorithm

In the general surface-from-stereo problem we are given a pair of images (left and right) showing a scene from two different viewpoints. For each point in the scene the goal is to find its distance from the viewer. We discuss an approach to this problem presented in [12] that combines feature matching, surface fitting, and feature detection. This algorithm makes the assumption that the scene contains objects with smooth surfaces, and computes a collection of quadratic patches that best describe the object surfaces in the scene.

In [12] the surface-from-stereo problem is decomposed into a number of subproblems. Surface-from-stereo is computed for three different image resolutions. For each resolution, the left image is partitioned into non-overlapping windows, the corresponding areas in the right image are found for each window, and surface-from-stereo is computed for these pairs. These two types of decompositions are data-level decompositions, i.e., identical computations are performed on different parts of the data. The algorithm then employed a collection of techniques to solve the surface-from-stereo problem for a pair of image regions at a particular resolution.

Surface-from-stereo calculation for a particular pair of image regions is performed by computing the zero-crossings in the two regions, finding all zero-crossing points in a window of the left image that can match a zero-crossing point in the corresponding right image region, and finally restricting the possible point-to-point matchings to those that give rise to planar surfaces. Since more than one planar surface fit may be found, they are further constrained by examining the planar fits of neighboring regions together and considering only those that can be approximated by quadratic surfaces. A quadratic surface approximates an object's region only if it does not cross a surface discontinuity. Discontinuity contours are detected by examining the relative slopes of the planar approximations to each of the regions in the neighborhood.

This algorithm was presented using a fixed order for performing the above tasks. Surface-from-stereo is first computed for the lowest resolution image pair. The solution to this subproblem is used to select, at the next higher resolution, windows in the right image that correspond to windows in the left image. That is, surface-from-stereo for a particular resolution is computed by first finding point-to-point matchings in all pairs of left- and right-image regions, computing all planar fits for the depth surfaces implied by the matchings, and finally computing the quadratic fits for depth surfaces corresponding to neighborhoods of regions. In the following paragraphs we show that under certain conditions, a dynamic ordering of the tasks can produce a more efficient implementation of this algorithm.

3.2.2 Finding Correct and Efficient Task Orderings

There is a fundamental difference between the feature detection algorithm described in Section 3.1 and the surface-from-stereo algorithm. The feature detection algorithm used search and was very naturally formulated using our model since the search problem could be solved incrementally. In [12] the task of finding point-to-point matches that give rise to planar surfaces is implemented using a non-incremental technique. For this reason we cannot formulate this task as a search process that uses search space reductions to find a solution. Nevertheless, the key factor that determines the efficiency of a particular task ordering is the existence of weak dependencies between subproblems. Even if we cannot interleave the processes for solving the subproblems, we can still order them

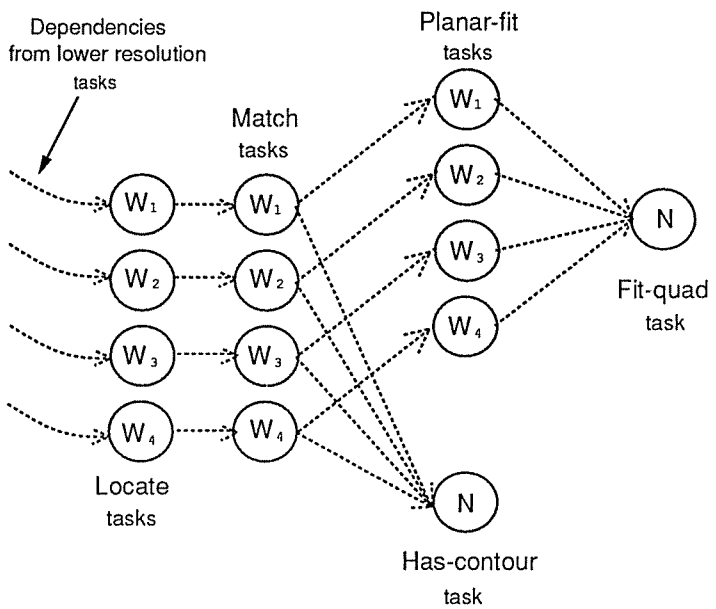

Figure 5: Part of the interference graph for the surface-from-stereo algorithm showing the strong dependencies between the subproblems. The *locate* subproblems involve finding the region in the right image that corresponds to window W_i in the left image. Solutions to the *match* subproblems give all the point-to-point matches for each window W_i. Solutions to the *planar-fit* subproblems restrict the matches to those that give rise to planar surfaces. *Fit-quad* finds a quadratic approximation for the neighborhood containing $W_1, \ldots, W_4$, and *has-contour* determines if the neighborhood contains a surface discontinuity.

taking into account the possible weak dependencies between them.

One property of the tasks in the surface-from-stereo algorithm is that they depend on the results of some other task. For example, the task of finding the point-to-point matches that give rise to planar surfaces in a pair of left- and right- image regions has meaning only after we have found all the possible point-to-point matches in those regions. We must thus follow a particular order for performing these tasks. Such a dependence between the algorithm's tasks is a strong dependency. This way we can build an interference graph for the algorithm's subproblems showing these dependencies and making evident what, if any, choices we have to order their solutions.

The interference graph in Figure 5 shows the strong dependencies between the tasks associated with surface-from-stereo calculation at a single resolution. Obviously, we must find all planar fits corresponding to a neighborhood of windows before we can find the quadratic fit corresponding to that neighborhood. It is also clear that there is more than one valid task ordering for fitting planes to the windows in a neighborhood and for fitting quadratic surfaces in non-overlapping neighborhoods. The interference graph represents the partial order that must be maintained for solving the subproblems, and therefore any total order that is consistent with it is correct.

The tasks that can be initially performed are the minimum (i.e., first) elements in the partial

order. Since in [12] the surface approximations found at a coarse resolution are used to locate the corresponding left and right image regions at higher resolutions, the minimum elements in the partial order of Figure 5 correspond to the tasks of locating the matching left and right image regions at the lowest resolution. This partial order enforces a coarse-to-fine ordering of the tasks.

Even though this partial order is very restrictive, there still are some choices for ordering the tasks efficiently. Efficiency in this context cannot be defined in the same way as in the case of the feature detector algorithm. In that example the search subproblems were solved incrementally by applying operators to their corresponding search spaces. The basic assumption was that the space reduction took constant time and thus the number of operator applications needed to find a complete solution was a reasonable measure of efficiency. In surface-from-stereo we have a fixed number of subproblems $P_1, \ldots, P_n$ that cannot be solved incrementally. This means that any measure of efficiency must be based on the number of steps T_i needed to solve each of the subproblems P_i, $i = 1, \ldots, n$. Since the subproblems are not identical, in general $T_i \neq T_j$, for $i \neq j$. Given a particular order for solving the subproblems, the efficiency of the algorithm can be measured by $T_1 + \cdots + T_n$.

As an example of how task orderings can affect the algorithm's efficiency, consider the problem of computing the quadratic fit for a neighborhood of windows. Assume that the neighborhood contains four windows and we have already computed planar fits corresponding to three of those windows. These three planar surfaces constrain the form of the resulting quadratic surface fit, thereby constraining the final form of the surface. The constraints can in turn be used to provide an acceptable range of slopes for the planar fit corresponding to the fourth window. Even though the task of finding all possible planar fits for a particular window cannot be completed faster, we can speed up the task of finding planar fits that validate a hypothesis for a quadratic fit.

An algorithm for computing the planar fit corresponding to an image window can proceed by checking first if it has constraints on possible slopes for the planar surfaces. If such constraints exist, the algorithm can try to find planar approximations in the constrained range of slopes. In the absence of such constraints or if the constraints are proved false (i.e., the imposed constrains were not sound), the algorithm can determine all possible fits. This modification to the original planar approximation procedure will ensure that in the presence of constraints the original, computationally-expensive procedure will not be used.

The modification we described makes the algorithm solving a particular subproblem behave as a hypothesize-and-test algorithm in the presence of hypotheses/constraints. It is evident that the feasibility of such a transformation is a necessary condition for the existence of task orderings that are more efficient than others. The information used by the transformed algorithms is essentially weak dependency information. In the example of planar fit calculations, the subproblems associated with a particular neighborhood are by definition related to each other since the surface associated with the neighborhood is likely to be smooth.

Similar dependencies exist between subproblems associated with different resolutions: Surface approximations arising from low resolution image pairs can constrain the form of the surfaces corresponding to higher-resolution images and the locations of matching left and right image regions. Part of the interference graph for the surface-from-stereo problem describing these dependencies is shown in Figure 6. Note that information from higher resolution windows can also guide the subproblems in lower resolution images.

We can now see that due to the presence of weak dependencies between tasks there are many cases where dynamically ordering these tasks offers an efficiency improvement. Surface continuity provides constraints for finding planar approximations efficiently since planar approximations in neighboring

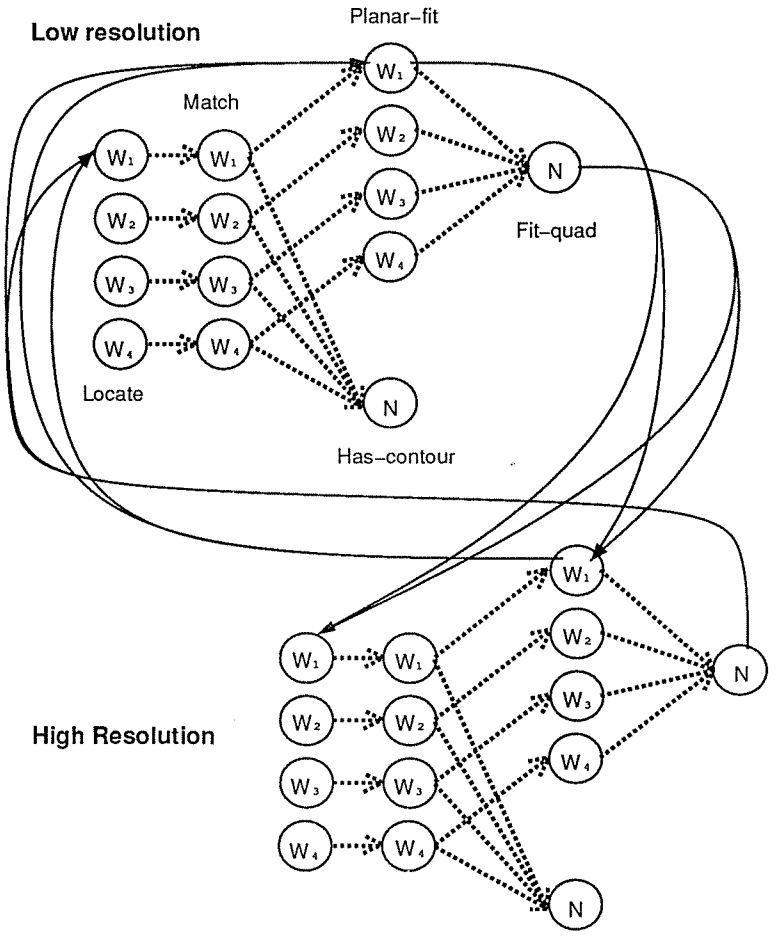

Figure 6: Part of the complete interference graph for the surface-from-stereo problem. Solid arcs represent weak dependencies, dotted arcs represent strong ones. Weak dependencies are only shown for window W_1.

regions will be similar. Regions of the image containing a single object surface will quickly give rise to surface approximations corresponding to those regions. On the other hand, when a surface discontinuity is detected in a neighborhood of windows, it is desirable for the tasks computing the locally-smooth surface approximations within that region to be delayed. When smooth surfaces span large image regions, information about those surfaces can be propagated to the regions containing discontinuities. Information about what task orders are more desirable can be derived from the weak dependencies between subproblems. In effect, the tasks scheduled first should provide weak dependency information to the tasks scheduled at later stages in order to improve their performance. It should therefore be evident that the information provided by the interference graph is of vital importance to the efficient scheduling of the algorithm's tasks.

4 Comparing the Efficiency of Static and Dynamic Task Orders

In the examples in Section 3 we argued that weak dependency information can be used to efficiently order an algorithm's tasks. This claim raises two questions: (a) Whether there is a static task ordering that is optimal (i.e., whether dynamic task ordering is a useful strategy), and (b) if not, whether we can actually devise an optimal dynamic-ordering strategy. In general, a task scheduling strategy must either select a subproblem and solve it non-incrementally (Section 3.2), or select an operator to perform a space reduction (Section 3.1). The decisions made by the task scheduler will thus depend on the granularity of the algorithm's decomposition. The efficiency of a particular task order given an input image can be measured by summing the number of steps needed to perform each of the algorithm's tasks, in that order. Since one task order may be efficient for some input images but very inefficient for all other images, we are interested in average performance.

Average-case analyses require some assumption to be made about the distributions of the algorithm's inputs. We will assume that the input is randomly drawn from a collection of images. We restrict this collection to contain only images that are consistent with the algorithm's domain knowledge. In many cases this limited degree of input randomness is a reasonable assumption. For the algorithm in Section 3.1 this assumption means that the images can contain projections of objects that are consistent with known models and appear at any position and orientation. Therefore, there is no a priori knowledge of which regions in the Hough space contain peaks, even though once a single peak in the space is found, the distribution of the rest of them is determined by the object models.

Under the above assumptions it is easy to see that a dynamic ordering strategy can produce task orders that are at least as efficient as any static task order. To show this we will assume that the dynamic ordering strategy is allowed to make only one choice. Furthermore, we will assume that the choice is between two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$ that perform the same types of computations on different parts of the image data. In this simple setting the only difference between a static and a dynamic task ordering is in the way $\mathcal{T}_i$ and $\mathcal{T}_j$ are scheduled. Since we assumed that the input images are random, a static task ordering will obviously make the most efficient ordering decision on only half of the images. On the other hand, the optimal task ordering strategy, by definition, will always make the right choice.

The most important problem in devising an optimal dynamic task scheduling strategy is its dependence on the input images. Let us consider the case where the partial order has more than one minimum element and no a priori information is available about the input image. In such a case we cannot create an optimal task order since we do not know which task to schedule first, even though such an order does exist. On the other hand, even in the simplest case where the number of steps needed to perform each of the tasks is independent of the image at hand, the problem of finding an optimal (static) scheduling strategy is NP-complete. These results lead us to the conclusion that in order to achieve near-optimum performance we must use a good heuristic strategy that takes into account the image to dynamically order the algorithm's tasks.

It is easy to see that in cases where data-level parallelism is used to decompose some of an algorithm's subproblems, a good heuristic strategy does exist. For the simple problem of ordering the tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, a static order would be equivalent to a dynamic strategy that flips an unbiased coin to make the scheduling decision. Therefore, any heuristic strategy that flips a coin biased towards the more efficient task order will be more efficient than any static order, on average. Since weak dependency information was used to constrain such tasks (Section 3), any heuristic strategy

preferring tasks that can use the information will be biased enough to be more efficient than any static task ordering. This existence result is a good indicator that the weak-dependency information represented by the interference graph can be very useful in determining efficient task orders.

5 Concluding Remarks

We have shown that hypothesize-and-test algorithms can be viewed as performing a set of interrelated, constrained search tasks. This formulation is particularly applicable in the domain of vision processing algorithms where the problems addressed by the tasks share geometrical information. The existence of weak and strong dependencies between tasks determines a partial order on their evaluation. Each total ordering consistent with the partial order produces a correct result, but at a different speed because of the dependencies between the data. These dependencies can be used to increase the efficiency of performing each of the tasks. We have argued that when a large number of images will be processed, no algorithm employing a static ordering of its tasks can always be very efficient. This result led to a new representation, the interference graph, that separates algorithm specification from the description of the order in which the algorithm's tasks must be performed. In addition, the representation can serve as the basis for an efficient dynamic scheduling strategy of the tasks involved.

An important property of the representation is that it can be easily extended to represent algorithms whose tasks do not utilize search. In this case, the only requirement is that the dependencies between the tasks can be used to increase their efficiency. The presence of dependencies between tasks transforms this requirement into a set of constraints that can be used to reduce their complexity during the execution of the algorithm.

Finally, our model is not inherently restricted to serial algorithms. In the case of parallel algorithms, a number of tasks will be performed by different processors. Even though we should not make any ordering restrictions between two tasks on different processors, the interference graph can still be used to affect scheduling decisions for tasks at each processor.

References

[1] J. Aloimonos. Visual shape computation. *Proc. IEEE*, 76:899–916, 1988.

[2] D. H. Ballard and C. M. Brown. *Computer Vision.* Prentice-Hall, 1982.

[3] R. A. Brooks. Model-based three-dimensional interpretations of two-dimensional images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 5:140–150, 1983.

[4] P. J. Burt. Attention mechanisms for vision in a dynamic world. *Proc. 9th Intl. Conf. on Pattern Recognition*, pages 977–987, 1988.

[5] L. S. Davis and A. Rosenfeld. Cooperating processes for low-level vision: A survey. *Artificial Intelligence*, 17:245–263, 1981.

[6] M. A. Fischler et al. Interactive aids for cartography and photo interpretation. Technical report, SRI International, 1979.

[7] W. E. L. Grimson. On the recognition of curved objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11:632–643, 1989.

[8] W. E. L. Grimson. The combinatorics of heuristic search termination for object recognition in cluttered environments. *First European Conference on Computer Vision*, pages 552–556, 1990.

[9] W. E. L. Grimson. The effect of indexing on the complexity of object recognition, 1990. MIT Artificial Intelligence Laboratory Memo 1226.

[10] W. E. L. Grimson and D. P. Huttenlocher. On the sensitivity of the hough transform for object recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12:255–274, 1990.

[11] A. R. Hanson and E. M. Riseman. Visions: A computer system for interpreting scenes. In *Computer Vision Systems*, pages 303–333. Academic Press, New York, 1978.

[12] W. Hoff and N. Ahuja. Surfaces from stereo: Integrating feature matching, disparity estimation, and contour detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11:121–136, 1989.

[13] Y. E. Ioannidis and E. Wong. Query optimization by simulated annealing. *Proc. 1987 International ACM SIGMOD Conference*, pages 9–22, 1987.

[14] D. G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31:355–395, 1987.

[15] D. G. Lowe and T. O. Binford. Perceptual organization as a basis for visual recognition. *Proc. AAAI*, pages 255–260, 1983.

[16] C. F. Neveu, C. R. Dyer, and R. T. Chin. Two-dimensional object recognition using multiresolution models. *Computer Vision, Graphics and Image Processing*, 34:52–65, 1986.

[17] H. P. Nii. The blackboard model and the evolution of blackboard architectures. *The AI Magazine*, pages 38–53, 1986.

[18] J. K. Tsotsos. A 'complexity level' analysis of immediate vision. *International Journal of Computer Vision*, 1:303–320, 1988.

[19] L. W. Tucker, C. R. Feynman, and D. M. Fritzsche. Object recognition using the connection machine. *Proc. of Computer Vision and Pattern Recognition*, pages 871–878, 1988.

[20] S. Ullman. Analysis of visual motion by biological and computer systems. *IEEE Computer*, pages 57–69, 1981.