Simon Thoresen

An efficient solution to inexact graph matching with application to computer vision

Department of Computer and Information Science
Norwegian University of Science and Technology
N-7491 Trondheim, Norway

NTNU Trondheim
Norges Teknisk-Naturvitenskapelige Universitet
Institutt for Datateknikk og Informasjonsvitenskap
Doktor ingeniøravhandling 2007

ISBN 978-82-471-3604-1 (electronic)
ISBN 978-82-471-3599-0 (printed)

ISSN 1503-8181 (Doktoravhandlinger ved NTNU)

Abstract

Graph matching is considered the recognition step in computer vision. Our work uses attributed relation graphs (ARG's) for image representation and analysis, a structural description that we enhance with the complete and coherent spatial knowledge of the source image.

In chapter 1 we reveal a trend where researchers are slowly incorporating more and more spatial knowledge into relational graphs. The ultimate realization of such knowledge in graphs is what we term a "spatially coherent attributed relational graph" whose connectivity is such that any degree of connectivity can be derived from any other. We argue that selective pruning or thresholding of connectivity in a graph is therefore the projection of a solution into a problem instance. This is our first contribution.

This trend degenerates most popular matching methods since these rely on graphs being sparsely connected, and typically collapse as connectivity increases.

In part 1 we introduce our second contribution; an inexact graph matcher whose performance increases with the connectivity of the graphs. Although the method is designed for our spatially coherent graphs, early research shows that it can even be applied to more traditional relational graphs as well. Our graph matcher extends the ideas of semilocal constraints to hold as global constraints. To solve intermediate instances of the assignment problem, we propose a very simple two-pass method that performs with sufficient accuracy.

We present all runtimes in the number of comparisons that are performed on vertices and edges in a problem instance, since this measurement is separate from processor-time – a number biased by implementation skill, processor architecture and operating system. Our method runs by the least possible amount of vertex comparisons, and a tiny fraction of the upper-bound edge comparisons. It has a memory footprint that scales effortlessly with graph sizes.

Finally, in part 2 we present our third and last contribution; an object matcher capable of combining a set of graph matching results derived from multiple vision domains.

Acknowledgements

First of all I would like to thank Richard E. Blake for accepting the responsibilities as my supervisor and for his assistance. I would like to thank the Norwegian University of Science and Technology for their financial support – it has been greatly appreciated and put to good use.

Dr Anderson at the University of Reading gets my highest praise for the incredible support and guidance he gave me while I was visiting for 6 months during 2006, without his help I would not have managed to put this thesis into words. I wish him the best of luck with all of his continued work.

I would like to thank friends and family, especially my parents, Erik and Torunn, my brother Chris and his girlfriend Siv, their soon-to-be son Vetle, my brother Ole-Jørgen, my sister Sara and her husband Ronny Støbakk, and their three wonderful children Majka, Sigurd and Gustav, for their love, understanding, encouragement and belief in me during my work with this thesis.

The images that appear in figure 1.1 is copyright [1], figure 1.2 is copyright [2], figures 1.3, 1.4, and 1.8 are copyright [3], and figure 1.12 is copyright [4].

Contents

List of Figures

List of Algorithms

Listings

Chapter 1

Introduction

1.1 Intention and milestones

Our supervisor, Professor Richard E. Blake at the Norwegian University of Science and Technology (NTNU), has over the course of many years developed a complete and working system for computer vision. It is a complex network of small programs and shell scripts that employs a nonlinear approach to graph matching similar to relaxation labeling (see [5]).

His system can be viewed in two parts; 1) a feature extractor, and 2) an inexact graph matcher. The feature extractor builds a candidate relational graph from a color image (see [6]) which the matcher then determines to which, if any, of all known relational graphs it is the most similar.

Papers such as [3] and [7] detail complete vision systems that deal with NP-completeness by constraining the problem instances such that matching can be done by exhaustive or heuristic searches through state-space. In [8] and [9] the authors deal exclusively with inexact graph matching, and demonstrate their algorithms using random and synthetic structures.

In our thesis we investigate how these approaches to an inherently intractable problem are able to solve it so efficiently. Most vision systems do not avoid the NP-completeness of the graph matching problem, instead they deal with it by employing "smart" context-specific feature extractors.

Furthermore there is a lack of research done in the application of dynamic programming techniques to problems such as graph matching. This is likely due to the fact that graph problems do not transform well into corresponding numerical problems, but we investigate this transformation in light of the "smart" feature

extractors and the goal of suboptimal solutions.

We also consider the choice of primitives used for the structural descriptions of a problem. Primitives such as points and lines imply a large description and a complex matching process, whereas more general primitives allow for smaller descriptions and simpler matching.

Finally, the feature extractor in [6] implicitly suggests an improved resolution on the edge-attributes when compared to the original graphs in [10], but there are no published thoughts or comments on this. What resolution on attribute detail is sufficient for an object description, and what effects does improved resolution imply? How and when, if at all, can the original resolution be used?

The research goal of our thesis is:

To develop a solution to subgraph isomorphism that manages sufficient accuracy in lower time-complexity than other published methods, and which can be applied to computer vision.

The goal is achieved through the following milestones:

1. Investigate how graph extraction is performed in previous litterature; consider trends and their ultimate impact on other proposed methods (achieved in chapter 1.5).
2. Look for a general graph matching distance measurement and its application (this effort spawned the "domain expert" explained in chapter 2).
3. Investigate how ideas in dynamic programming can be applied to solving subgraph isomorphism in pseudo-polynomial time (discussed in chapter 1.6, and contained in the algorithm in chapter 2).
4. Develop an algorithm to solve subgraph isomorphism in light of the result of the previous milestones (achieved in chapter 2).

This chapter introduces the problem area, the underlying litterature and proposes our solution. Part I contains a detailed description of our algorithm, it introduces our measures of success, and demonstrates the efficiency of the method on random graphs. Finally, part II describes and demonstrates the application of the algorithm to computer vision.

(a) Short range infrared image. (b) Examples of searched details.

Figure 1.1: Identification of armored vehicles at short range using infrared images. This example is a reprint from [1].

1.2 Approaches to computer vision

Computer vision is the study and application of methods which allow computers to "understand" image content or content of multidimensional data in general. The term "understand" means here that specific information is being extracted from the image data for a specific purpose: either for presenting it to a human operator (e.g., if cancerous cells have been detected in a microscopy image), or for controlling some process (e.g., an industry robot or an autonomous vehicle). The image data that is fed into a computer vision system is often a digital gray-scale or colour image, but can also be in the form of two or more such images (e.g., from a stereo camera pair), a video sequence, or a 3D volume (e.g., from a tomography device). In most practical computer vision applications, the computers are pre-programmed to solve a particular task, but methods based on learning are now becoming increasingly common.

In biological vision and visual perception real vision systems of humans and various animals are studied, resulting in models of how these systems are implemented in terms of neural processing at various levels. Computer vision, on the other hand, studies and describes technical vision system which are implemented in software or hardware, in computers or in digital signal processors. There is some interdisciplinary work between biological and computer vision but, in general, the field of computer vision studies processing of visual data as a purely technical problem in the environment of computer devices.

The field of computer vision can be characterized as immature and diverse. Even though earlier work exists, it was not until the late 1970's that a more focused study of the field started when computers could manage the processing of large data sets such as images. However, these studies usually originated from various other fields, and consequently there is no standard formulation of the "computer

vision problem." Also, and to an even larger extent, there is no standard formulation of how computer vision problems should be solved. Instead, there exists an abundance of methods for solving various well-defined computer vision tasks, where the methods often are very task specific and seldom can be generalized over a wide range of applications. Many of the methods and applications are still in the state of basic research, but more and more methods have found their way into commercial products, where they often constitute a part of a larger system which can solve complex tasks (e.g., in the area of medical images, or quality control and measurements in industrial processes).

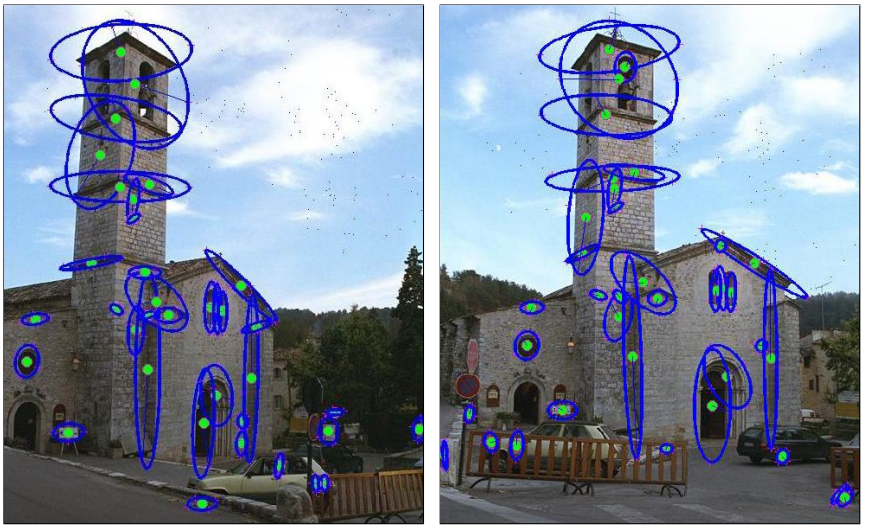
One of the most prominent application fields is medical computer vision or medical image processing. This area is characterized by the extraction of information from image data for the purpose of making a medical diagnosis of a patient. Typically image data is in the form of microscopy images, X-ray images, angiography images, ultrasonic images, and tomography images. An example of information which can be extracted from such image data is detection of tumours, arteriosclerosis or other malign changes. It can also be measurements of organ dimensions, blood flow, etc. This application area also supports medical research by providing new information (e.g., about the structure of the brain, or about the quality of medical treatments).

A second application area in computer vision is in industry. Here, information is extracted for the purpose of supporting a manufacturing process. One example is quality control where details or final products are being automatically inspected in order to find defects. Another example is measurement of position and orientation of details to be picked up by a robot arm.

Military applications are probably one of the largest areas for computer vision, even though only a small part of this work is open to the public. The obvious examples are detection of enemy soldiers or vehicles and guidance of missiles to a designated target. More advanced systems for missile guidance send the missile to an area rather than a specific target, and target selection is made when the missile reaches the area based on locally acquired image data. Modern military concepts, such as "battlefield awareness," imply that various sensors, including image sensors, provide a rich set of information about a combat scene which can be used to support strategic decisions. In this case, automatic processing of the data is used to reduce complexity and to fuse information from multiple sensors to increase reliability.

Computer vision is by some seen as a subfield of artificial intelligence where image data is being fed into a system as an alternative to text based input for controlling the behaviour of a system. Some of the learning methods which are used in computer vision are based on learning techniques developed within artificial intelligence.

A second field which plays an important role is neurobiology, specifically the



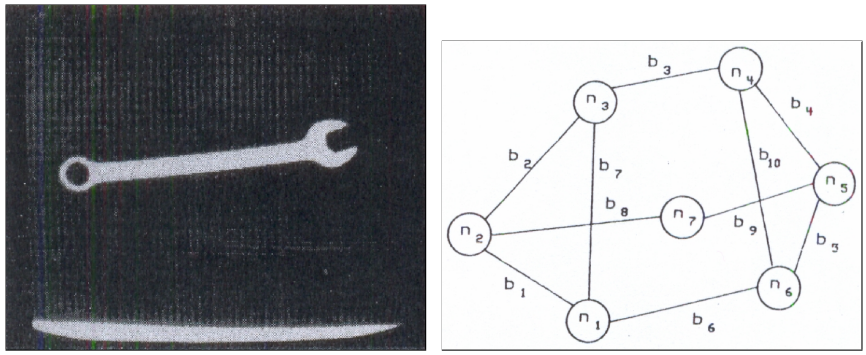
(a) Known scene as it exists in database. (b) Unknown scene to be recognized.

Figure 1.2: Scene recognition by local features using nearest neighbour methods. All detected features are marked in both images. This example is a reprint from [2].

study of the biological vision system. Over the last century, there has been an extensive study of eyes, neurons, and the brain structures devoted to processing of visual stimuli in both humans and various animals. This has led to a coarse, yet complicated, description of how "real" vision systems operate in order to solve certain vision related tasks. These results have led to a subfield within computer vision where artificial systems are designed to mimic the processing and behaviour of biological systems, at different levels of complexity.

Yet another field related to computer vision is signal processing. Many existing methods for processing of one-variable signals, typically temporal signals, can be extended in a natural way to processing of two-variable signals or multi-variable signals in computer vision. However, because of the specific nature of images there are many methods developed within computer vision which have no counterpart in the processing of one-variable signals. A distinct character of these methods is the fact that they are non-linear which, together with the multi-dimensionality of the signal, defines a subfield in signal processing as a part of computer vision.

Many of the related research topics can also be studied from a purely mathematical point of view. For example, many methods in computer vision are based on statistics, optimization or geometry. Pattern recognition uses various methods to extract information from signals in general, mainly based on statistical



(a) An image of a model object

(b) Attributed relation graph representation for the single-object image (a).

Figure 1.3: This example is a reprint from [3].

approaches, and a significant part of this research is devoted to applying these methods to image data.

Pattern recognition aims to classify data (patterns) based on either a priori knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space.

A complete pattern recognition system consists of a sensor that gathers the observations to be classified or described; a feature extraction mechanism that computes numeric or symbolic information from the observations; and a classification or description scheme that does the actual job of classifying or describing observations, relying on the extracted features.

The classification or description scheme is usually based on the availability of a set of patterns that have already been classified or described. This set of patterns is termed the training set and the resulting learning strategy is characterised as supervised learning. Learning can also be unsupervised, in the sense that the system is not given an a priori labelling of patterns, instead it establishes the classes itself based on the statistical regularities of the patterns.

Consider a classical m-class pattern recognition problem. When we consider each pattern as a single entity we can use a set of n characteristic measurements (features) to represent each pattern under study. In such a case, each pattern is represented by an n-dimensional feature vector and the recognition of patterns can be accomplished by applying various techniques in discriminant analysis and statistical decision theory. Such an approach is often called decision-theoretic or statistical approach. However, when the patterns under study are very complex or when the number of pattern classes m is very large the number of features n

required for recognition could also become very large. Consequently, the classical decision-theoretic approach often becomes ineffective or computational infeasible in solving this kind of problems.

One way to approach this kind of problem is to represent a complex pattern by its simpler subpatterns and hope that we can treat each simpler subpattern as a single entity and use decision-theoretic methods for the subpatterns (see [11] and [12]). Of course, the relations among subpatterns must be taken into consideration. If each subpattern is again very complex, we may represent each subpattern by even simpler subpatterns, until we are sure that the simplest subpatterns, named "pattern primitives" by [13], can be easily treated by simple decision-theoretic methods.

In [10] the authors illustrate how weighted graphs can be used to formulate a structural description of an object. Such descriptions have been further enhanced with parametric information and represented by attributed relation graphs (ARG, see [3]). Such graphs are capable of describing and relating the pattern primitives in an elegant and efficient structure. See figure 1.3 as an example on how graphs can be used to express the content of an image. Graphs matching is traditionally used for man made objects that contain sharp edges, basic colors and controlled lighting since these are properties that allow for less ambigous graphs.

In the following sections we examine how such graphs are used in computer vision, and what problems they pose.

1.3 Vision by inexact graph matching

Let us first introduce the concept of vision by inexact graph matching.

A database contains a set $\{M_k\}$ of models. Each model M_k is defined by the graphs $\{G_j\}$ extracted from model images. During the storage process, each graph G_j is added to the database with a link to the model k for which it has been extracted. Formally, the simplest database is a table of pairs (G_j, k).

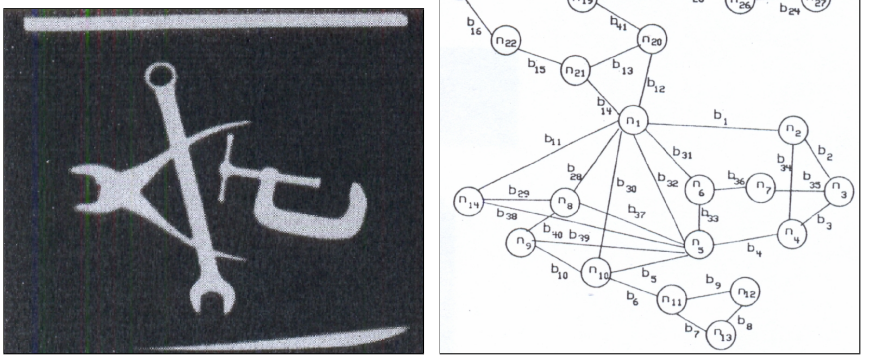
Recognition consists of finding the model M_k which corresponds to a given query image I; that is the model which is the most similar to this image. A graph G_l is extracted from the image, then compared to every graph G_j in the database by computing their "distance" by a function $d(G_l, G_j)$.

The model associated with the "closest" graph is then selected as the best match. The dictionary definition (see [14]) of the verb "match" is to "1; agree almost exactly. 2; be comparable or equivalent in character or form." When one speak of two objects matching, it is often assumed that matching is a symmetric process; A matches B if and only if B matches A. The use of the term "inexact matching" is to stress the asymmetric nature of our method; we are looking for the model in the database that is the least different from the unknown one.

In [10], the authors formulate a structural description D of an object as a pair $D = (P, R)$. $P = \{P1, \cdots, Pn\}$ is a set of primitives, one for each of the n primitive parts of the object. Each primitive Pi is a binary relation $Pi \subseteq A \times V$ where A is a set of possible attributes and V is a set of possible values. $R = \{PR1, \cdots, PRK\}$ is a set of named N-ary relations over P. For each $k = 1, \cdots, K$. PRk is a pair (NRk, Rk) where NRk is a name for relation Rk, and for some positive integer Mk, $Rk \subseteq P^{Mk}$. Thus, set P represents the parts of an object, and set R represents the interrelationships among the parts. Note that the elements of any relation Rk may include as components primitives, attributes, values and any symbols necessary to specify the given relationship.

Through [3] and [15] this description is realized as an attributed relation graph (ARG). This ARG is defined as a 6-tuple $G = (V, E, A_V, A_E, \alpha_V, \alpha_E)$ where V and E are respectively the sets of the vertices and the edges of the ARG; A_V and A_E are the sets of vertex- and edge-attributes, while α_V and α_E the functions associating to each vertex or edge the corresponding attributes.

The attributes of a node or an edge have the form $t(p_1, ..., P_{k_t})$, where t is a type chosen over a finite alphabet T, and $(p_1, ..., p_{k_t})$ are a tuple of parameters, also from finite sets $P'_1, ..., P'_{k_t}$. Both the number of parameters, k_t, and the sets they belong to depend on the type of the attribute, and for some type k_t may be equal to 0, i.e. the attribute has no parameters. The type information is used to discriminate among different kinds of nodes (or edges), while the parameters carry the information which characterizes the nodes (or edges) of a given type. Usually



(a) An image of overlapping objects.

(b) Attributed relation graph representation for the multi-object image (a).

Figure 1.4: An example of a complex scene that can be matched against the ARG in figure 1.3 to identify the wrench. This example is a reprint from [3].

the nodes of the graphs are used to represent the primitives of the structural descriptions, and the edges to represent the relations between the primitives; but the availability of different node or edge types allows us to use ARG's as very general descriptors.

The database of models and the distance measurement above hold equally well under these descriptors as it holds for simpler graphs.

For readability we will express ARG's only by its vertices and edges, such as $G = (V, E)$. The attribute alphabet, values and associations are implicitly given by the context in which these graphs appear. If needed for clarity, these elements will be referenced using the above notation.

One way the structural descriptions are used is to define "prototype" models. The structural descriptions of prototype models are used as part of the knowledge base of the recognition system. Such a system inputs "candidate" images, computes their structural descriptions, and tries to identify each candidate with a stored model. Thus, instead of asking whether two structural descriptions match each other, we only ask whether a candidate structural description matches a prototype structural description.

However, the methods for the automatic generation of the prototypes from a suitable set of ARG's standing as examples are very complex.

There are mainly three different approaches to the problem.

The first approach relies on the assumption that an expert of the domain defines by hand the prototypical descriptions of the classes. This approach cannot neglect the problem of the definition of a suitable deformation model allowing the recognition of noisy and distorted samples. To this concern some authors (see [16]) define by hand the deformation model, while others (see [17]) use a training set for tuning its parameters. The main advantage of this approach is that it does not require a large training set. On the other hand, the inadequacy of human knowledge to find, for each class, a set of prototypes really representative of a given class, significantly increase the risk of confusion among different classes, especially in noisy domains.

The second approach is based on the use of a large set of patterns, possibly containing a large variety of distorted samples; the advocates of this approach say that any pattern may be recognized if enough data is collected (see [18]). The main advantage of this approach is that no a priori knowledge is required, and the classification system is relatively simple. On the other hand, these systems usually exhibit a limited capability of recognizing a pattern different from those in the reference set, making their use critical when data are not sufficient to build a reference set containing all the variants which may occur in a class.

The third approach (see [19], [15]) considers the determination of the class prototypes as a symbolic machine learning problem, formulated as follows: given a suitably chosen set of input data, and possibly some background domain knowledge, the goal is to discover a description of each class which is more general than the bare enumeration of the training samples, but still retains the ability of discerning objects belonging to different classes. One of the advantages of this approach is that the descriptions obtained are usually explicit and easily interpretable by humans, so allowing an expert to validate or to improve them, or to achieve a better understanding of what has gone wrong in case of errors. While this approach is promising and appealing for the extreme generality of the found prototypes, its major drawback is the computational cost, which is fairly high so limiting the applicability in complex applications.

In [20] a pattern deformational model is proposed. A class of structure-preserving deformations, i.e., deformations which change only the values of the attributes of nodes and branches of the ARG is defined. The corresponding ARG matching algorithm, called "Error-Correcting Isomorphism," is consequently based on a graph isomorphism for the syntactic part of the ARG. This algorithm, though powerful enough for some practical applications, reveals to be not adequate when the object to be recognized is partially occluded, or greatly distorted in some of its parts. In [21], a generalization of [20] is proposed that includes the possibility of deletion of nodes and branches. However, when severe variations among the members of the same class are possible, even this last model is not effective.

These ideas are extended in [22] where the authors propose an ARG inexact matching algorithm. The inexactness implies that the sample graph is consid-

ered matchable with one of the prototypes if a set of syntactic and semantic transformations (selected among those applicable to the considered prototype) can be found. such that the transformed graph sample is isomorphic to the graph of the prototype.

Our method belongs to the second approach above, supported by the claims in [18] (any pattern may be recognized if enough data is collected). Our knowledge base consists of a large variety of samples for each model, and we have no need to find the most general prototype for each model. Our decisions are made simply on the grounds of which of the stored descriptors that most resembles the candidate one. This means that our method requires no preprocessing of the database, there is no search for a partitioning of a complex hyperspace. This has the added benefit of allowing new models to be added to the knowledgebase in real time, thus allowing learning systems to use our method with ease.

The complexity of our method is given solely by the complexity of the subgraph isomorphism problem, a problem that other methods, such as [22], contain as subproblems. Let us in the next section consider the difficulty of this problem.

1.4 The complexity of inexact graph matching

In this section we introduce the complexity of the inexact matching of two graphs. We briefly discuss the theory of NP-completeness and how this applies to our problem. We comment on other research that expresses the complexity of similar problems, and conclude with a formal description of our problem and its intractability.

Let us say that a function $f(n)$ is $O(g(n))$ whenever there exists a constant c such that $|f(n)| \leq c \cdot |g(n)|$ for all values of $n \geq 0$. A polynomial time algorithm is defined to be one whose time complexity function is $O(p(n))$ for some polynomial function p, where n is used to denote the input length. Any algorithm whose time complexity function cannot be so bounded is called an exponential time algorithm. We shall refer to a problem as intractable if it is so hard that no polynomial time algorithm can possibly solve it.

The foundations for the theory of NP-completeness were laid in [23].

The paper emphasizes the significance of "polynomial time reducibility," that is, reductions for which the required transformation can be executed by a polynomial time algorithm. If there is a polynomial time reduction from one problem to another, then any polynomial time algorithm for the second problem can be converted into a corresponding polynomial time algorithm for the first problem.

It also focuses attention on the class NP of decision problems, problems whose solution is either "yes" or "no", that can be solved in polynomial time by a nondeterministic computer. Most of the apparently intractable problems encountered in practice, when phrased as decision problems, belong to this class.

The author of [23] proves that one particular problem in NP, called the "satisfiability" problem, has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability problem can be solved with a polynomial time algorithm, then so can every other problem in NP, and if any problem in NP is intractable, then the satisfiability problem also must be intractable. The satisfiability problem is thus the "hardest" problem in NP. It is also suggested that other problems in NP might share with the satisfiability problem this property of being the "hardest" member of NP. Subsequently, Richard Karp presented a collection of results (see [24]) proving that indeed the decision problem versions of many well known combinatorial problems, including the traveling salesman problem, are just as "hard" as the satisfiability problem. Since then a wide variety of other problems have been proved equivalent in difficulty to these problems, and this equivalence class has been labeled the class of "NP-complete problems". As a matter of convenience, the theory of NP-completeness is designed to be applied only to decision problems. Abstractly, a decision problem Π consists simply of a set D_Π of instances and a subset $Y_\Pi \subseteq D_\Pi$ of yes-instances. However, most decision problems of interest possess a considerable amount of additional structure,

and we adopt the format of [25] to emphasizes this structure. The format consists of two parts, the first part specifying a generic instance of the problem in terms of various components, which are sets, graphs, functions, numbers, etc., and the second part stating a yes-no question asked in terms of the generic instance. The way in which this specifies D_Π and Y_Π should be apparent. An instance belongs to D_Π if and only if it can be obtained from the generic instance by substituting particular objects of the specified types for all the generic components, and the instance belongs to Y_Π if and only if the answer for the stated question, when particularized to that instance, is "yes."

Among the 21 original NP-complete problems is CLIQUE, given as:

CLIQUE
INSTANCE: A graph $G = (V, E)$ and a positive integer $J \leq |V|$.
QUESTION: Does G contain a clique of size J or more, that is, a subset $V' \subseteq V$ such that $|V'| \geq J$ and every two vertices in V' are joined by an edge in E?

The principle technique used for demonstrating that two problems are related is that of "reducing" (denoted by "$\propto$") one to the other, by giving a constructive transformation that maps any instance of the first problem into an equivalent instance of the second. Such a transformation provides the means for converting any algorithm that solves the second problem into a corresponding algorithm for solving the first problem.

Transformed from CLIQUE by [23] the problem of SUBGRAPH ISOMORPHISM has been proven to lie in NPC:

SUBGRAPH ISOMORPHISM
INSTANCE: Graphs $G = (V_1, E_1)$, $H = (V_2, E_2)$.
QUESTION: Does G contain a subgraph isomorphic to H, i.e., a subset $V \subseteq V_1$ and a subset $E \subseteq E_1$ such that $|V| = |V_2|$, $|E| = |E_2|$, and there exists a one-to-one function $f : V_2 \rightarrow V$ satisfying $\{u, v\} \in E_2$ if and only if $\{f(u), f(v)\} \in E$?

Let this be a first approximation to our problem. As such, it is NP-complete.

There is no known polynomial time algorithm for solving this problem. However, suppose someone claimed, for a particular instance of this problem, that the answer is "yes." If we were sceptical about this claim, we could demand that they supplied us with a subgraph isomorphism from graph H to graph G. It would then be a simple matter to verify the truth or falsity of their claim by applying this isomorphism to map the vertices in H to those in G, and then comparing the implied edge-associations. This verification procedure could be

specified as a general algorithm that has time complexity bound polynomially by the size of Π.It is this notion of polynomial time "verifiability" that the class NP is intended to isolate. Notice that polynomial time verifiability does not imply polynomial time solvability. A problem that can not be solved in polynomial time by a nondeterministic computer means that even checking a solution for validity is an intractable problem. Such a problem lies outside NP, and is therefore more difficult than any NP-complete problem.Any decision problem Π, whether a member of NP or not, to which we can transform an NP-complete problem will have the property that it cannot be solved in polynomial time unless P=NP. We might say that such a problem Π is "NP-hard," since it is, in a sense, at least as hard as the NP-complete problems. Any decision problem Π which we can transform to some problem in NP, will have the property that it can be solved in polynomial time if P=NP and hence can be "no harder" than the NP-complete problems. This type of problem Π is "NP-easy," since it is just as easy as the hardest problems in NP. A standard example of a problem that is NP-hard and that may not be as easy as the NP-complete problems is the following from [26] and [27].

MINIMUM EQUIVALENT EXPRESSION
INSTANCE: A well-formed Boolean expression E involving literals on a set V of variables, the constants T (true) and F (false), and the logical connectives $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\rightarrow$ (implies), and a nonnegative integer K.
QUESTION: Is there a well-formed Boolean expression E' that contains K or fewer occurrences of literals such that E' is equivalent to E, that is, such that for all truth assignments to V the truth values of E' and E agree?

This problem is NP-hard since SATISFIABILITY is reducible to it. However, no one has been able to show that an oracle for SATISFIABILITY (or for any problem in NP) would enable us to solve it in polynomial time, therefore it is not NP-easy.

Enumeration problems provide natural candidates for the type of problem that might be intractable even if P=NP. In a search problem Π, each instance $I \in D_\Pi$ has an associated solution set $S_\Pi(I)$, and, given I, we are required to find one element of $S_\Pi(I)$ (the corresponding decision problem asks whether or not $S_\Pi(I)$ is empty). The enumeration problem based on the search problem Π is "Given I, what is the cardinality of $S_\Pi(I)$, that is, how many solutions are there?"

The enumeration problems associate with NP-complete problems are clearly NP-hard, since if we know the cardinality of $S_\Pi(I)$ we can easily tell whether or not $S_\Pi(I)$ is empty. Even if P=NP, and we could tell in polynomial time whether an arbitrary graph contains a Hamiltonian circuit, it is not apparent that this

would enable us to count how many Hamiltonian circuits are contained in G in polynomial time.

Consider the following problem: Given a bipartite graph G, how many distinct perfect matching does it contain? (Recall that a bipartite graph $G = (V, E)$ is one in which the vertex set V is partitioned into two sets V_1 and V_2, and no edge has both endpoints in the same set. A perfect matching is a set of edges $E' \subseteq E$ such that every vertex in V is included in exactly one edge in E'.) The underlying search problem is well known, since it is just the "marriage problem", which can be solved in polynomial time. Nevertheless, in [28] the enumeration problem is proved to be complete in a class of problems outside NP (namely #P-complete).

Let us now return to our problem and show how its complexity appears to lie outside NP. In [8] the authors define "the weighted graph matching problem" as follows: Given two undirected graphs G and g which may be sparse and whose edges may take values in R^1, find the match matrix M such that the following objective function is minimized.

$$E_{wg}(M) = -\frac{1}{2}\sum_{a=1}^{A}\sum_{i=1}^{I}\sum_{b=1}^{A}\sum_{j=1}^{I} M_{ai}M_{bj}C_{aibj} \tag{1.1}$$

subject to $\forall a \sum_{i=1}^{I} M_{ai} \leq 1$, $\forall i \sum_{a=1}^{A} M_{ai} \leq 1$, $\forall ai M_{ai} \in 0, 1$.

Graphs G and g have A and I vertices respectively. $\{C_{aibj}\}$ is defined by:

$$C_{aibj} = \begin{cases} 0, & \text{if either } G_{ab} \text{ or } g_{ij} \text{ is NULL;} \\ c(G_{ab}, g_{ij}) & \text{otherwise.} \end{cases} \tag{1.2}$$

$\{G_{ab}\}$ and $\{g_{ij}\}$ are the adjacency matrices of the graphs, whose elements may be in R^1 or NULL. These matrices are symmetric with NULL elements along the diagonal (because the graphs are non-reflexive). So, G_{ab} is the weight of the edge between vertices a and b of graph G. The matrix M indicates which vertices in the two graphs match:

$$M_{ai} = \begin{cases} 1, & \text{if node } a \text{ in } G \text{ corresponds to vertex } i \text{ in } g; \\ 0, & \text{otherwise.} \end{cases} \tag{1.3}$$

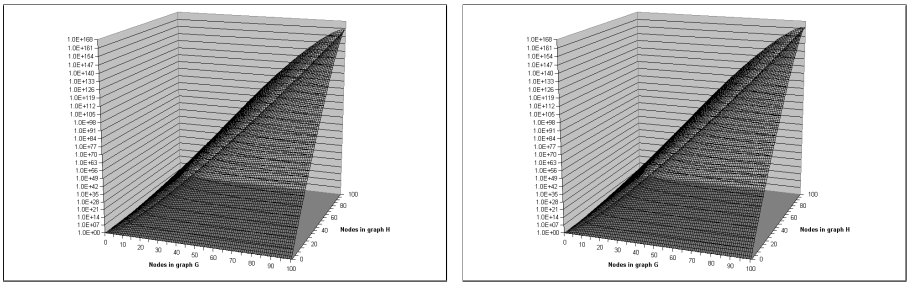
The function $c(\cdot, \cdot)$ is chosen as a measure of compatibility between the edges of the two graphs. This function is similar to the compatibility functions used within the relaxation labeling framework in [29], [30] and [31]. By explicitly defining C to be 0 when an edges is missing we are ensuring that C will also be sparse when the graphs are sparse.

The authors of [8] claim that "the weighted graph matching problem" is NP-complete since it contains the problem LARGEST COMMON SUBGRAPH as a special case. By their reference, that problem is formulated as:

> **LARGEST COMMON SUBGRAPH**
> INSTANCE: Graphs $G = (V_1, E_1)$, $H = (V_2, E_2)$, positive integer K.
> QUESTION: Do there exist subsets $E_1' \subseteq E_1$ and $E_2' \subseteq E_2$ with $|E_1'| = |E_2'| \geq K$ such that the two subgraphs $G' = (V_1, E_1')$ and $H' = (V_2, E_2')$ are isomorphic?

This is a problem proven to lie in NPC by transformation from CLIQUE by [25]. It should be noticed, however, that the authors of [8] seemto have misinterpreted this as a problem of finding the largest possible K for any two graphs. A decision problem, such as LARGEST COMMON SUBGRAPH, can be derived from an optimization problem, such as "the weighted graph matching problem". If the optimization problem asks for a structure of a certain type that has a minimum "cost" among all such structures, we can associate with that problem thedecision problem that includes a numerical bound B as an additional parameter and that asks whether there exists a structure of the required type having costno more than B. Decision problems can be derived from maximization in an analogous way, simply by replacing "no more than" by "at least."The key point to observe about this correspondence is that, so long as the cost function is relatively easy to evaluate, the decision problem can be no harder than the corresponding optimization problem. Clearly, if we could find a minimum cost structure in polynomial time, then we could also solve the associateddecision problem in polynomial time. All we need to do is find the minimum cost structure, compute its cost, and compare that cost to the given bound B. Thus,if we could demonstrate that the decision problem is NP-complete (as indeed it is), we would know that the optimization problem is at least as hard.Even if LARGEST COMMON SUBGRAPH is NP-complete, the optimization problem need only be NP-hard.Let us now reformulate "the weighted graph matching problem" as the optimization problem it in fact is. Instead of searching for a common subgraph, we require all vertices of the smaller graph be associated with vertices in the larger. The cost to be minimized is expressed by the functions $Ev(v_i, v_j)$and $Ee(e_k, e_l)$, where edge-associations are implied by the vertex-associations. Associating vertices v_1 and v_2 in graph G to vertices v_3 and v_3 in graph H respectively, implies that an edge (v_1, v_2) in graph G is associated with the edge (v_3, v_4) in graph H.

> **MINIMUM COST SUBGRAPH ISOMORPHISM**
> INSTANCE: Given graphs $G = (V_1, E_1)$, $H = (V_2, E_2)$ where $|V_1| \leq |V_2|$, a vertex cost metric $Ev(v_i, v_j)$ for associating a vertex $v_i \in V_1$ to a vertex $v_j \in V_2$, and an edge cost metric $Ee(e_k, e_l)$ for associating an edge $e_k \in E_1$ to an edge $e_l \in E_2$.



(a) α component of upper bound. (b) β component of upper bound.

Figure 1.5: Upper bound for MINIMUM COST SUBGRAPH ISOMORPHISM.

> QUESTION: Under these metrics, what is the minimum cost subgraph isomorphism from graph G to graph H?

This problem does not simply require the cardinality of $S_\Pi(I)$, but it requires an evaluation of each $Y_\Pi \in S_\Pi(I)$ under the metrics $Ev(v_i, v_j)$ and $Ee(e_k, e_l)$. The problem is NP-hard because the derivable decision problem is SUBGRAPH ISOMORPHISM, and it is a candidate for the #P-complete class of [28] by the required enumeration.

Methods such as [32], [22], [8], and [15] present methods to find approximate solutions to variations of this problem. Those methods all rely on the connectivity, or rather the lack thereof, of the graphs G and H in the problem instances. They use missing connectivity as a heuristic for associating vertices from one graph to vertices in the other. These methods perform well under the demonstrated conditions, but by complexity bound to $|E_1|$ and $|E_2|$ they quickly degenerate as the connectivity of either graph grows.

In the next section we argue that every relation graph in computer vision should, in fact, be fully connected. Let us, therefore, discuss what impact such graphs have. By connectivity alone they constitute the worst-case problem instances. If the graphs are also directed and reflective, they contain $|V_1|^2$ and $|V_2|^2$ edges respectively. There are no "missing" edges in these graphs, so the previously mentioned methods have no syntactic information that they can use to prune unlikely vertex-associations, and they are forced to try every single association.

The cardinality of the set containing every possible subgraph isomorphism from graph G to graph H is:

$$\frac{|V_2|!}{(|V_2| - |V_1|)!} \tag{1.4}$$

For each of these isomorphisms it is necessary to evaluate the associations they

contain. Let us define α to be the time required for the function $Ev(v_i, v_j)$ to evaluate one vertex-association, and β to be the time required for the function $Ee(e_k, e_l)$ to evaluate one edge-association. Because an isomorphism associates every vertex in the smaller graph G to some vertex in the larger graph H, an isomorphism implicitly also associates every edge in graph G to some edge in graph H. The number of vertex-associations in an isomorphism therefore equals the number of vertices in graph G, and likewise the number of edge-associations equals the number of edges in graph G. Evaluation of a single isomorphism thus requires $(\alpha|V_1| + \beta|E_1|)$ time, which can be expressed by the number of vertices in the smaller graph alone; $(\alpha|V_1| + \beta|V_1|^2)$.

This gives us an upper bound on the time required to evaluate every possible subgraph isomorphism from graph G to graph H, by the equation:

$$(\alpha|V_1| + \beta|V_1|^2) \cdot \frac{|V_2|!}{(|V_2| - |V_1|)!} \tag{1.5}$$

This bound on MINIMUM COST SUBGRAPH ISOMORPHISM is shown in figures 1.5(a) and 1.5(b). Since there is no necessary relation between the values of α and β, they are drawn in separate figures. Although the β component is larger than the α component by a magnitude of $|V_1|$, this is not readily available from the figures since these are drawn using a logarithmic vertical axis. The logarithmic scale is needed, however, to contain the exponential growth of both components.

Having underlined the inherent difficulty of the problem at hand, let us in the next sections discuss our problem instances and propose an approximate solution to these.

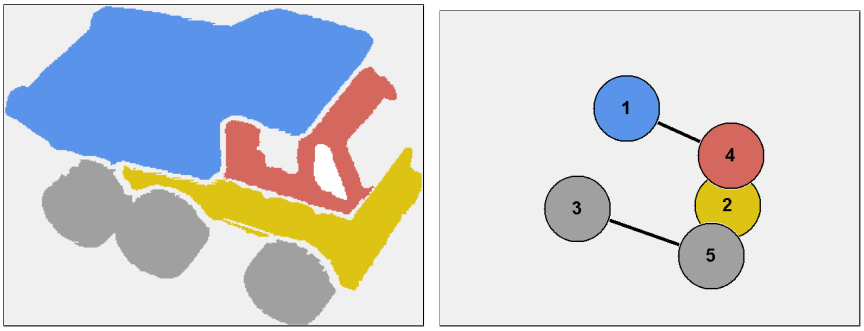
1.5 Problem instances in computer vision

In this section we present our first contribution; we demonstrate that all problem instances of MINIMUM COST SUBGRAPH ISOMORPHISM in computer vision contain graphs G and H that are fully connected. Traditional approximation methods are therefore unsuitable to solve these problems. This achieves milestone 1; investigating the trends in graph extraction research.

Observe the image in figure 1.6(a). By context, the reader will undoubtedly relate the shapes in the image by their spatial layout and apparent clustering or cliques. A commonly extracted graph is shown in figure 1.6(b). Such extraction is intuitive, and it has been thoroughly strengthened by examples in pattern recognition literature such as [33], [32], [10], [13], [3], [8].

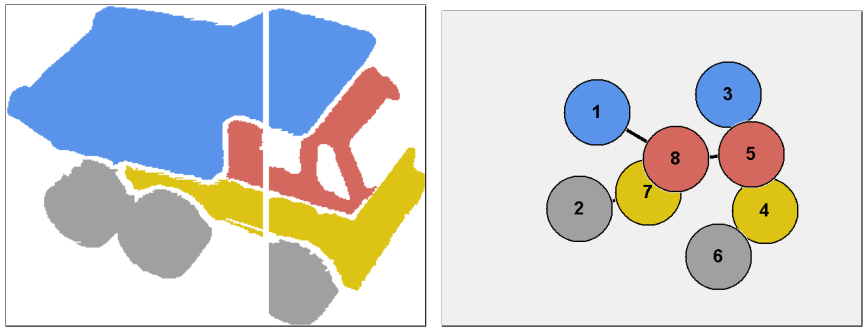
There is, however, an inherent problem with this extraction. Let us illustrate this by introducing noise to the image in figure 1.6(a). The simple distortion illustrated in figure 1.7(a) produces the dissimilar graph in figure 1.7(b) by applying the same scheme for connectivity. It is quite possible to counter noise and distortions by preprocessing the image using applicable filters (see [6]), but this does not necessarily solve the problems altogether. As the size of the problem instance increases, the distinguishing between random noise and image elements becomes more difficult.

Any labeling or connectivity scheme can be broken by introducing tailored noise, therefore, these schemes should not be considered reliable for ordering vertices or edges. See how labeling vertices by order of encounter in left-to-right, top-to-bottom scan is broken by distortion in figures 1.7. See also how nearest-neighbour connectivity is broken by the same distortion. Note that the demonstrated distortion it atypical (caused by a faulty camera or by object occlusion), it serves simply to illustrate the fragile nature of the connectivity schemes.



(a) (b)

Figure 1.6: Undistorted image and corresponding graph.



(a) (b)

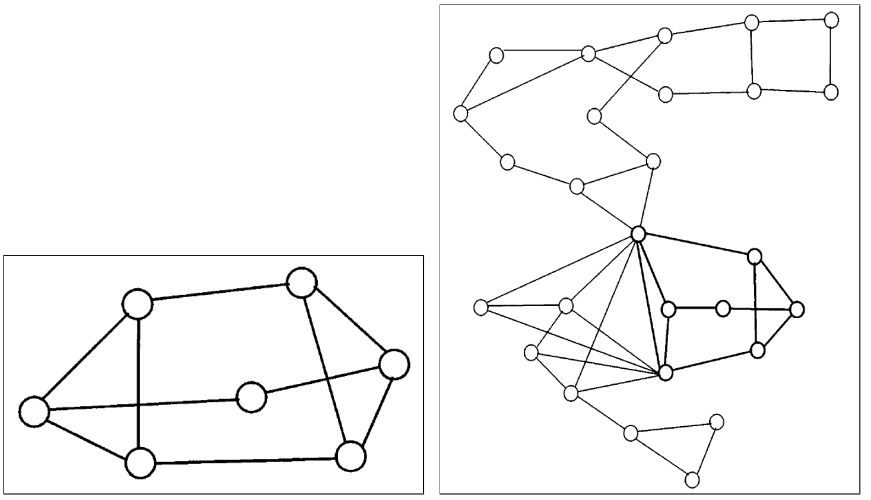
Figure 1.7: Distorted image and corresponding graph.

In [20], a pattern-deformation model is proposed, while a generalization of the method, including the possibility of deleting nodes and branches, is discussed in [21]. The algorithm, though powerful enough for some practical applications, is not effective when large variations among the members of a class may exist. This is analogous to the discussion of noise above.

Two decades after the publication of [20], [22] proposes an ARG inexact matching algorithm; the inexactness implies that the candidate graph is considered matchable with one of the prototypes if a set of syntactic and semantic transformations (selected among those applicable to the prototype) can be found such that the transformed graph candidate is isomorphic to the graph of the prototype. As will become evident throughout this section, because of its dependency on syntactic information, it will degenerate as the connectivity of the graphs increases.

In one experiment outlined in [3], attributed relation graph matching was used to locate an object within a multiobject scene. ARG's were produced from real images using a multilayer graph transducer scheme. An ARG produced from an image of a wrench (see figure 1.3) was matched against an ARG produced from an image of a number of overlapping objects which included the wrench (see figure 1.8(a). The multiple attributes on the nodes were line segment length, arc segment length, arc segment span and contour length. The multiple link types were joint, intersection and facing features. The nodes in figure 1.8(b) that match to figure 1.8(a) have been highlighted.

The graphs in figures 1.3 and 1.4 are clearly sparse, a characteristic that has carried through graph matching research for several decades by publications such as [33], [32], [10], [13], [3], [34] and [8]. As illustrated in the previous section on complexity, the subgraph isomorphism problem becomes progressively more difficult as the connectivity of the graphs increases. This unyielding increase of complexity may be the reason why researchers have used sparser graphs for



(a) Graphical representation of a wrench.

(b) Graphical representation of a scene with a number of overlapping machine parts.

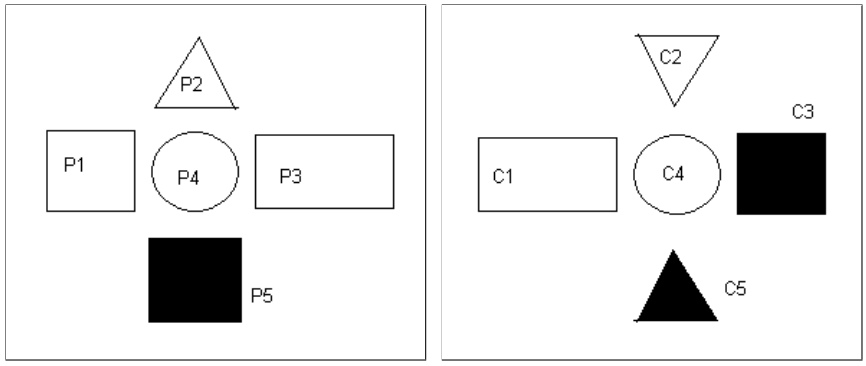
Figure 1.8: Graphs as they appear in [3].

demonstrating their methods. Consequently, the information carried by connectivity has not enjoyed the attention it deserves.

In section 1.3, we introduce the notions of "pattern primitives," as in [13]. In the very idea of building complex patterns by combining these pattern primitives lies the notion of relating these patterns to each other. These pattern primitives, features, feature points, or whatever name is used to label them, all exist at some spatial position in the image from which they are being extracted. Regardless of the shape or spatial extent in which these primitives appear they will individually have a spatial centroid. This holds for any dimensionality of the source image. By this rationale, every pattern primitive will have a spatial relationship to all other pattern primitives in the same image.

Graph matching methods used in computer vision often make no use of these readily available, coherent and pervasive spatial relations among the pattern primitives. It is disconcerting to see how often graphs of primitives have been pruned to make them sparse, when the images from which they are generated are by definition, spatially interrelated.

Please observe again the graphs in figures 1.3 and 1.4. Even though the rendering of vertices corresponds to how the underlying primitives appear in the original image, many vertices remain unconnected to more than a few neighbours. The existence of the wrench is blatantly obvious by connectivity alone. This scheme is repeated in [8] by hand-crafting an ARG from a prototype image, and then



(a) (b)

Figure 1.9: A prototype object (a) and a candidate object (b), both made up of primitive parts.

"discovering" that prototype as a subgraph in another hand-crafted ARG.

Consider the prototype object and candidate object shown in figure 1.9. Given below are a structural Description Dp for the Prototype object and a structural Description Dc for the Candidate object, expressed using the syntax of [10].

$$
\begin{aligned}
Dp &= \{P, RP\} \\
P &= \{P1, P2, P3, P4, P5\} \\
RP &= \{(Left, Left\_P), (Above, Above\_P)\} \\
Left\_P &= \{(P1, P4), (P4, P3)\} \\
Above\_P &= \{(P2, P4), (P4, P5)\} \\
P1 &= \{(shape, rectangular), (colour, white)\} \\
P2 &= \{(shape, triangular)\} \\
P3 &= \{(shape, rectangular)\} \\
P4 &= \{(shape, circular)\} \\
P5 &= \{(colour, black)\}
\end{aligned}
$$

$$
\begin{aligned}
Dc &= \{C, RC\} \\
C &= \{C1, C2, C3, C4, C5\} \\
RC &= \{(Left, Left\_C), (Above, Above\_C)\} \\
Left\_C &= \{(C1, C4), (C4, C3)\} \\
Above\_C &= \{(C2, C4), (C4, C5)\} \\
C1 &= \{(shape, rectangular), (colour, white)\} \\
C2 &= \{(shape, triangular), (colour, white)\} \\
C3 &= \{(shape, rectangular), (colour, black)\} \\
C4 &= \{(shape, circular)\} \\
C5 &= \{(shape, triangular), (colour, black)\}
\end{aligned}
$$

Relational attributes such as $Left$ and $Above$ in these descriptions have dominated the graph matching community in computer vision. Problems arise when one is to evaluate the compatibility of such attributes. Since there is no numerical value associated with the attributes, one is left with a binary compatibility – two edges either agree or disagree. The expression of compatibility becomes even more difficult as the conceptual content of these keyword- attributes increase; in [34] we see relations $Above\_Left$, $Above\_Right$, $Touches$ and others.

Four years after introducing those advanced relations, the same authors propose in [35] to express the spatial relationship between pattern primitives using a single character hexadecimal direction code in the range $[0..f]$. This attribute allows for a compatibility function that is able to discern similarity by numerical difference. There is no reasoning on this seemingly arbitrary decision on resolution, and these attributes still do not express the spatial distance of the underlying image.

We propose the use of an n-dimensional vector $\vec{d}$ as edge-attribute, where n is the number of spatial dimensions in the original image and thus the dimensionality of the spatial centroids of the pattern primitives. Although we comment on higher orders of spatial dimensionality, our work is based exclusively on experiments that process 2-dimensional images. To attribute an edge e that connects vertex v_1 to v_2, simply assign to it the vector $\vec{d} = \vec{c_2} - \vec{c_1}$, where $\vec{c_1}$ and $\vec{c_2}$ are the centroids of vertex v_1 and v_2 respectively.

An astonishing characteristic of graphs that represent the spatial relationship of its vertices by such attributed edges, is that the spatial content can be coherently and completely expressed using any degree of connectivity, as long as there are no disconnected vertices.

To demonstrate this we need some formal definitions.

For a graph $G = (V, E)$ with vertex set V and edge set E, a simple circuit in G is

a sequence $< v_1, v_2, ..., v_k >$ of distinct vertices from V such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$ and such that $(v_k, v_1) \in E$. A Hamiltonian circuit in G is a simple circuit that includes all the vertices of G.

A fully connected graph $H = (V', E')$ is one in which every two vertices in V' are joined by an edge in E'.

Let us now detail the addition of edges to a Hamiltonian circuit $G = (V, E)$ until every two vertices in V are joined by an edge, and G appears as a fully connected graph. Again we stress that the edges of E must be spatially coherent.

Initialize a vector $\vec{c}$ of dimensionality equal to that of the spatial relations of the edges in E. We refer to this vector as the "cursor," since it serves to hold a current spatial position. Traverse the vertices of G in the order they appear in the circuit $< v_1, v_2, ..., v_k >$. For every vertex v_i, associate the current value of the cursor $\vec{c}$ as its spatial centroid. For each edge traversed $(v_i, v_{i+1}) \in E$, add the spatial relational vector of that edge to the cursor $\vec{c}$. When this traversal is completed, every vertex in G has been assigned a spatial centroid. Now add new edges to E, in any order, until every two vertices in V are joined by an edge; using the difference of the vertices' spatial centroid as that edge's spatial relational vector.

This traversal is simplified by application to an Hamiltonian circuit, but it can be easily extended to cover any graph of any connectivity. As long as the edges convey a coherent spatial layout of the vertices, it is simply a matter of traversing the existing edges until all vertices have been associated with a centroid.

Note that this method simultaneously proves that any graph $G = (V, E)$ can have its edges pruned until one finds the smallest $E' \subseteq E$ such that every vertex $v \in V$ belongs to at least one $e \in E'$, and still convey the same spatial information for the pattern described by G.

This fact warrants a reconsideration of how methods that operate on graphs in computer vision should be implemented. The methods of [32], [34], [22], [8], [15], and any method that uses syntactical information of the graphs to guide the match, as applied to these graphs, will fail catastrophically. By reducing connectivity to that of a Hamiltonian circuit, complexity becomes trivial. The syntactic information given by connectivity of each vertex is now non-existent since all vertices have 2 connected edges. Similarly, by increasing connectivity until the graph becomes fully connected, the syntactic information is again lost since every vertex has the same amount of connected edges.

Furthermore, any intermediate degree of connectivity can be achieved by selective pruning of edges from a fully connected graph. It seems apparent to the authors that such subjective observation of the patterns is deliberate; anyone is able to fabricate spatially coherent graphs whose layout best conforms to the method to be demonstrated. By applying method-knowledge to the construction

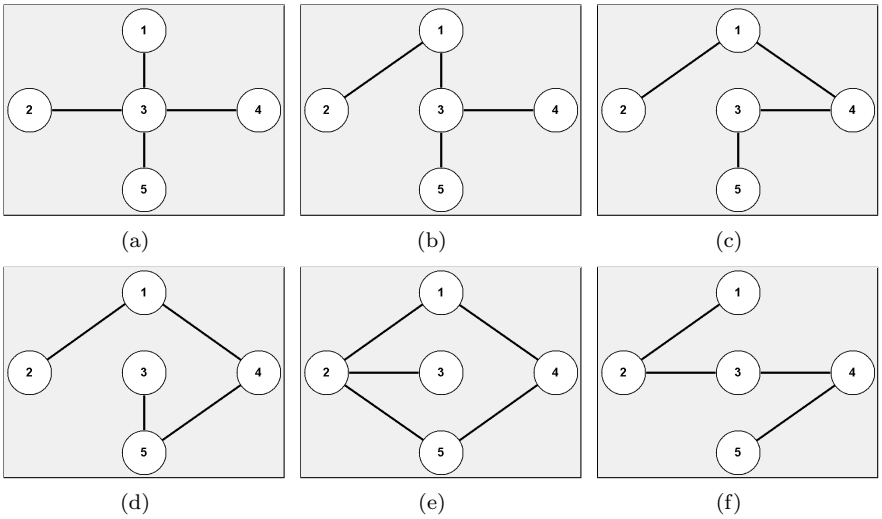


Figure 1.10: Various spatially coherent descriptions of figure 1.9.

of the graph one is effectively projecting a solution into the problem instance. Figure 1.10 demonstrates how different graphs can express the exact content of figure 1.9(a). Similarly, the descriptions *Dp* and *Dc* from [10] given with figure 1.9 clearly illustrates how candidate graphs can be constructed to be obviously compatible with some prototype graph.

By evaluation of the criticized methods on instances of Hamiltonian circuits and cliques, all that effectively remains for these is the stochastic information contained in the attributes of the vertices. The application of the proposed methods conforms to searching for a nearest neighbour in a hyperspace whose dimensionality is given by the pattern primitives.

In [6] the authors describe a feature extractor that is able to produce an attributed relation graph from any 2-dimensional image. The graphs produced are fully connected, similar to ours. Until self-cited in [36], the work has interestingly enough not been commented on in any major publication seen by us. The citation in [36] relates a matching method for model-based computer vision to these graphs. The proposed method is a non-deterministic, suboptimal search that employs time-scheduling at both model- and graph-levels. Because that paper is primarily concerned with scheduling, no direct results are given.

By the realization of these spatial relationships and their imposed requirement of spatial coherency, we propose that all graphs in computer vision should be unreflective, directed, and that there are no disconnected vertices. There should be no reflection because the edges contain the difference of the connected vertices, a

measurement that is meaningless under reflection. The edges should be directed since the spatial difference only has meaning under direction. However, two separate edges connecting the same two vertices are redundant since one edge can be reversed to obtain the other. Finally, there should be no disconnected vertices since that makes the graph spatially incoherent.

The number of edges in any graph $G = (V, E)$, under these constraints, is in the range from $|V| - 1$ to $\frac{1}{2}|V|(|V| - 1)$. This limitation on the number of edges yields a vastly improved upper bound in β for MINIMUM COST SUBGRAPH ISOMORPHISM in chapter 1.4, as shown in figures 1.11(a) and 1.11(b).

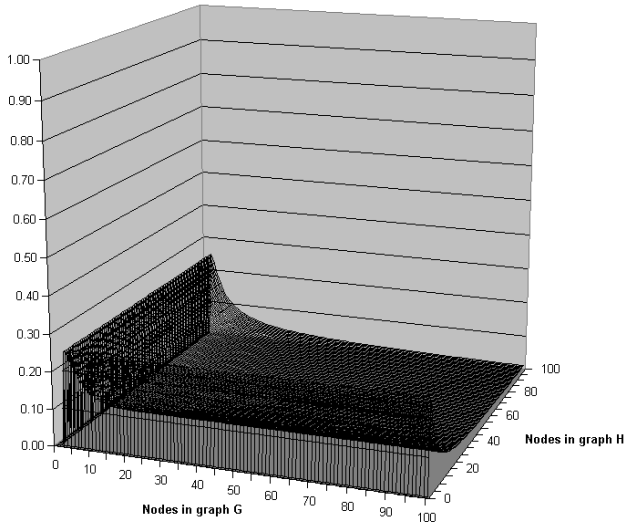
Our method relies on every graph being fully connected; any missing connectivity is derived using the method described above. All recorded runtimes should, as such, be compared to that of figure 1.11(b). However, since the least connected graphs contain the same information as the fully connected ones, we find it more appropriate that our recorded runtimes are compared to that of brute-forcing the smaller graphs. Therefore; all further comments and illustrations relating runtimes in β to that of "brute-force" implies these smaller values.

The improved upper bound on MINIMUM COST SUBGRAPH ISOMORPHISM is thus:

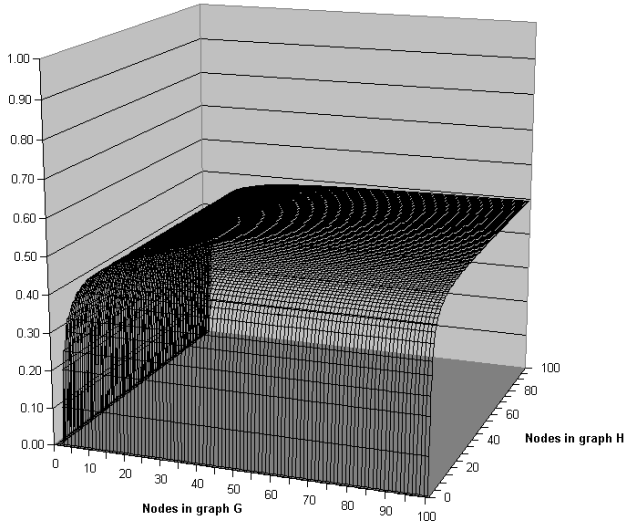
$$(\alpha|V_1| + \beta(|V_1| - 1)) \cdot \frac{|V_2|!}{(|V_2| - |V_1|)!} \tag{1.6}$$

The difference between the α and the β component in this equation, as opposed to equation 1.5, has effectively vanished; the β component has gone from being tremendously larger than the α component to being slightly smaller.

Let us end this section by commenting that the reviewed methods and publications in this chapter retain their viability by not applying them to the attributed relation graphs described here, graphs that capture the very nature of the pattern primitives and their relations. Applied to hand-crafted ARG's ([10], [13], [37], [3], [8], [15]) or random synthetic structures ([8], [38]), such methods have been proven to achieve high accuracy at low computational cost.



(a) Magnitude of β component over corresponding bound in figure 1.5(b) when $|E| = |V|-1$.



(b) Magnitude of β component over corresponding bound in figure 1.5(b) when $|E| = \frac{1}{2}|V|(|V|-1)$.

Figure 1.11: Improved upper bound in β for MINIMUM COST SUBGRAPH ISOMORPHISM.

1.6 Approximations to the optimization problem

This section informally describes our method of solving any instance of MINIMUM COST SUBGRAPH ISOMORPHISM. We demonstrate how fully connected graphs, as derived in the previous section, can be used to strengthen the solution rather than allow it to degenerate the algorithm's runtime.

By the realization that traditional methods to solve subgraph isomorphism in computer vision are reduced to search through stochastic compatibility of nodes in the graphs, let us begin by considering the analogous techniques of nearest neighbour classification and hyperspace partitioning.

In [39] the authors describe shape-indexing as the process to recover from the index the most similar model shapes to a given image shape. In terms of feature vectors, or points in a feature space, this corresponds to finding a set of nearest neighbours (NN) to a query-point. A lot of previous indexing approaches in model-based vision [40], [41], [42], [43], [44], [45], [46] have used hash tables for this task. This is somewhat surprising since it is well-known in other communities (e.q. pattern recognition, algorithms) that tree structures do the job much more efficiently.

In large part this oversight can be explained by the fact that indexing techniques are generally applied in low-dimensional spaces, where hash table search can be quite efficient. Such spaces are adequate when the number of objects is small, but higher-dimensional feature vectors are essential when the model database becomes large, because they provide a much greater degree of discrimination. Unfortunately, nearest-neighbour search times depend exponentially on the dimension of the space.

One data structure that has been used recently for NN lookup is the k-d tree [47]. While the "curse of dimensionality" is also a problem for k-d trees, the effects are not as severe. Hash table inefficiency is mainly due to the fact that bin sizes are fixed, whereas those in a k-d tree are adaptive to the local density of stored points. Thus, in some cases (high-density region), a hashing approach will have to do a long linear search through many points contained in the bin in which the query point lands; in other cases (low-density), an extensive search through adjacent bins may be required before the best nearest neighbour can be determined. In addition, there is the difficulty of choosing an appropriate bin size.

Both [39] and [7] demonstrate fast and accurate lookups using a modified k-d tree algorithm called the "best-bin-first" search method. However, images in [7] generate on the order of 1,000 feature vectors, of which only 3 need to agree on a class for the method to propose the solution. By their experiments, they are able to use a cut-off for examining at most 200 neighbours in a probabilistic best-bin-first search of 30,000 feature vectors with almost no loss of performance compared

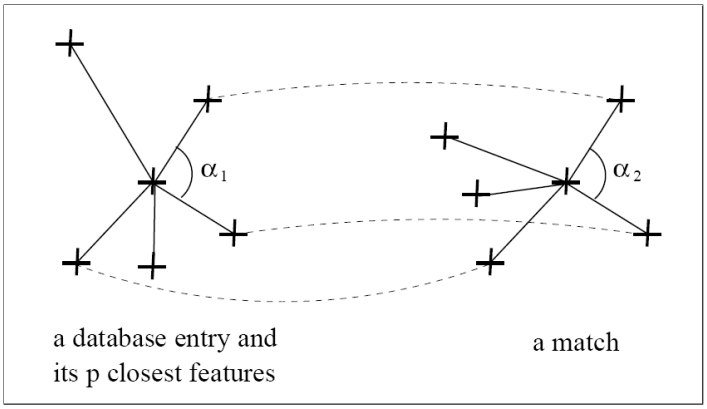


Figure 1.12: Semi local constraints; neighbours of the point have to match and angles have to correspond. This example is a reprint from [4].

to finding an exact solution. By the enormous redundancy in the feature vectors, given by the fact that 3 out of 1,000 vectors suffice as evidence, a performance gain of 1 to 150 is considerably devaluated.

Instead of simply optimizing the traversal of a k-d tree, [4] proposes the use of local shape configurations, as in figure 1.12. In order to increase the recognition rate, a geometric constraint is added. This constraint is based on the angle between neighbour points, and it requires that these angles have to be locally consistent, for example the angles α_1 and α_2 in figure 1.12. Semi local constraints have previously been used in [48] and [49].

These constraints are what our method carries across into the inexact matching of attributed relation graphs. The idea itself is to augment the measurement of vertex- compatibility by the implications of the current associations. Just as [4] deduces the angles α_1 and α_2, the association of two vertices from one graph to two vertices in another implies the association of the connecting edges. Instead of simply comparing the connectivity of vertices, our method refines vertex-compatibility by the implied edge-associations.

By the nature of our refinements we evaluate every possible vertex-associations between the two graphs G and H only ever once. This has an enormous impact on the upper bound in α for MINIMUM COST SUBGRAPH ISOMORPHISM in both equation 1.5 and 1.6, as shown in figures 1.13(a) and 1.13(b), and obvious in its equation:

$$\alpha(|V_1| \cdot |V_2|) + \beta(|V_1| - 1) \cdot \frac{|V_2|!}{(|V_2| - |V_1|)!} \tag{1.7}$$

Notice that the α component has been completely disconnected from the exponential; effectively making its contribution to the runtime approach zero as the number of vertices in the graphs increases. This improvement is so profound that we will, for the rest of this thesis, simply disregard the α component of the runtime. Any further references to runtimes, unless explicitly stated otherwise, will imply only the β component.

Although this improvement may seem like a trivial and immediate result of our method, it makes for an enormous leap in bounding the solution runtime.

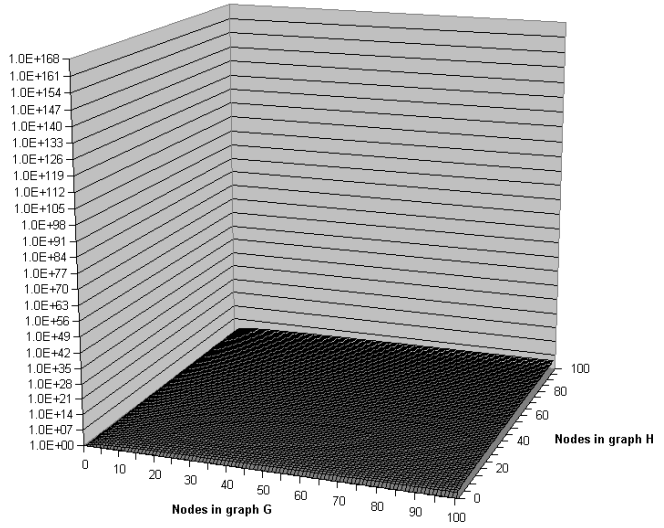
1.6.1 Algorithm overview

Algorithm 1 is a rough description of our approach to finding an approximate solution to any instance of MINIMUM COST SUBGRAPH ISOMORPHISM. This is intended as an overview that relates each step of the algorithm to other research – the formal and complete description is given in chapter 2.

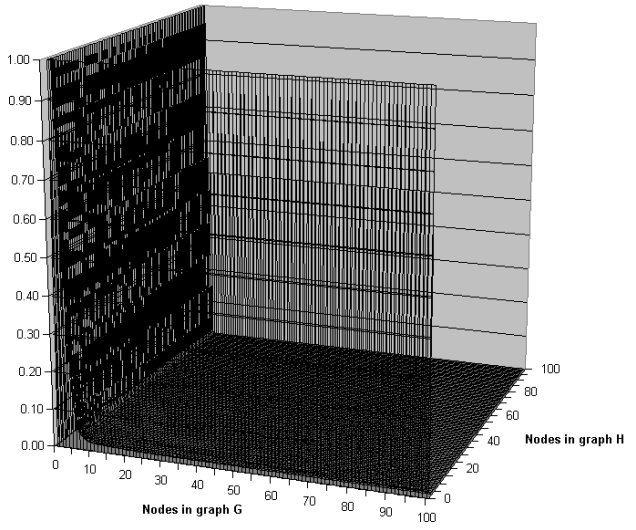
Algorithm 1 An overview of our approximation method

1: Allocate a cost matrix C of size $|V_1| \times |V_2|$.
2: Assign to c_{ij} the stochastic cost of associating $v_i \in V_1$ to $v_j \in V_2$.
3: $A \leftarrow \emptyset$
4: $c_A \leftarrow \infty$
5: **for** the n most compatible associations (v_k, v_l) by C **do**
6: Apply the syntactic cost implied by the association (v_k, v_l) to every element $c_{pq} \in C$ and store in C'.
7: :
8: (possibly recurse steps 5-16 by C')
9: :
10: Solve the assignment problem by C' and store in A'.
11: Sum elements of C' that are part of A' as $c_{A'}$.
12: **if** $c_{A'} < c_A$ **then**
13: $A \leftarrow A'$
14: $c_A \leftarrow c_{A'}$
15: **end if**
16: **end for**
17: Return A as minimum cost subgraph isomorphism.

Steps 1 and 2 construct a matrix C to hold the compatibility of any vertex $v_i \in V_1$ to any vertex $v_j \in V_2$. This is analogous to the compatibility coefficients, C_{aibj}, in [8], but vastly more efficient. For example, two 200 node undirected graphs with 10% connectivity would need a million element list of floating point numbers along with associated book-keeping in memory by [8], whereas our method only requires a matrix of 200×200 numbers for any connectivity.



(a) α component of improved upper bound.



(b) Magnitude of α component over corresponding bound in figure 1.5(a).

Figure 1.13: Improved upper bound for MINIMUM COST SUBGRAPH ISOMORPHISM in α.

The values of each element $c_{ij} \in C$ are given by the metric $Ev(v_i, v_j)$. This measurement will differ with each problem instance according to what type of pattern primitives are held by the vertices of the ARG's.

The set A is a set of vertex-tuples on the form (v_i, v_j) where $v_i \in V_1$ and $v_j \in V_2$. It is initialized in step 3, and updated in step 13 to always contain the minimum cost isomorphism between graphs G and H. The variable c_A is the sum cost of the associations in A.

Step 5 constitutes the scheduler of our method. The loop through steps 5-16 is controlled by this logic, and therefore requires special attention during implementation. If this step fails to discard any possible association (v_k, v_l), our method will traverse the whole search space.

Relaxation matching, as introduced to shape matching in [33] and [32], proposes the use of an "association graph." This is a structure that effectively holds the entire search space, but methods propose ways of pruning this before traversal. By the memory requirements noted above from [8], this is not an idea that holds as problem instances grow. We propose instead to evaluate possible associations in order of decreasing compatibility, as given by the matrix C. Any published method to prune the "association graph" can be applied in this step, to the same end.

The authors of [10] term ϵ-homomorphisms as a morphism between two graphs whose sum error, or cost, is below a threshold ϵ. We propose the use a association threshold ε on the element c_{kl}, that corresponds to the association (v_k, v_l) being evaluated, as well as a morphism threshold ϵ that discards any partial solution A' (see below) whose sum cost exceeds it.

In [4] the application of semilocal constraints is coupled with the requirement that no more than 50 percent of the pattern primitives need to be matched correctly. In [7] this requirement is lowered to 3 matches out of 1,000 pattern primitives. This is applied to step 5 by keeping the number n as low as experimental results allow. This is analogous to the low-level scheduling in [36].

The proposition of a cut-off on the number of total associations examined in [7] can be implemented by a total number N of associations examined for each problem instance. Let us emphasize that our logic processes the associations in order of increasing cost, by N our method will efficiently only be discarding the least compatible associations.

Step 6 considers syntactic compatibility implied by the assignment (v_k, v_l). At this point, the association (v_k, v_l) is assumed. The element c_{pq} holds the compatibility of $v_p \in V_1$ and $v_q \in V_2$, and can therefore be refined by the compatibility of the edges $(v_k, v_p) \in E_1$ and $(v_l, v_q) \in E_2$, given by the metric Ee.

Contrary to the stochastic compatibility of the vertices that require a unique

implementation for every possible type of pattern primitive, the attributes of our edges have been argued to always hold the spatial difference of the connected vertices. By this rational, the cost of associating edges needs simply be the spatial difference of the two. Let vector $\vec{e_1}$ be the attribute of edge (v_k, v_p) and vector $\vec{e_2}$ be the attribute of edge (v_l, v_q). In the range $[0, 1]$, this measurement is then given by:

$$De(\vec{e_1}, \vec{e_2}) = \begin{cases} 0, & \text{if both edges are missing;} \\ 1, & \text{if either edge is missing;} \\ \frac{1}{\sqrt{d}} \, |\vec{e_1} - \vec{e_2}|, & \text{otherwise.} \end{cases} \tag{1.8}$$

The denominator $\sqrt{d}$ expresses the maximum length possible by a vector in a unit extent hypercube of dimension d. This expression requires that each vector element is scaled to a number in the range $[0, 1]$ of the corresponding image dimension. In any standard 2-dimensional image this is achieved by dividing the first element of the vector by image width, and the second element by image height. The denominator would then be $\sqrt{2}$.

This measurement needs to be scaled and transformed by the edge cost metric Ee, as appropriate for the values of the vertex cost metric Ev.

Steps 7-9 are an abbreviation that expresses the recursive nature of our method. The algorithm implies a depth-first traversal of all the vertex associations that are allowed by step 5. The idea is simple; 1) store the current vertex tuple (v_k, v_l) in a partial solution P, and 2) run steps 5-16 on the modified compatibility matrix C' supplying P as a partial solution to A'. For each recursion the cost matrix is progressively refined using the syntactic compatibility as it becomes available by the implied edge associations. At first recursion, every element c'_{ij} has a single implied edge by the association in P. The second, deeper recursion implies two edges for every element c'_{ij}, and so forth.

Because the algorithm only considers the edges implied by the current vertex association, there will never be more than a single edge implied for any one element c'_{ij}. Therefore, the syntactic compatibility refinement presented above can be reapplied without modification.

The recursion is analogous to what [10] terms "looking ahead." At first recursion we are "looking ahead by one," at second recursion we achieve "looking ahead by two," and so forth. By the compatibility matrix C we are able to reduce computational complexity significantly, and we can look ahead by any number of steps that available memory allows. As opposed to [50], [51] and [8] our method even allows for negative expressions of compatibility.

As suggested by [4], [39] and [7] it is not necessary for every pattern primitive to be compatible to constitute a good match; as noted before, [7] is content with even

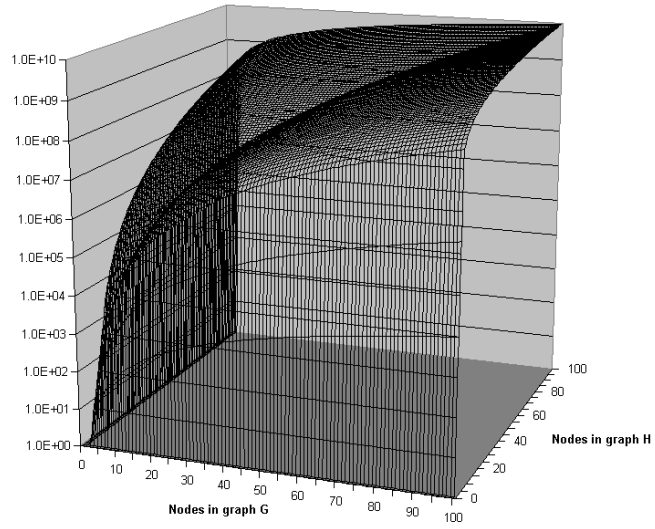


Figure 1.14: Runtime of depth-limited search.

3 out of 1,000 being compatible. By experiments we have found that the accuracy of our method does not increase notably beyond the first few recursions. A series of experiments that used simulated annealing to tune controlling parameters found that even 3 levels of recursion sufficed to yield optimal results for a wide range of graphs.

By depth-limit alone, even a full traversal of search space now scales elegantly with increasing instance sizes (see figure 1.14, as compared to figure 1.5(b)).

There is an obvious increase in memory requirements as the depth of recursion increases, as each recursion requires a duplication of the current compatibility matrix. Fortunately, since our method runs depth-first, there is only linear growth, and by the limitation to a very shallow recursion depth the memory footprint remains minimal.

Step 10 requires a solution to "the assignment problem" by C'. Let us briefly explain this problem.

Imagine a problem whose objective is to find the maximum assignment given a square benefit matrix C' of numbers. That is, we are given a set of variables c'_{ij} where $c'_{ij} \in R^1$. Then we associate a variable $m_{ij} \in \{0, 1\}$ with each c_{ij}, such that $\forall j \sum_{i=1}^{I} m_{ij} = 1$ and $\forall i \sum_{j=1}^{J} m_{ij} = 1$. Our aim is to find the matrix M which maximizes the following:

$$Dm(M) = \sum_{i=1}^{I} \sum_{j=1}^{J} m_{ij} c'_{ij} \tag{1.9}$$

This is the assignment problem, a classic problem in combinatorial optimization (see [52]).

In [50] it is proven that any square matrix whose elements are all positive will converge to a double stochastic matrix just by the iterative process of alternatively normalizing the rows and columns (a doubly stochastic matrix is a matrix whose elements are all positive and whose rows and columns all add up to one). This idea was combined with the "softmax" of [51] to solve the assignment problem by convergence in [8].

Unfortunately, the softmax is itself based on convergence, and the solution to the assignment problem becomes a slow multi-layered converging algorithm with many controlling parameters. Furthermore, the requirement of positive-only costs carries implications on how the cost metrics Ee and Ev must be implemented; forcing all cost into non-negativity by [53] requires intimate knowledge on the behaviour of e and leads to a complicated behaviour of all values. Our method allows predictable and linear behaviour of cost, considerably simplifying the implementation of any metric.

We propose a fast 2-step approximate solution to the assignment problem.

First, construct an auxiliary matrix C'' whose elements c''_{ij} are the distance of the corresponding c'_{ij} to the mean values of the i-th row and j-th column of C' by:

$$c''_{ij} = c'_{ij} - \frac{1}{2}(\frac{1}{I} \sum_{k=1}^{I} c'_{kj} + \frac{1}{J} \sum_{l=1}^{J} c'_{il}) \tag{1.10}$$

Second, traverse the elements $c''_{ij} \in C''$ in order of increasing value. For every c''_{ij}, if neither row i or column j is marked, add the corresponding assignment to the solution. Then mark both row i and column j. This process halts by its own logic when either all rows or all columns are marked.

Steps 11-15 are simple bookkeeping of the currently best found subgraph isomorphism, given by the vertex association tuples in set A.

Step 17 returns A as the minimum cost subgraph isomorphism from graph G to graph H. By the logic of the loop through steps 5-16, the set of vertex associations A is, at all times, the currently best found isomorphism.

Part I

An efficient solution to inexact graph matching

Chapter 2

Graph matcher algorithm

The graph matcher algorithm we propose in this chapter is our second contribution. It is a two-part recursive algorithm that consists of an initialization part (see chapter 2.1) and a recursion part (see chapter 2.2). All controlling parameters of the algorithm are contained in an entity we refer to as a "domain expert" (see section 2.3). The use and refinement of a matrix of vertex-association costs is an application of ideas from dynamic programming, and therefore achieves milestone 3.

A solution to an instance of MINIMUM COST SUBGRAPH ISOMORPHISM is given by a set of vertex-vertex associations, or vertex tuples, (v_i, v_j) where $v_i \in V_1$ and $v_j \in V_2$, that associates every vertex in the smaller graph $G = (V_1, E_1)$ to some vertex in the larger graph $H = (V_2, E_2)$.

For the sake of brevity in our algorithm descriptions we define a vertex tuple $(v_i, *)$ to be any tuple that has vertex v_i as its first element, and similarly a tuple $(*, v_j)$ to be any tuple with vertex v_j as its second element.

To resolve any ambiguity in the description below an implementation of the algorithm has been included in the appendix (see listing A.1).

2.1 Initialization

The initialization part of the algorithm is mostly an implementation detail needed for any recursive approach to a problem; it is clearly needed as a first step because recursion is just a series of calls to itself. As noted in chapter 1.6, this is the only time our algorithm will ever look at stochastic compatibility between the pattern primitives held by the graph vertices. Recursion is called upon an empty

isomorphism A, which is complete upon return.

Algorithm 2 Graph matcher, initialization

1: Allocate a cost matrix C of size $|V_1| \times |V_2|$.
2: **for all** $v_i \in V_1$ **do**
3: **for all** $v_j \in V_2$ **do**
4: $c_{ij} \leftarrow Ev(v_i, v_j)$
5: **end for**
6: **end for**
7: Perform recursion for $A = \emptyset$ and C.
8: Return A as minimum cost subgraph isomorphism.

Step 1 allocates space for a 2-dimensional matrix C of size $|V_1| \times |V_2|$. For every possible association of a vertex $v_i \in V_1$ to any vertex $v_j \in V_2$, there is a corresponding element $c_{ij} \in C$ to hold the cost of this. This matrix is different from the traditional goodness coefficients, because it exists at minimum size, and it is duplicated at every call to algorithm 3. The memory footprint of our method is limited to $|C| \cdot target - depth$ by depth-first.

Steps 2 - 6 assigns values to each element $c_{ij} \in C$ by the metric Ev.

Step 7 calls upon the recursion half of the algorithm, using an initial empty set A and the cost matrix C. As control returns from recursion, the set A contains the best found subgraph isomorphism from graph G to graph H.

Step 8 terminates the algorithm by returning set A as the solution to the problem instance.

2.2 Recursion

The recursion part of the algorithm is called repeatedly by itself through steps 13 - 28 until it reaches some target conditions where steps 30 - 36 terminate and solutions begin to ascend to the initial call. We argue in chapter 1.4 that the only interesting component of the algorithm runtime is the number of edge-comparisons performed during a complete run, and therefore trace this count in every pass through steps 1 - 5.

Our strategy to find a complete match is to allow only sufficiently compatible graphs to reach target depth, at which point an isomorphism is suggested. By the sum cost c of each suggested isomorphisms, our algorithm chooses the one which is the most compatible.

Steps 1 - 5 applies the vertex tuple (v_k, v_l) that was added to A just prior to entering this recursion to refine the values of the cost matrix C. The refinements

Algorithm 3 Graph matcher, recursion

for all $v_i \in V_1$ **do**
  for all $v_j \in V_2$ **do**
    $c_{ij} \leftarrow c_{ij} + Ee((v_i, v_k), (v_j, v_l)))$, where (v_k, v_l) is the last added tuple in A.
  end for
end for
Allocate a contribution matrix C' of size $|V_1| \times |V_2|$.
for $i = 0$ to $|V_1|$ **do**
  for $j = 0$ to $|V_2|$ **do**
    $c'_{ij} \leftarrow c_{ij} - \frac{1}{2}(\frac{1}{|V_1|}\sum_{k=1}^{|V_1|} c_{kj} + \frac{1}{|V_2|}\sum_{l=1}^{|V_2|} c_{il})$
  end for
end for
if $|A| < n$, where n is target depth **then**
  $A_{Min} \leftarrow \emptyset$
  $c_{Min} \leftarrow \infty$
  for all c'_{ij} in order of increasing value **do**
    if $(v_i, *) \notin A$ and $(*, v'_j) \notin A$ **then**
      if (v_i, v'_j) is acceptable **then**
        $A_R \leftarrow A + (v_i, v'_j)$
        $C_R \leftarrow C$
        Perform recursion for A_R and C_R.
        if $c_R < c_{Min}$ **then**
          $A_{Min} \leftarrow A_R$
          $c_{Min} \leftarrow c_R$
        end if
      end if
    end if
  end for
  Return A_{Min} and c_{Min} to calling process.
else
  for all c'_{ij} in order of increasing value **do**
    if $(v_i, *) \notin A$ and $(*, v'_j) \notin A$ **then**
      $A \leftarrow A + (v_i, v'_j)$
    end if
  end for
  $c \leftarrow \sum_{ij}^{A} c_{ij}$
  Return A and c to calling process.
end if

are done by the metric Ee over the edges implied to equal under this tuple. See chapter 1.6 for more information on this. On the initial call to recursion the set A is empty, consequently ignoring this step altogether.

Step 6 allocates space for a 2-dimensional matrix C' similar to the cost matrix C. This is an implementation detail to solve the "assignment problem", as proposed in chapter 1.6.

Steps 7 - 11 assigns values to each element $c'_{ij} \in C'$. Although, as presented, this appears to have high computational complexity, it is easily reducible to a two-pass traversal of C by first calculating the sum of each row i and each column j (see listing A.1). Unlike methods that rely solely on such a solution to perform graph matching, we regard this as a mere sorting operation on the cost- and contribution matrix.

Step 12 branches to keep recursion going until we reach some "target depth" n. The value of n is a controlling parameter of the algorithm, so by design it is given by the domain expert.

Steps 13 - 14 reserves space for book-keeping the currently best-found solution. A_{Min} will, at any time, hold the minimum cost subgraph isomorphism found, if any, and c_{Min} the corresponding sum cost.

Step 15 traverses vertex-vertex associations in the order of which their contribution c'_{ij} to row i and column j in C is minimum.

Step 16 branches to keep recursion off vertices in either graph that has already been associated by a tuple in A.

Step 17 is a crucial branch in the algorithm; it aims to discard any association that does not pass some criteria set by a series of controlling parameters. See chapter 2.3 for a complete discussion of the domain expert.

Step 18 duplicates the current isomorphism A as A_R, extending it by adding the currently accepted vertex tuple (v_i, v_j). This duplication is necessary to allow recursions that do not directly alter A.

Step 19 duplicates the refined cost matrix C as C_R. Duplication is again necessary to allow recursions that do not alter the cost matrix C.

Step 20 performs the actual recursion on the duplicate A_R and C_R.

Steps 21 - 24 performs book-keeping on the currently best found subgraph isomorphism.

Step 28 returns A_{Min} to the recursion level above, eventually returning to the original call from algorithm 2.

Steps 30 - 34 adds vertex tuples (v_i, v_j) to A in the order of which their contribution c'_{ij} to row i and column j in C is minimum. Step 31 ensures that the

isomorphism by A stays one-to-one. This is the greatest strength of our algorithm – we are able to achieve sufficient convergence of C' after only a few recursions, to confidently suggest a complete subgraph isomorphism.

Step 35 calculates the sum cost of the isomorphism by A over the cost matrix C; this is the sum of the elements $c_{ij} \in C$ for all vertex tuples $(v_i, v_j) \in A$.

Step 36 returns the complete subgraph isomorphism given by A and its sum cost c to the recursion level above.

2.3 The domain expert explained

In order to make our algorithm as general as possible, applicable to any domain where problems can be expressed as inexact matching of graphs, the controlling parameters have been segmented into a separate entity. The domain expert is, as the name implies, considered an expert in the field it belongs to.

The immediate problem with a cost based recursion is that it puts a lot of strain on the choice of values and value-ranges. If some weights are flawed, either attenuated or exaggerated, it might have a serious impact on the outcome of the matcher. To this end the metrics Ev and Ee are key elements of the domain expert, and their values will never be considered by any entity or parameter outside the scope of the expert.

In step 35 of algorithm 3 there is a summation of the elements $c_{ij} \in C$ to determine the goodness of the chosen isomorphism. Any such summation of traditional positive-only cost is bound to bias smaller subgraphs, since these constitute fewer elements to sum. Not only will this inherently prevent the larger prototype graphs to be chosen as a match, no matter how good, but by the introduction of noise smaller graphs can match just about any subgraph of a larger graph. To this end we allow negative cost for both stochastic- and syntactic compatibility.

As cutting-edge research find new ways to prune traversal in the search of subgraph isomorphisms, these results can be applied directly to the domain experts without any change to the algorithm itself. With every result presented in this thesis there will be an accompanying description of what parameters were used.

Although the segmentation of controlling parameters is very much an implementation detail, its purpose is two-fold; 1) it makes the algorithm description a lot less cluttered, and therefore more readable, and 2) by our object matcher in chapter 8 it becomes a helpful tool to discriminate between several sets of parameters.

Chapter 3

Graph matcher illustration

In this chapter we introduce an implementation of a domain expert, complete with vertex- and edge-metrics and a set of controlling parameters for the algorithm. The illustration is done in the domain of coloured regions; each vertex in our graphs are assigned a 3-byte RGB colour. The four graphs in figure 3.1 make up the library of known graphs for the purpose of these illustrations.

The vertex metric calculates the sum of differences between each of the three colour-components in the colours held by the compared vertices v_i and v_j. Since each of the components are byte-values, the denominator scales the sum difference into the range $[0, 1]$. The constant $\frac{1}{2}$ shifts the range to $[-\frac{1}{2}, \frac{1}{2}]$ to achieve negative costs when the difference is small.

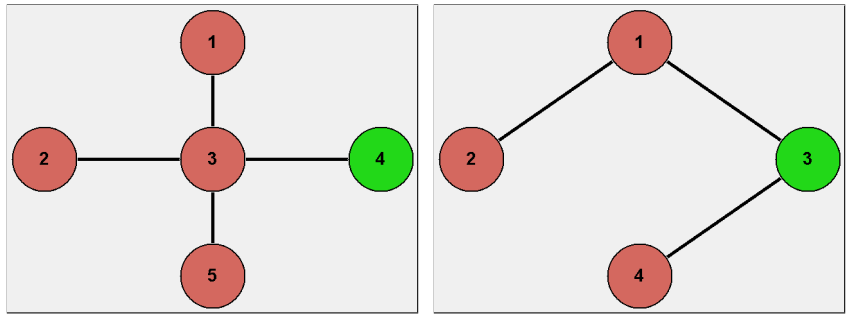
$$Ev(v_i, v_j) = \frac{|R_i - R_j| + |G_i - G_j| + |B_i - B_j|}{3 \cdot 255} - \frac{1}{2} \tag{3.1}$$

The edge metric is, as argued in chapter 1.6, the difference of the spatial relationships $\vec{e_i}$ and $\vec{e_j}$, held by the compared edges e_i and e_j respectively. The constant $\frac{1}{10}$ is used to achieve negative refinements when the difference is small.

$$Ee(e_i, e_j) = |\vec{e_i} - \vec{e_j}| - \frac{1}{10} \tag{3.2}$$

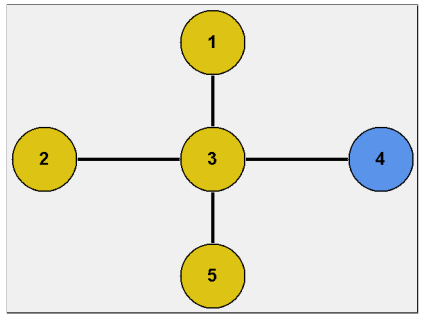
The target depth in step 12 of algorithm 3 is chosen to be 3, on the grounds given in chapter 1.6. The number of iterations through the recursion loop in steps 15 - 27 of the same algorithm are limited to 3 by a low-level cut-off, a value chosen on experience with the algorithm. In addition our domain expert limits the total number of recursions to 9 by a high-level cut-off.

In step 17 of algorithm 3 our domain expert determines acceptability of the

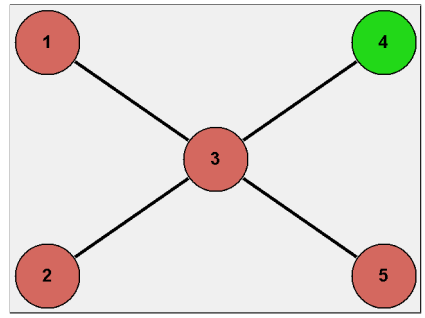


(a) Graph 1

(b) Graph 2



(c) Graph 3

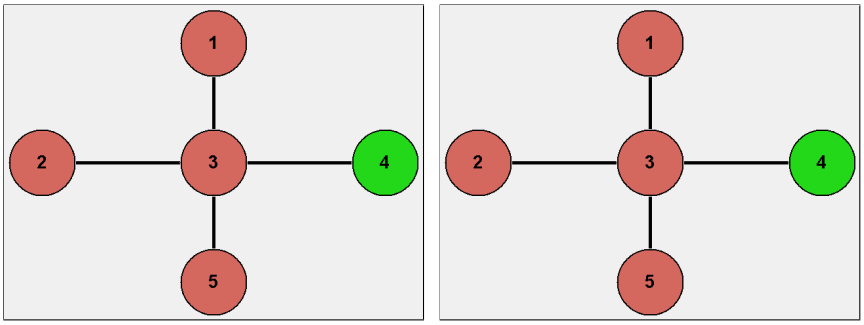


(d) Graph 4

Figure 3.1: The complete content of our example graph library.

association (v_i, v_j) by the value of element $c_{ij} \in C$. The threshold enforced by the domain expert is a dynamic value given by the function $\epsilon = a|A| + b$, where $a = -\frac{1}{10}$ and $b = -\frac{1}{2}$. The choice of a and b are reasoned from the metrics Ev and Ee; the chosen b rejects anything but perfectly matched colours, and through the chosen a it rejects any graph whose edges to not agree perfectly.

In step 36 an additional threshold filters the isomorphisms on the grounds of the average cost per vertex association in A; $\bar{c} = c/|A|$. This threshold $\varepsilon = -\frac{1}{2}$ ensures that no degeneration of the isomorphism can occur in the loop through steps 31 - 34.



(a) Graph G (b) Graph H

Figure 3.2: Graphs that perfectly match.

3.1 Details of a perfect match

In this section we illustrate how the algorithm matches a graph to itself. Figure 3.2 is a graphical representation of the graphs.

To better demonstrate program flow we use a recursion identifier r that is printed as a header before the corresponding paragraphs. These identifiers are assigned incrementally as the algorithm performs recursions. With each header we also print the current depth to provide additional information to the reader.

Initialization

The initiation algorithm 2 builds the cost matrix C, below, of naive vertex-vertex assignments by the metric Ev. Since this is a match to itself, see how the diagonal through the matrix constitutes the least-cost match.

		Graph G				
		1	2	3	4	5
Graph H	1	-0.50	-0.50	-0.50	0.00	-0.50
	2	-0.50	-0.50	-0.50	0.00	-0.50
	3	-0.50	-0.50	-0.50	0.00	-0.50
	4	0.00	0.00	0.00	-0.50	0.00
	5	-0.50	-0.50	-0.50	0.00	-0.50

The matrix is passed into the recursion (algorithm 3) with an empty assignment set $A = \emptyset$.

Recursion 0, depth 0

Steps 1 - 5 are skipped because A is empty.

Steps 6 - 11 builds the contribution-matrix C':

		Graph G				
		1	2	3	4	5
Graph H	1	-0.10	-0.10	-0.10	0.25	-0.10
	2	-0.10	-0.10	-0.10	0.25	-0.10
	3	-0.10	-0.10	-0.10	0.25	-0.10
	4	0.25	0.25	0.25	-0.40	0.25
	5	-0.10	-0.10	-0.10	0.25	-0.10

The achievement of this matrix is that it locates the key association $(4, 4)$, even though the corresponding element $c_{ij} \in C$ is no different from those of associating any of the vertices $\{1, 2, 3, 5\}$ to any other in the same set.

Because $c_{44} = -0.5$ passes the threshold $\epsilon = -\frac{1}{10} \cdot 0 - \frac{1}{2}$, the first recursion occurs for $A_R = \{(4, 4)\}$.

Recursion 1, depth 1

The cost matrix C is refined in steps 1 - 8 on the assumption that vertex 4 in graph G equals vertex 4 in graph H. For every element $c_{ij} \in C$ the edge metric Ee applies a modification based on its agreement with this association. As commented in chapter 1.6, reflection makes no sense in our model, so a total number of $|V_1 - 1| \cdot |V_2 - 1|$ refinements are applied.

		Graph G				
		1	2	3	4	5
Graph H	1	-0.60	0.11	-0.10	~~0.00~~	0.40
	2	0.11	-0.60	-0.10	~~0.00~~	0.11
	3	-0.10	-0.10	-0.60	~~0.00~~	-0.10
	4	~~0.00~~	~~0.00~~	~~0.00~~	~~-0.50~~	~~0.00~~
	5	0.40	0.11	-0.10	~~0.00~~	-0.60

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.56	0.17	0.01	~~0.07~~	0.44
	2	0.17	-0.50	0.04	~~0.10~~	0.17
	3	0.01	0.04	-0.42	~~0.14~~	0.01
	4	~~0.07~~	~~0.10~~	~~0.14~~	~~-0.40~~	~~0.07~~
	5	0.44	0.17	0.01	~~0.07~~	-0.56

There is an arbitrary choice of association when two or more elements of C' are equal. In our case, recursion occurs for $A_R = \{(4,4),(5,5)\}$.

Recursion 2, depth 2

Again, the cost matrix C is refined by the implied edge-associations. The same amount of refinements occur at this point as in the previous recursion, and by the association $(5,5)$ the cost of association with vertex 4 in either graph is also changed.

		Graph G				
		1	2	3	4	5
Graph H	1	-0.70	0.71	0.30	~~0.61~~	~~0.40~~
	2	0.71	-0.70	0.30	~~0.90~~	~~0.11~~
	3	0.30	0.30	-0.70	~~0.40~~	~~-0.10~~
	4	~~0.61~~	~~0.90~~	~~0.40~~	~~-0.60~~	~~0.00~~
	5	~~0.40~~	~~0.11~~	~~-0.10~~	~~0.00~~	~~-0.60~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.96	0.45	0.15	~~0.34~~	~~0.29~~
	2	0.45	-0.96	0.15	~~0.64~~	~~-0.01~~
	3	0.15	0.15	-0.74	~~0.25~~	~~-0.10~~
	4	~~0.34~~	~~0.64~~	~~0.25~~	~~-0.86~~	~~-0.11~~
	5	~~0.29~~	~~-0.01~~	~~-0.10~~	~~-0.11~~	~~-0.56~~

Recursion occurs for $A_R = \{(4,4),(5,5),(2,2)\}$.

Recursion 3, depth 3

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.80	~~0.71~~	0.70	~~1.21~~	~~1.30~~
	2	~~0.71~~	~~-0.70~~	~~0.30~~	~~0.90~~	~~0.11~~
	3	0.70	~~0.30~~	-0.80	~~0.80~~	~~0.30~~
	4	~~1.21~~	~~0.90~~	~~0.80~~	~~-0.70~~	~~0.61~~
	5	~~1.30~~	~~0.11~~	~~0.30~~	~~0.61~~	~~-0.70~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-1.43	~~0.27~~	0.26	~~0.62~~	~~0.83~~
	2	~~0.27~~	~~-0.96~~	~~0.04~~	~~0.49~~	~~-0.19~~
	3	0.26	~~0.04~~	-1.06	~~0.39~~	~~0.01~~
	4	~~0.62~~	~~0.49~~	~~0.39~~	~~-1.26~~	~~0.16~~
	5	~~0.83~~	~~-0.19~~	~~0.01~~	~~0.16~~	~~-1.02~~

Since we are now at target depth, steps 30 - 34 complete the isomorphism by stepping through the elements of C' in order of increasing value. If the current element $c_{ij} \in C'$ does not correspond to an already associated vertex in either graph G or H, it is added to A.

This completes A as $\{(4,4),(5,5),(2,2),(1,1),(3,3)\}$.

Step 35 calculates the sum of the isomorphism as $c = -3.70$, which passes the isomorphism threshold ε because $-3.70/5 < -\frac{1}{2}$.

The isomorphism A is returned to the recursion level above.

Recursion 2, depth 2

As control returns to step 20 at depth 2, the returned data is compared to the book-keeping variable c_{Min}. Because $-3.70 < \infty$, the isomorphism $A_R = \{(4,4),(5,5),(2,2),(1,1),(3,3)\}$ is stored as A_{Min}.

At the second iteration of the loop through steps 15 - 27, the element $(1,1)$ is chosen for recursion. Notice that redundancy seems to occur since the association $(1,1)$ is included in A_{Min}. This is not the case, however, since the algorithm knows nothing about what remaining associations will make up the isomorphism this time around.

Recursion occurs for $A_R = \{(4,4),(5,5),(1,1)\}$

Recursion 4, depth 3

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.70~~	~~0.71~~	~~0.30~~	~~0.61~~	~~0.40~~
	2	~~0.71~~	-0.80	0.70	~~1.80~~	~~0.71~~
	3	~~0.30~~	0.70	-0.80	~~0.80~~	~~0.30~~
	4	~~0.61~~	~~1.80~~	~~0.80~~	~~-0.70~~	~~0.61~~
	5	~~0.40~~	~~0.71~~	~~0.30~~	~~0.61~~	~~-0.70~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.96~~	~~0.27~~	~~0.04~~	~~0.16~~	~~0.14~~
	2	~~0.27~~	-1.43	0.26	~~1.18~~	~~0.27~~
	3	~~0.04~~	0.26	-1.06	~~0.36~~	~~0.04~~
	4	~~0.16~~	~~1.18~~	~~0.36~~	~~-1.32~~	~~0.16~~
	5	~~0.14~~	~~0.27~~	~~0.04~~	~~0.16~~	~~-0.96~~

The isomorphism is completed through steps 30 - 34 as $A = \{(4,4),(5,5),(1,1),(2,2),(3,3)\}$, and step 35 calculates $c = -3.70$. This isomorphism is returned to the recursion level above.

Recursion 2, depth 2

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and finds that it is not an improvement. The isomorphism A_R is discarded.

Recursion occurs for $A_R = \{(4,4),(5,5),(3,3)\}$

Recursion 5, depth 3

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.80	1.32	~~0.30~~	~~1.21~~	~~1.30~~
	2	1.32	-0.80	~~0.30~~	~~1.80~~	~~0.71~~
	3	~~0.30~~	~~0.30~~	~~-0.70~~	~~0.40~~	~~-0.10~~
	4	~~1.21~~	~~1.80~~	~~0.40~~	~~-0.70~~	~~0.61~~
	5	~~1.30~~	~~0.71~~	~~-0.10~~	~~0.61~~	~~-0.70~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-1.47	0.65	~~-0.05~~	~~0.55~~	~~0.78~~
	2	0.65	-1.47	~~-0.05~~	~~1.13~~	~~0.20~~
	3	~~-0.05~~	~~-0.05~~	~~-0.74~~	~~0.05~~	~~-0.30~~
	4	~~0.55~~	~~1.13~~	~~0.05~~	~~-1.36~~	~~0.09~~
	5	~~0.78~~	~~0.20~~	~~-0.30~~	~~0.09~~	~~-1.06~~

The isomorphism is completed as $A = \{(4,4),(5,5),(3,3),(2,2),(1,1)\}$, with cost $c = -3.70$, and returned to the recursion level above.

Recursion 2, depth 2

Again, the returned isomorphism A_R is discarded because it is not an improvement over A_{Min}.

The domain expert now cuts off the recursion by its low-level cut-off parameter, and A_{Min} is returned to the recursion level above.

Recursion 1, depth 1

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and because $-3.70 < \infty$, it is stored as A_{Min}.

The association corresponding to the second lowest element $c'_{ij} \in C'$ is added to A, $(1,1)$, and recursion occurs for $A_R = \{(4,4),(1,1)\}$.

Recursion 6, depth 2

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.60~~	~~0.11~~	~~-0.10~~	~~0.00~~	~~0.40~~
	2	~~0.11~~	-0.70	0.30	~~0.90~~	0.71
	3	~~-0.10~~	0.30	-0.70	~~0.40~~	0.30
	4	~~0.00~~	~~0.90~~	~~0.40~~	~~-0.60~~	~~0.61~~
	5	~~0.40~~	0.71	0.30	~~0.61~~	-0.70

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.56~~	~~-0.01~~	~~-0.10~~	~~-0.11~~	~~0.29~~
	2	~~-0.01~~	-0.96	0.15	~~0.64~~	0.45
	3	~~-0.10~~	0.15	-0.74	~~0.25~~	0.15
	4	~~-0.11~~	~~0.64~~	~~0.25~~	~~-0.86~~	~~0.34~~
	5	~~0.29~~	0.45	0.15	~~0.34~~	-0.96

Recursion occurs for $A_R = \{(4,4),(1,1),(2,2)\}$.

Recursion 7, depth 3

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.70~~	~~0.11~~	~~0.30~~	~~0.61~~	~~1.30~~
	2	~~0.11~~	~~-0.70~~	~~0.30~~	~~0.90~~	~~0.71~~
	3	~~0.30~~	~~0.30~~	-0.80	~~0.80~~	0.70
	4	~~0.61~~	~~0.90~~	~~0.80~~	~~-0.70~~	~~1.21~~
	5	~~1.30~~	~~0.71~~	0.70	~~1.21~~	-0.80

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-1.02~~	~~-0.19~~	~~0.01~~	~~0.16~~	~~0.83~~
	2	~~-0.19~~	~~-0.96~~	~~0.04~~	~~0.49~~	~~0.27~~
	3	~~0.01~~	~~0.04~~	-1.06	~~0.39~~	0.26
	4	~~0.16~~	~~0.49~~	~~0.39~~	~~-1.26~~	~~0.62~~
	5	~~0.83~~	~~0.27~~	0.26	~~0.62~~	-1.43

The isomorphism is completed as $A = \{(4,4),(1,1),(2,2),(5,5),(3,3)\}$, with cost $c = -3.70$, and returned to the recursion level above.

Recursion 6, depth 2

As control returns to step 20 of the initial recursion, the isomorphism A_R is stored as A_{Min} through steps 21 - 24 since at this point $c_{Min} = \infty$.

Recursion occurs for $A_R = \{(4,4),(1,1),(3,3)\}$.

Recursion 8, depth 3

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.70~~	~~0.71~~	~~-0.10~~	~~0.61~~	~~1.30~~
	2	~~0.71~~	-0.80	~~0.30~~	~~1.80~~	1.32
	3	~~-0.10~~	~~0.30~~	~~-0.70~~	~~0.40~~	~~0.30~~
	4	~~0.61~~	~~1.80~~	~~0.40~~	~~-0.70~~	~~1.21~~
	5	~~1.30~~	1.32	~~0.30~~	~~1.21~~	-0.80

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-1.06~~	~~0.20~~	~~-0.30~~	~~0.09~~	~~0.78~~
	2	~~0.20~~	-1.47	~~-0.05~~	~~1.13~~	0.65
	3	~~-0.30~~	~~-0.05~~	~~-0.74~~	~~0.05~~	~~-0.05~~
	4	~~0.09~~	~~1.13~~	~~0.05~~	~~-1.36~~	~~0.55~~
	5	~~0.78~~	0.65	~~-0.05~~	~~0.55~~	-1.47

The isomorphism is completed as $A = \{(4,4),(1,1),(3,3),(5,5),(2,2)\}$, with cost $c = -3.70$, and returned to the recursion level above.

Recursion 6, depth 2

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and finds that it is not an improvement. The isomorphism A_R is discarded.

At this point, the only possible next association to add to A is $(5,5)$. As commented above, the existence of an association in a complete isomorphism does not necessarily mean that all isomorphisms that include that association have been exhausted. However, any set of association that the algorithm has previously recursed are actually exhausted. The addition of $(5,5)$ to the set $\{(4,4),(1,1)\}$

would make a set equivalent to $\{(4,4),(5,5),(1,1)\}$ which has already been recursed. This redundancy is avoided in implementation (see listing A.1) by a hash-table that stores an ordered representation of every set that has been recursed.

Therefore, no further recursion happens, and A_{Min} is returned to the recursion level above.

Recursion 1, depth 1

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and finds that it is not an improvement. The isomorphism A_R is discarded.

Recursion occurs for $A_R = \{(4,4),(2,2)\}$.

Recursion 9, depth 2

At this point we reach the high-level cut-off parameter of the domain expert; no further recursions are allowed what so ever. Since we are not at target depth, the algorithm enters the block through steps 13 - 28, and in step 17 every association is rejected.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 1, depth 1

Step 21 discards the empty isomorphism A_R.

The isomorphism A_{Min} is returned to the recursion level above.

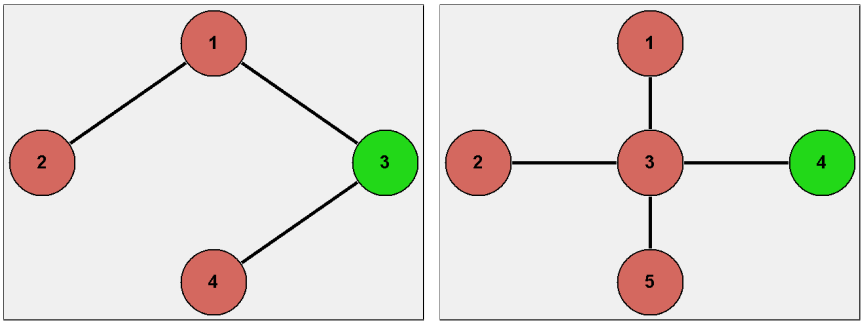
Recursion 0, depth 0

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and because $-3.70 < \infty$, it is stored as A_{Min}.

The domain expert rejects any further recursion, and A_{Min} is returned to step 7 of algorithm 2.

Initialization

The isomorphism $A = \{(4,4),(5,5),(2,2),(1,1),(3,3)\}$, with cost $c = 3.70$, is returned to the calling process as the solution to MINIMUM COST SUBGRAPH ISOMORPHISM from graph G to graph H.



(a) Graph G (b) Graph H

Figure 3.3: Graphs that contain a perfect subgraph match.

3.2 Details of a subgraph match

In this section we illustrate how the algorithm matches a graph to a subgraph of itself. Figure 3.3 is a graphical representation of the graphs. Notice that the only difference between the two graphs is that node 3 in graph H is missing from graph G.

Initialization

The cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	-0.50	-0.50	0.00	-0.50
	2	-0.50	-0.50	0.00	-0.50
	3	-0.50	-0.50	0.00	-0.50
	4	0.00	0.00	-0.50	0.00
	5	-0.50	-0.50	0.00	-0.50

Recursion 0, depth 0

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-0.10	-0.10	0.21	-0.10
	2	-0.10	-0.10	0.21	-0.10
	3	-0.10	-0.10	0.21	-0.10
	4	0.30	0.30	-0.39	0.30
	5	-0.10	-0.10	0.21	-0.10

Recursion occurs for $A_R = \{(3, 4)\}$.

Recursion 1, depth 1

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	-0.60	0.11	~~0.00~~	0.40
	2	0.11	-0.60	~~0.00~~	0.11
	3	-0.10	-0.10	~~0.00~~	-0.10
	4	~~0.00~~	~~0.00~~	~~-0.50~~	~~0.00~~
	5	0.40	0.11	~~0.00~~	-0.60

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-0.57	0.18	~~0.07~~	0.43
	2	0.17	-0.50	~~0.10~~	0.17
	3	-0.05	-0.01	~~0.09~~	-0.05
	4	~~0.07~~	~~0.11~~	~~-0.39~~	~~0.07~~
	5	0.43	0.18	~~0.07~~	-0.57

Recursion occurs for $A_R = \{(3, 4), (4, 5)\}$.

Recursion 2, depth 2

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	-0.70	0.71	~~0.61~~	~~0.40~~
	2	0.71	-0.70	~~0.90~~	~~0.11~~
	3	0.30	0.30	~~0.40~~	~~-0.10~~
	4	~~0.61~~	~~0.90~~	~~-0.60~~	~~0.00~~
	5	~~0.40~~	~~0.11~~	~~0.00~~	~~-0.60~~

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-0.97	0.45	~~0.34~~	~~0.32~~
	2	0.45	-0.97	~~0.63~~	~~0.03~~
	3	0.04	0.04	~~0.15~~	~~-0.17~~
	4	~~0.35~~	~~0.64~~	~~-0.85~~	~~-0.07~~
	5	~~0.24~~	~~-0.05~~	~~-0.15~~	~~-0.57~~

Recursion occurs for $A_R = \{(3,4),(4,5),(2,2)\}$.

Recursion 3, depth 3

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	-0.80	~~0.71~~	~~1.21~~	~~1.30~~
	2	~~0.71~~	~~-0.70~~	~~0.90~~	~~0.11~~
	3	0.70	~~0.30~~	~~0.80~~	~~0.30~~
	4	~~1.21~~	~~0.90~~	~~-0.70~~	~~0.61~~
	5	~~1.30~~	~~0.11~~	~~0.61~~	~~-0.70~~

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-1.43	~~0.31~~	~~0.62~~	~~0.86~~
	2	~~0.22~~	~~-0.97~~	~~0.45~~	~~-0.20~~
	3	0.10	~~-0.08~~	~~0.24~~	~~-0.11~~
	4	~~0.62~~	~~0.53~~	~~-1.26~~	~~0.20~~
	5	~~0.78~~	~~-0.19~~	~~0.12~~	~~-1.03~~

The isomorphism is completed as $A = \{(3,4),(4,5),(2,2),(1,1)\}$, with cost $c = -2.90$, and returned to the recursion level above.

Recursion 2, depth 2

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and because $-2.90 < \infty$, it is stored as A_{Min}.

Recursion occurs for $A_R = \{(3,4),(4,5),(1,1)\}$.

Recursion 4, depth 3

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	~~-0.70~~	~~0.71~~	~~0.61~~	~~0.40~~
	2	~~0.71~~	-0.80	~~1.80~~	~~0.71~~
	3	~~0.30~~	0.70	~~0.80~~	~~0.30~~
	4	~~0.61~~	~~1.80~~	~~-0.70~~	~~0.61~~
	5	~~0.40~~	~~0.71~~	~~0.61~~	~~-0.70~~

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	~~-0.97~~	~~0.22~~	~~0.12~~	~~0.13~~
	2	~~0.31~~	-1.43	~~1.17~~	~~0.31~~
	3	~~-0.08~~	0.10	~~0.20~~	~~-0.08~~
	4	~~0.21~~	~~1.18~~	~~-1.32~~	~~0.21~~
	5	~~0.13~~	~~0.22~~	~~0.12~~	~~-0.97~~

The isomorphism is completed as $A = \{(3,4),(4,5),(1,1),(2,2)\}$, with cost $c = -2.90$, and returned to the recursion level above.

Recursion 2, depth 2

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and finds that it is not an improvement. The isomorphism A_R is discarded.

The isomorphism A_{Min} is returned to the recursion level above.

Recursion 1, depth 1

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and because $-2.90 < \infty$, it is stored as A_{Min}.

Recursion occurs for $A_R = \{(3,4),(1,1)\}$.

Recursion 5, depth 2

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	~~-0.60~~	~~0.11~~	~~0.00~~	~~0.40~~
	2	~~0.11~~	-0.70	~~0.90~~	0.71
	3	~~-0.10~~	0.30	~~0.40~~	0.30
	4	~~0.00~~	~~0.90~~	~~-0.60~~	~~0.61~~
	5	~~0.40~~	0.71	~~0.61~~	-0.70

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	~~-0.57~~	~~-0.05~~	~~-0.15~~	~~0.24~~
	2	~~0.03~~	-0.97	~~0.63~~	0.45
	3	~~-0.17~~	0.04	~~0.15~~	0.04
	4	~~-0.07~~	~~0.64~~	~~-0.85~~	~~0.35~~
	5	~~0.32~~	0.45	~~0.34~~	-0.97

Recursion occurs for $A_R = \{(3,4),(1,1),(2,2)\}$.

Recursion 6, depth 3

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	~~-0.70~~	~~0.11~~	~~0.61~~	~~1.30~~
	2	~~0.11~~	~~-0.70~~	~~0.90~~	~~0.71~~
	3	~~0.30~~	~~0.30~~	~~0.80~~	0.70
	4	~~0.61~~	~~0.90~~	~~-0.70~~	~~1.21~~
	5	~~1.30~~	~~0.71~~	~~1.21~~	-0.80

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	~~-1.03~~	~~-0.19~~	~~0.12~~	~~0.78~~
	2	~~-0.20~~	~~-0.97~~	~~0.45~~	~~0.22~~
	3	~~-0.11~~	~~-0.08~~	~~0.24~~	0.10
	4	~~0.20~~	~~0.53~~	~~-1.26~~	~~0.62~~
	5	~~0.86~~	~~0.31~~	~~0.62~~	-1.43

The isomorphism is completed as $A = \{(3,4),(1,1),(2,2),(4,5)\}$, with cost $c = -2.90$, and returned to the recursion level above.

Recursion 5, depth 2

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and because $-2.90 < \infty$, it is stored as A_{Min}.

The isomorphism A_{Min} is returned to the recursion level above.

Recursion 1, depth 1

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and finds that it is not an improvement. The isomorphism A_R is discarded.

Recursion occurs for $A_R = \{(3,4),(2,2)\}$.

Recursion 7, depth 2

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	-0.70	~~0.11~~	~~0.61~~	1.30
	2	~~0.11~~	~~-0.60~~	~~0.00~~	~~0.11~~
	3	0.30	~~-0.10~~	~~0.40~~	0.30
	4	~~0.61~~	~~0.00~~	~~-0.60~~	~~0.61~~
	5	1.30	~~0.11~~	~~0.61~~	-0.70

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-1.03	~~0.04~~	~~0.35~~	0.97
	2	~~-0.06~~	~~-0.50~~	~~-0.09~~	~~-0.06~~
	3	0.01	~~-0.13~~	~~0.18~~	0.01
	4	~~0.34~~	~~0.00~~	~~-0.79~~	~~0.34~~
	5	0.97	~~0.04~~	~~0.35~~	-1.03

At this point there are only two associations the association threshold ϵ; $(1, 1)$ and $(4, 5)$. Adding any of these two associations to A yield isomorphisms that have already been exhausted. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 1, depth 1

Step 21 discards the empty isomorphism A_R.

The isomorphism A_{Min} is returned to the recursion level above.

Recursion 0, depth 0

Step 21 compares the cost of the returned isomorphism A_R to A_{Min}, and because $-2.90 < \infty$, it is stored as A_{Min}.

Recursion occurs for $A_R = \{(2, 5)\}$.

Recursion 8, depth 1

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	0.11	~~-0.50~~	1.31	0.98
	2	0.40	~~-0.50~~	1.48	0.81
	3	-0.10	~~-0.50~~	1.02	0.52
	4	-0.10	~~0.00~~	0.11	0.90
	5	~~-0.50~~	~~-0.50~~	~~0.00~~	~~-0.50~~

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-0.06	~~-0.44~~	0.63	0.45
	2	0.20	~~-0.47~~	0.77	0.26
	3	-0.17	~~-0.34~~	0.43	0.09
	4	-0.17	~~0.16~~	-0.47	0.47
	5	~~-0.33~~	~~-0.10~~	~~-0.34~~	~~-0.69~~

At this point there are no associations that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 0, depth 0

Step 21 discards the empty isomorphism A_R.

Recursion occurs for $A_R = \{(2, 3)\}$.

Recursion 9, depth 1

The refined cost matrix C is:

		Graph G			
		1	2	3	4
Graph H	1	-0.10	~~-0.50~~	1.02	0.52
	2	0.52	~~-0.50~~	1.40	0.52
	3	~~-0.50~~	~~-0.50~~	~~0.00~~	~~-0.50~~
	4	0.40	~~0.00~~	-0.10	0.40
	5	0.52	~~-0.50~~	1.02	-0.10

The contribution matrix C' is:

		Graph G			
		1	2	3	4
Graph H	1	-0.30	~~-0.34~~	0.51	0.32
	2	0.22	~~-0.44~~	0.79	0.22
	3	~~-0.45~~	~~-0.10~~	~~-0.27~~	~~-0.45~~
	4	0.23	~~0.18~~	-0.59	0.23
	5	0.32	~~-0.34~~	0.51	-0.30

Again, there are no available associations that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 0, depth 0

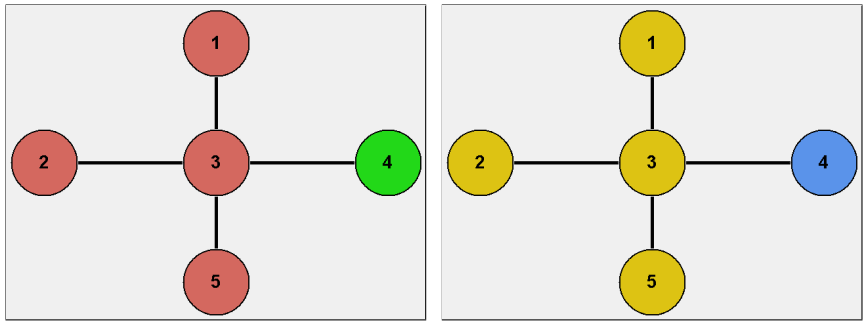
Step 21 discards the empty isomorphism A_R.

The isomorphism A_{Min} is returned to step 7 of algorithm 2.

Initialization

The isomorphism $A = \{(3,4),(4,5),(2,2),(1,1)\}$, with cost $c = -2.90$, is returned to the calling process as the solution to MINIMUM COST SUBGRAPH ISOMORPHISM from graph G to graph H.

Note how the summation of negative costs allows the larger isomorphism in the previous section to achieve a lower minimum cost than this. Had the costs been required to be all non-negative it would detract from performing larger isomorphisms.



(a) Graph G (b) Graph H

Figure 3.4: Graphs that do not match stochastically.

3.3 Details of a stochastic mismatch

In this section we illustrate how the algorithm attempts to match graphs that have stochastic differences. Figure 3.4 is a graphical representation of the graphs. The stochastic information in a graph is given by the attributes of its vertices. The stochastic difference between graphs G and H are therefore the difference between the colours assigned to their nodes.

Initialization

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.17	-0.17	-0.17	0.17	-0.17
	2	-0.17	-0.17	-0.17	0.17	-0.17
	3	-0.17	-0.17	-0.17	0.17	-0.17
	4	0.00	0.00	0.00	0.00	0.00
	5	-0.17	-0.17	-0.17	0.17	-0.17

Recursion 0, depth 0

The contribution matrix C' is:

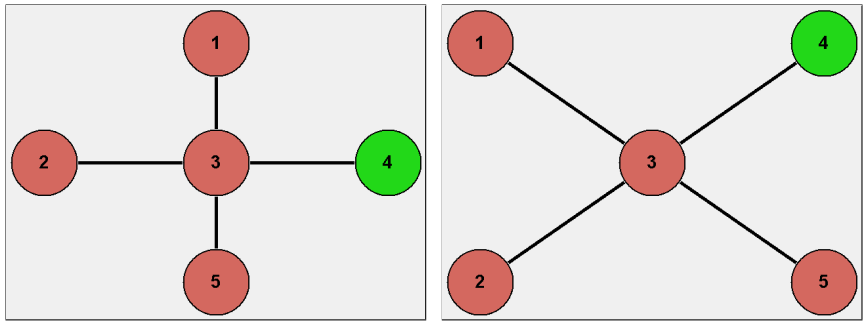
		Graph G				
		1	2	3	4	5
Graph H	1	-0.05	-0.05	-0.05	0.15	-0.05
	2	-0.05	-0.05	-0.05	0.15	-0.05
	3	-0.05	-0.05	-0.05	0.15	-0.05
	4	0.07	0.07	0.07	-0.07	0.07
	5	-0.05	-0.05	-0.05	0.15	-0.05

There are no elements $c_{ij} \in C$ that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to step 7 of algorithm 2.

Initialization

The algorithm has found no solution to MINIMUM COST SUBGRAPH ISOMORPHISM from graph G to graph H.



(a) Graph G (b) Graph H

Figure 3.5: Graphs that do not match syntactically.

3.4 Details of a syntactic mismatch

In this section we illustrate how the algorithm attempts to match graphs that have syntactic differences. Figure 3.5 is a graphical representation of the graphs. The syntactic information is traditionally defined by the edges in a graph. Since our graphs are always fully connected, the syntactic difference is given by the difference of the attributes of corresponding edges, consequently by the difference of the spatial layout of the two graphs.

Initialization

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.50	-0.50	-0.50	0.00	-0.50
	2	-0.50	-0.50	-0.50	0.00	-0.50
	3	-0.50	-0.50	-0.50	0.00	-0.50
	4	0.00	0.00	0.00	-0.50	0.00
	5	-0.50	-0.50	-0.50	0.00	-0.50

Recursion 0, depth 0

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.10	-0.10	-0.10	0.25	-0.10
	2	-0.10	-0.10	-0.10	0.25	-0.10
	3	-0.10	-0.10	-0.10	0.25	-0.10
	4	0.25	0.25	0.25	-0.40	0.25
	5	-0.10	-0.10	-0.10	0.25	-0.10

Recursion occurs for $A_R = \{(4, 4)\}$.

Recursion 1, depth 1

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	0.11	0.11	-0.60	~~0.00~~	0.11
	2	0.81	0.40	0.11	~~0.00~~	-0.60
	3	0.52	0.52	-0.10	~~0.00~~	-0.10
	4	~~0.00~~	~~0.00~~	~~0.00~~	~~-0.50~~	~~0.00~~
	5	0.98	0.98	0.40	~~0.00~~	0.11

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.11	-0.07	-0.55	~~0.08~~	0.18
	2	0.50	0.13	0.05	~~-0.02~~	-0.62
	3	0.19	0.23	-0.16	~~-0.03~~	-0.14
	4	~~-0.19~~	~~-0.15~~	~~0.07~~	~~-0.40~~	~~0.10~~
	5	0.49	0.53	0.17	~~-0.20~~	-0.09

Recursion occurs for $A_R = \{(4, 4), (5, 2)\}$.

Recursion 2, depth 2

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	0.71	0.71	-0.70	~~0.61~~	~~0.11~~
	2	~~0.81~~	~~0.40~~	~~0.11~~	~~0.00~~	~~-0.60~~
	3	1.54	1.54	0.30	~~0.40~~	~~-0.10~~
	4	~~0.90~~	~~1.31~~	~~0.61~~	~~-0.60~~	~~0.00~~
	5	2.46	2.46	1.30	~~0.61~~	~~0.11~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.07	-0.07	-1.01	~~0.36~~	~~0.01~~
	2	~~0.10~~	~~-0.31~~	~~-0.13~~	~~-0.17~~	~~-0.62~~
	3	0.53	0.53	-0.23	~~-0.07~~	~~-0.42~~
	4	~~0.04~~	~~0.45~~	~~0.22~~	~~-0.92~~	~~-0.17~~
	5	1.13	1.13	0.44	~~-0.19~~	~~-0.54~~

Recursion occurs for $A_R = \{(4,4),(5,2),(3,1)\}$.

Recursion 3, depth 3

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~0.71~~	~~0.71~~	~~-0.70~~	~~0.61~~	~~0.11~~
	2	~~2.13~~	~~1.30~~	~~0.11~~	~~0.90~~	~~-0.70~~
	3	2.55	2.55	~~0.30~~	~~0.80~~	~~0.30~~
	4	~~1.80~~	~~2.63~~	~~0.61~~	~~-0.70~~	~~0.90~~
	5	3.94	3.94	~~1.30~~	~~1.21~~	~~0.71~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.54~~	~~-0.54~~	~~-1.01~~	~~0.18~~	~~-0.17~~
	2	~~0.64~~	~~-0.19~~	~~-0.43~~	~~0.24~~	~~-1.21~~
	3	0.79	0.79	~~-0.51~~	~~-0.13~~	~~-0.48~~
	4	~~0.16~~	~~0.99~~	~~-0.08~~	~~-1.51~~	~~0.24~~
	5	1.72	1.72	~~0.03~~	~~-0.18~~	~~-0.53~~

The isomorphism is completed as $A = \{(4,4),(5,2),(3,1),(1,3),(2,5)\}$, with cost $c = 4.40$. The isomorphism A does not pass the threshold ε, and nothing is returned to the recursion level above.

Recursion 2, depth 2

Step 21 discards the empty isomorphism A_R.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 1, depth 1

Step 21 discards the empty isomorphism A_R.

Recursion occurs for $A_R = \{(4,4),(3,1)\}$.

Recursion 4, depth 2

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~0.11~~	~~0.11~~	~~-0.60~~	~~0.00~~	~~0.11~~
	2	2.13	1.30	~~0.11~~	~~0.90~~	-0.70
	3	1.54	1.54	~~-0.10~~	~~0.40~~	0.30
	4	~~0.90~~	~~1.31~~	~~0.00~~	~~-0.60~~	~~0.90~~
	5	2.46	2.46	~~0.40~~	~~0.61~~	0.71

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	~~-0.58~~	~~-0.54~~	~~-0.55~~	~~-0.10~~	~~0.00~~
	2	1.04	0.25	~~-0.25~~	~~0.40~~	-1.21
	3	0.46	0.50	~~-0.45~~	~~-0.10~~	-0.20
	4	~~-0.06~~	~~0.39~~	~~-0.23~~	~~-0.98~~	~~0.52~~
	5	1.08	1.13	~~-0.25~~	~~-0.19~~	-0.08

There are no elements $c_{ij} \in C$ that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 1, depth 1

Step 21 discards the empty isomorphism A_R.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 0, depth 0

Step 21 discards the empty isomorphism A_R.

Recursion occurs for $A_R = \{(3, 3)\}$.

Recursion 5, depth 1

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.10	-0.10	~~-0.50~~	1.02	0.52
	2	0.52	-0.10	~~-0.50~~	1.02	-0.10
	3	~~-0.50~~	~~-0.50~~	~~-0.50~~	~~0.00~~	~~-0.50~~
	4	0.40	1.02	~~0.00~~	-0.10	1.02
	5	0.52	0.52	~~-0.50~~	0.40	-0.10

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.27	-0.27	~~-0.38~~	0.70	0.35
	2	0.35	-0.27	~~-0.38~~	0.70	-0.27
	3	~~-0.38~~	~~-0.38~~	~~-0.10~~	~~-0.03~~	~~-0.38~~
	4	0.08	0.70	~~-0.03~~	-0.57	0.70
	5	0.35	0.35	~~-0.38~~	0.08	-0.27

There are no elements $c_{ij} \in C$ that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 0, depth 0

Step 21 discards the empty isomorphism A_R.

Recursion occurs for $A_R = \{(3, 5)\}$.

Recursion 6, depth 1

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	0.11	0.11	~~-0.50~~	1.48	0.98
	2	0.40	-0.60	~~-0.50~~	1.31	0.40
	3	-0.10	-0.10	~~-0.50~~	1.02	0.52
	4	-0.10	0.90	~~0.00~~	0.40	1.31
	5	~~-0.50~~	~~-0.50~~	~~-0.50~~	~~0.00~~	~~-0.50~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.09	-0.09	~~-0.52~~	0.84	0.49
	2	0.32	-0.68	~~-0.40~~	0.79	0.03
	3	-0.16	-0.16	~~-0.38~~	0.51	0.16
	4	-0.33	0.67	~~-0.05~~	-0.27	0.79
	5	~~-0.28~~	~~-0.28~~	~~-0.10~~	~~-0.22~~	~~-0.57~~

Recursion occurs for $A_R = \{(3, 5), (2, 2)\}$.

Recursion 7, depth 2

The refined cost matrix C is:

		Graph G				
		1	2	3	4	5
Graph H	1	0.71	~~0.11~~	~~0.40~~	2.96	2.46
	2	~~0.40~~	~~-0.60~~	~~-0.50~~	~~1.31~~	~~0.40~~
	3	0.30	~~-0.10~~	~~-0.10~~	2.04	1.54
	4	-0.20	~~0.90~~	~~0.61~~	1.30	2.63
	5	~~0.11~~	~~-0.50~~	~~-0.60~~	~~0.61~~	~~0.11~~

The contribution matrix C' is:

		Graph G				
		1	2	3	4	5
Graph H	1	-0.08	~~-0.54~~	~~-0.25~~	1.48	1.08
	2	~~0.17~~	~~-0.68~~	~~-0.58~~	~~0.39~~	~~-0.41~~
	3	-0.20	~~-0.45~~	~~-0.45~~	0.85	0.46
	4	-0.86	~~0.40~~	~~0.10~~	-0.05	1.39
	5	~~0.00~~	~~-0.45~~	~~-0.55~~	~~-0.19~~	~~-0.58~~

There are no elements $c_{ij} \in C$ that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 6, depth 1

Step 21 discards the empty isomorphism A_R.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to the recursion level above.

Recursion 0, depth 0

There are no elements $c_{ij} \in C$ that pass the association threshold ϵ. No recursion occurs, so A_{Min} remains empty and $c_{Min} = \infty$.

The isomorphism $A_{Min} = \emptyset$, with cost $c_{Min} = \infty$, does not pass the isomorphism threshold ε, and nothing is returned to step 7 of algorithm 2.

Initialization

The algorithm has found no solution to MINIMUM COST SUBGRAPH ISOMORPHISM from graph G to graph H.

Chapter 4

Graph matcher results

4.1 Overview

For our experiments we generate a set of 300 attributed relation graphs, each containing somewhere between 1 and 100 vertices. Every vertex is assigned an arbitrary colour and spatial centroid. The colours are 3-byte RGB values. The centroids are 2-dimensional vectors with each component in the range $[0, 1]$. Edges are added to the graph to make it fully connected, their spatial relational vectors are derived from the connected vertices' centroids. Of the 300 graphs, 200 are copied into the library of prototype graphs.

We use the metrics from chapter 3.

Each experiment matches all 300 candidate graphs against all 200 prototype graphs, for a total of 60,000 performed matches, under varying conditions. For each of these 60,000 matches there are 4 different outcomes:

(a) The candidate graph is correctly matched to some prototype graph, we label these instances "True-Positive."

(b) The candidate graph is correctly unmatched to any prototype graph in library, labelled "True-Negative."

(c) The candidate graph is incorrectly match to some prototype graph, labelled "False-Positive."

(d) The candidate graph is incorrectly unmatched to any prototype graph in library, labelled "False-Negative."

These outcomes are counted using the corresponding variables a, b, c, and d. From these we derive the following statistics:

- **Accuracy** – A measure of the absence of error. Also a measure of how close an estimate is to its true value; it measures the portion of all decisions that were correct decisions. It is defined as $(a+b)/(a+b+c+d)$. It falls in the range $[0,1]$, with 1 being the best score.
- **Error** – This measures the portion of all decisions that were incorrect decisions. It is defined as $(c+d)/(a+b+c+d)$. It falls in the range $[0,1]$, with 0 being the best score.
- **Precision** – The degree of conformity among a set of observations. As such, precision is a measure of dispersion of the probability distribution associated with a measurement and expresses the degree of repeatability of a measurement. This is the portion of the assigned categories that were correct. It is defined as $a/(a+c)$. It falls in the range $[0,1]$, with 1 being the best score.
- **Recall** – This measures the portion of the correct categories that were assigned. It is defined as $a/(a+d)$. It falls in the range $[0,1]$, with 1 being the best score.
- **F1** – This measures an even combination of precision and recall. It is defined as $2pr/(p+r)$. In terms of a, b, and c, it may be expressed as $2a/(2a+c+d)$. It falls in the range $[0,1]$, with 1 being the best score.

The F1 measure is often the only simple measure that is worth trying to maximize on its own – consider the fact that you can get a perfect precision score by always assigning zero matches, or a perfect recall score by always assigning every match. A truly smart system will assign the correct matches, and only the correct matches, maximizing precision and recall at the same time, and therefore maximizing the F1 score.

Sometimes it is worth trying to maximize the accuracy score, but accuracy (and its counterpart error) are considered fairly crude scores that do not give much information about the performance of the matcher.

4.1.1 A note on the figures

Each experiment is presented with a graphical representation of their runtimes; one figure for each of the four outcomes of a match. As explained in chapter 1 we do not present the number of seconds elapsed on some given platform, but rather the number of refinements performed on the cost matrix during a match (i.e. the β component of the runtime).

Beneath each figure is the value of the corresponding count variable (a, b, c or d), the average depth inspected by the matcher, the average number of recursions performed, and the average number of refinements performed.

4.2 Improving runtimes

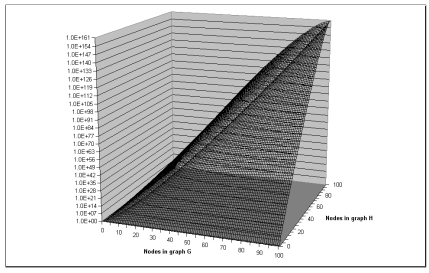
All our experiments are run using the same set of controlling parameters as the illustrations in chapter 3. This first section introduces each parameter, in turn, so that their individual contribution becomes apparent. Although these parameters reduce actual runtimes to a miniscule fraction of the upper-bound, there is no loss of accuracy when applied to our set of graphs.

The statistics below hold throughout the improvements applied in these sections:

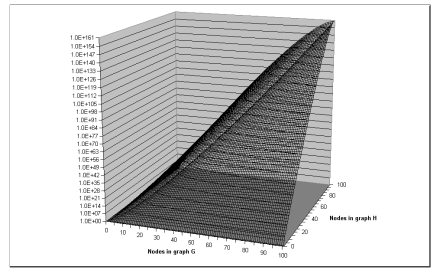
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

4.2.1 Unrestricted

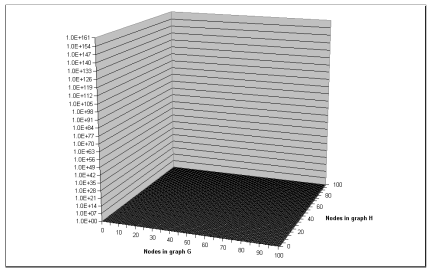
For reference we begin by running the algorithm without a target depth or any discrimination of associations at step 17 of algorithm 3. In effect, nothing is pruned, and the algorithm performs at the upper-bound of runtime.



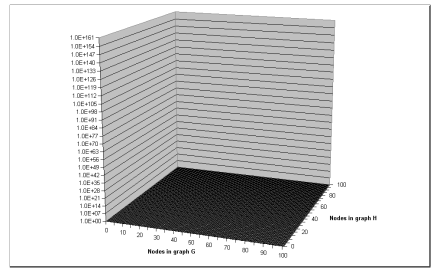
(a) True-Positive. Count 40,000, depth 50.5, recursions 4.18E+154, refinements 4.09E+156.



(b) True-Negative. Count 20,000, depth 50.5, recursions 4.18E+154, refinements 4.09E+156.



(c) False-Positive. None.



(d) False-Negative. None.

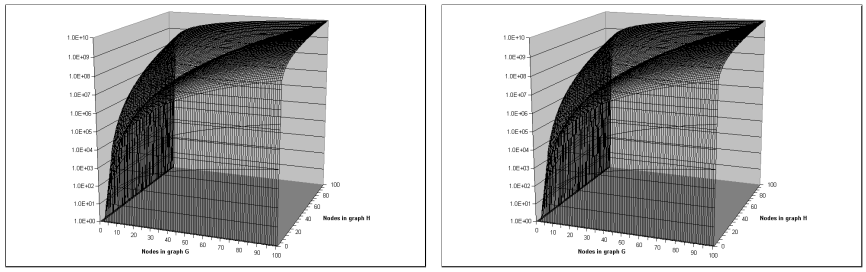
Figure 4.1: Unrestricted match.

4.2.2 Depth threshold

The first pruning technique we apply is target depth; by experiments we have found depth $n = 3$ to suffice for most circumstances. This halts all recursions when the current set of associations A contains 3 vertex tuples. At that point, steps 30 - 34 traverses the contribution matrix C' in order of increasing value to complete a subgraph isomorphism.

By the logic of our algorithm, this technique can be applied without compromising the results. Each vertex association in A implies an edge association for every single element $c_{ij} \in C$, and even at a low number of associations in A, the traversal-order of the contribution matrix C' (by the refined C) converges.

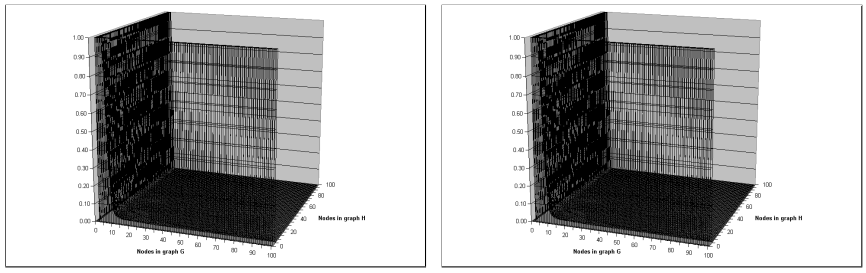
See figure 4.3 for significant improvement of runtimes over the unrestricted instances. The average number of applied refinements to the cost matrix C is reduced from 4.09E+156 to 1.35E+9.



(a) True-Positive. Count 40,000, depth 2.99, recursions 3.06E+7, refinements 1.35E+9.

(b) True-Negative. Count 20,000, depth 2.99, recursions 3.06E+7, refinements 1.35E+9.

Figure 4.2: Match with depth threshold.



(a) True-Positive.

(b) True-Negative.

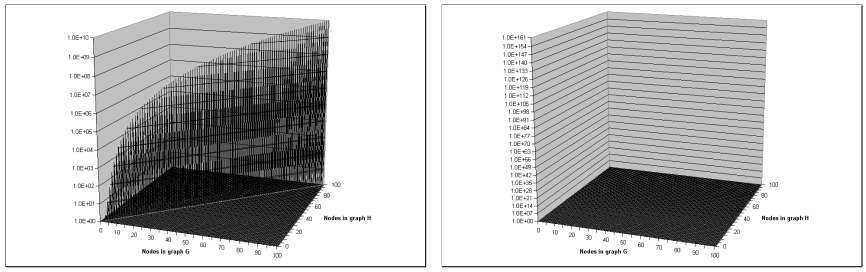
Figure 4.3: Improvement by depth threshold.

4.2.3 Depth- and association threshold

By applying a threshold on the association cost at step 17, the algorithm is able to prune most associations that are not part of a perfect subgraph isomorphism. The function $\epsilon = a|A| + b$ gives a dynamic threshold that becomes increasingly severe as depth increases. Reasoned from the metrics; $a = -\frac{1}{10}$ and $b = -\frac{1}{2}$.

An additional threshold is applied to the average cost per association, $\bar{c} = c/|A|$, at step 36 to filter isomorphisms that degenerate after reaching the target depth. This threshold is also reasoned from the metrics; $\varepsilon = -\frac{1}{2}$.

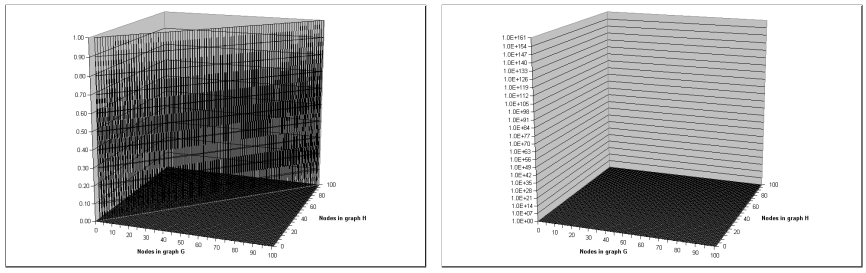
At this point, only the perfect matches perform the maximum amount of refinements. All mismatches are pruned without recursion. The number of applied refinements are down from 1.35E+9 to 1.58E+7.



(a) True-Positive. Count 40,000, depth 0.02, recursions 1.90E+5, refinements 1.58E+7.

(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.

Figure 4.4: Match with depth- and association threshold.



(a) True-Positive.

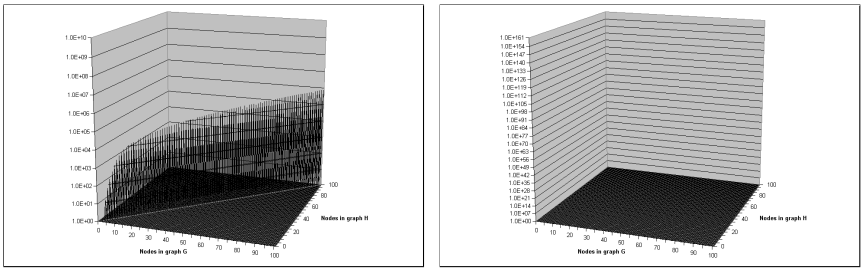
(b) True-Negative.

Figure 4.5: Improvement by depth- and association threshold.

4.2.4 Depth- and association threshold, restricted local recursions

The number of iterations through the recursion loop in steps 15 - 27 of algorithm 3, whose association is accepted by step 17, is counted and limited by a low-level cut-off. By experience we choose to allow 5 iterations; everything beyond the first 5 accepted associations are rejected. The rationale behind this approach is that the logic of the algorithm investigates the associations in a best-first fashion.

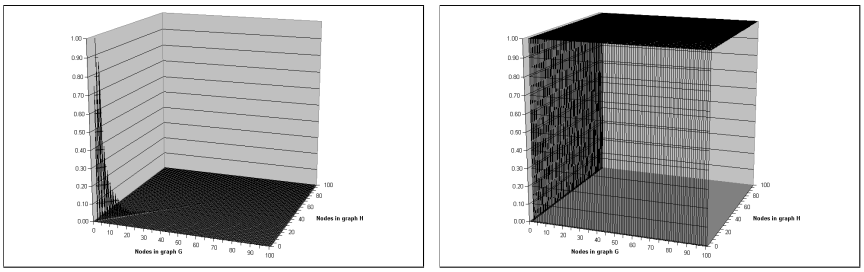
This method is able to prune away a large amount of the refinements performed on perfect matches. Refinements are reduced from 1.58E+7 to 2,494.40.



(a) True-Positive. Count 40,000, depth 0.01, recursions 0.71, refinements 2,494.40.

(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.

Figure 4.6: Match with depth- and association threshold, restricted local recursions.



(a) True-Positive.

(b) True-Negative.

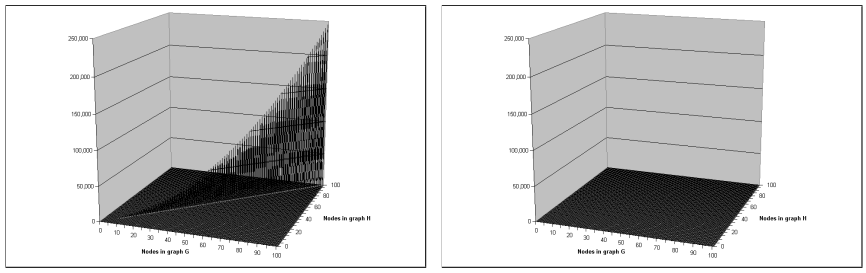
Figure 4.7: Improvement by depth- and association threshold, restricted local recursions.

4.2.5 Depth- and association threshold, restricted local and global recursions

The final pruning technique applied in algorithm 3 is a high-level cut-off that limits the total number of recursions allowed. As soon as that number of recursions have been performed, step 17 rejects all further associations. By experience we choose this cut-off to be 50.

In effect; for correct matches we are able to reduce the average number of refinements from 4.09E+156 to 820.96, whereas for correct unassigned matches we are able to reduce the average number of refinements from 4.09E+156 to 0.

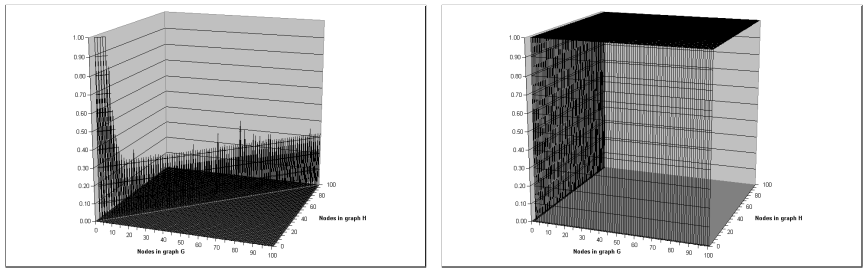
Statistical results remain perfect throughout.



(a) True-Positive. Count 40,000, depth 0.01, recursions 0.24, refinements 820.96.

(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.

Figure 4.8: Match with depth- and association threshold, restricted local and global recursions.



(a) True-Positive.

(b) True-Negative.

Figure 4.9: Improvement by depth- and association threshold, restricted local and global recursions.

4.3 Performance under noise

Using the set of pruning techniques introduced in chapter 4.3.2, the remainder of this chapter will demonstrate how our method runs under various types of noise. We define "noise" as some alteration to the candidate graph. Each type of noise is therefore a separate method of modifying a graph.

For each type of the noise there is also a measure p, whose value is given by the heading of each experiment. The value of p relates to the amount of noise that is applied to all candidate graphs before matching. The effect of p is different for each type of noise, and its meaning is given alongside the introduction of the noise type.

4.3.1 Spatial noise

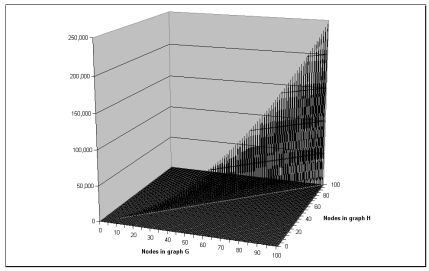
By "spatial noise" we mean modifications to the spatial information contained in the edges E of a graph $G = (V, E)$. Because our method requires spatially coherent graphs, any noise to a single edge can be smoothed by averaging against the spatial information contained in the other edges. Furthermore, since our graphs are fully connected, as the number of vertices in a graph increases, the number of edges increase exponentially, and the more resilient the graph becomes to this type of edge noise.

Instead of this simple noise, we apply spatial noise by first altering the spatial information in a selected edge e_i, and then updating all other edges $\forall e_j \in E, e_j \neq e_i$ to be spatially coherent with the modified e_i. This is analogous to moving a pattern primitive in a source image.

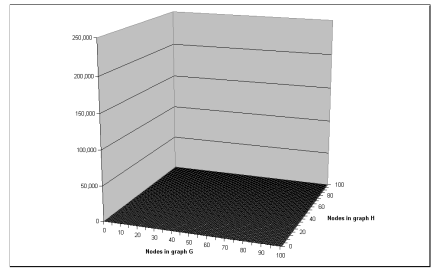
The percentage p is a measurement of the amount of noise that is applied to every single edge of the graph. All components $(x, y, z, ..)$ of the spatial relational vector $\vec{e_i}$ (its dimensionality is given by the source image) of edge $e_i \in E$ are modified by separate random values. These values are chosen from the range $[-\frac{1}{2}p, \frac{1}{2}p]$, scaled to the extent of the current component (e.g. if the x-component lies in the range $[-50, 75]$, the scale factor for that component is 125).

In these experiments we use 2-dimensional spatial relations where both the x- and the y-component lie in the range $[0, 1]$. The scaling we apply is therefore 1.0 for both components.

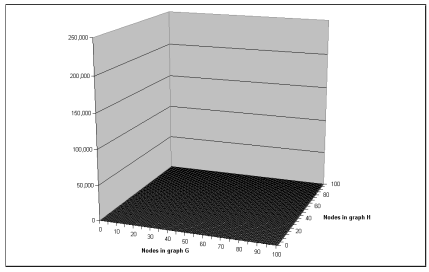
By the analogy of moving a pattern primitive in a source image, the measurement p is the maximum extent, expressed in a percentage of the image dimension, in which the move occurs.



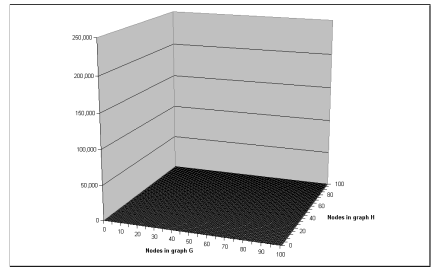
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.24, refinements 820.96.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

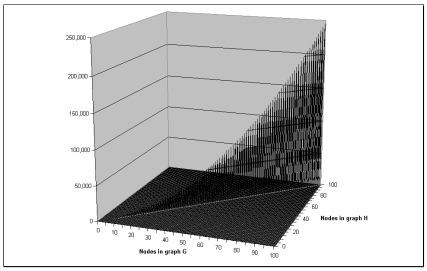
Figure 4.10: Match with 10% spatial noise.

10% spatial noise

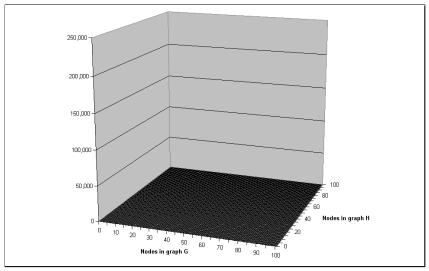
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The association threshold is $\epsilon = -\frac{1}{2}$, the average cost threshold $\varepsilon = -\frac{1}{2}$.

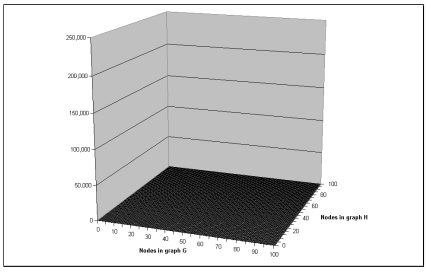
All results are perfect. By figure 4.10 it is evident that no time is spent on any of the matches except for those that give True-Positive results.



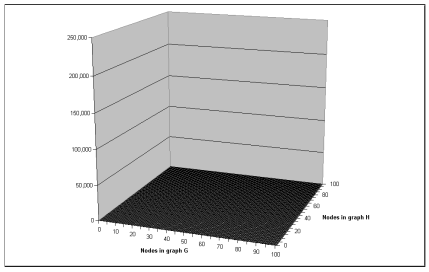
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.24, refinements 820.90.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

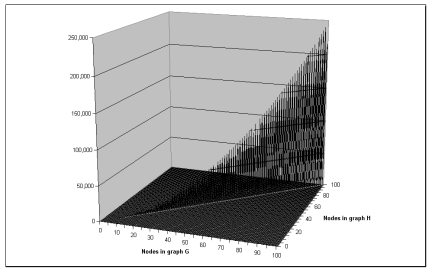
Figure 4.11: Match with 25% spatial noise.

25% spatial noise

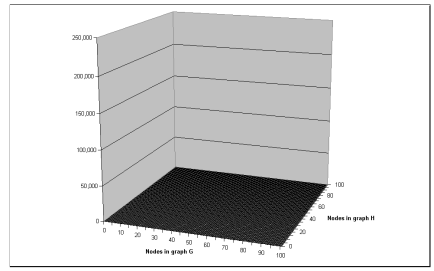
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = \frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = 0$.

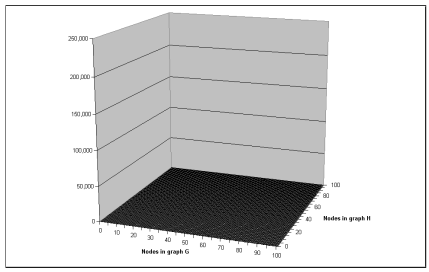
All results are perfect. Again, no runtime is spent investigating anything but True-Positive instances.



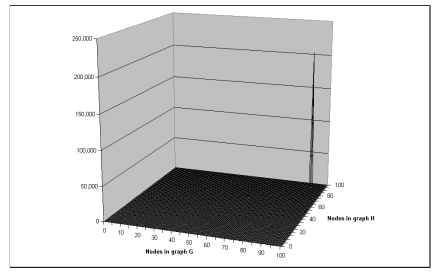
(a) True-Positive. Count 37,600, depth 0.01, recursions 0.22, refinements 842.69.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. Count 2,400, depth 0.01, recursions 0.06, refinements 180.70.

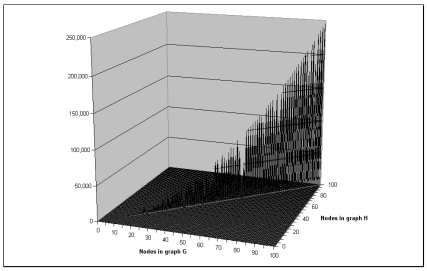
Figure 4.12: Match with 50% spatial noise.

50% spatial noise

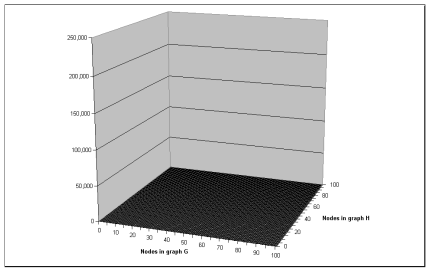
Accuracy	Error	Precision	Recall	F1
0.96	0.04	1.00	0.94	0.97

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = \frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = 0$.

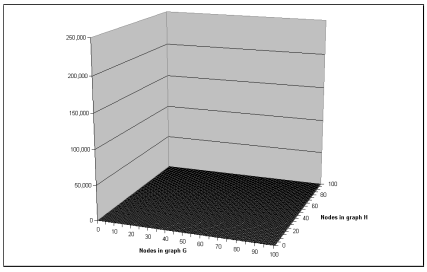
Some candidate graphs that should have been matched to corresponding prototype graphs where determined not to exist among the known graphs. As a result, the Recall value is not 1.0, and following the value of F1 is also not perfect. Some time is now spent on the False-Negative instances, which indicates that, even though the matcher did determine them unmatched, some time was spent on the candidate graphs that should have been assigned a match.



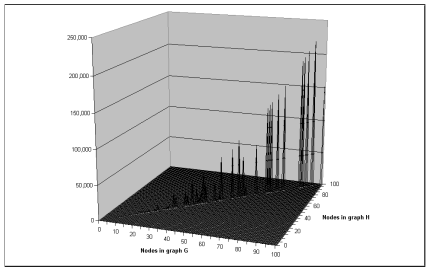
(a) True-Positive. Count 27,200, depth 0.02, recursions 0.21, refinements 919.13.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. Count 12,800, depth 0.01, recursions 0.12, refinements 387.02.

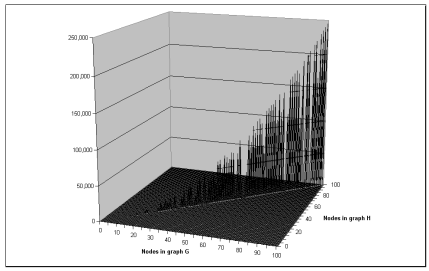
Figure 4.13: Match with 75% spatial noise.

75% spatial noise

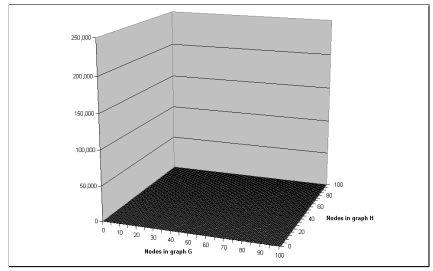
Accuracy	Error	Precision	Recall	F1
0.79	0.21	1.00	0.68	0.81

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = \frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = 0$.

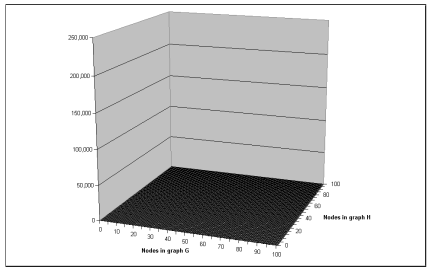
As above, the error that occurs are only False-Negatives. Even at this amount of spatial noise, Precision remains at 1.0, and F1 is good. No time is spent looking at instances except for True-Positives and False-Negatives. By comparing the values in figure 4.13 to those in figure 4.10 we see that the algorithm now requires more time to decide on a match. This trend seems to carry through the increase of noise, which is conceptually easy to understand.



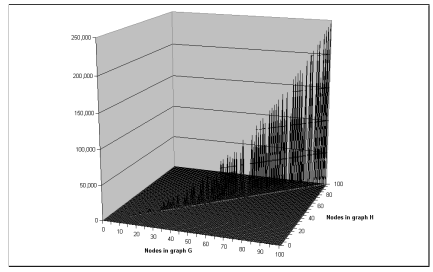
(a) True-Positive. Count 18,000, depth 0.01, recursions 0.21, refinements 953.45.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. Count 22,000, depth 0.01, recursions 0.13, refinements 509.91.

Figure 4.14: Match with 95% spatial noise.

95% spatial noise

Accuracy	Error	Precision	Recall	F1
0.63	0.37	1.00	0.45	0.62

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = \frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = 0$.

Even when the pattern primitives are allowed to move at random across the full extent of the source image, Precision remains 1.0. However, at this point there are more cases where candidate graphs are wrongly found not to exist among the prototype graphs, than there are cases where candidate graphs are correctly matched. The accuracy at which the algorithm finds instances that are True-Negatives salvages an impressive F1 even now.

The average amount of time spent on False-Negatives is less than the corresponding time spent on True-Positives, which indicates that the control parameters might be too restrictive to find the correct match.

4.3.2 Pattern noise

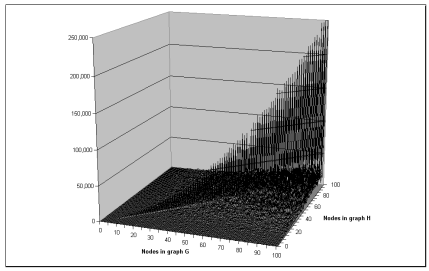
By "pattern noise" we mean modifications to the information contained in the vertices V of a graph $G = (V, E)$. The application of this noise to a single vertex $v_i \in V$ is distortion of all numerical values that are used to describe the pattern primitive of v_i.

The percentage p is a measurement of the amount of noise that is applied to every single vertex of the graph. All components of the pattern primitive of vertex v_i are modified by separate random values. As with spatial noise, these values are chosen from the range $[-\frac{1}{2}p, \frac{1}{2}p]$, scaled to the extent of the current component.

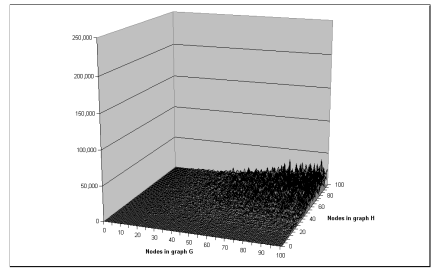
Because the pattern primitives we use in these experiments are 3-byte RGB values, each of its three components lie in the byte-range $[0, 255]$, and its extent is therefore 255.

Although it might seem backwards, this noise is actually quite easy to counter as long as the spatial relations in the edges of the graph remain unchanged. Although more time is required to find the correct isomorphisms, the refinements applied to the cost matrix C as the algorithm recurses likely associations will quickly converge when the first few associations have been correctly guessed. As the number of vertices in the graph increases, the probability that a necessary few vertices remain close to unchanged also increases, and consequently the algorithm can reach target depth with a set of correct associations.

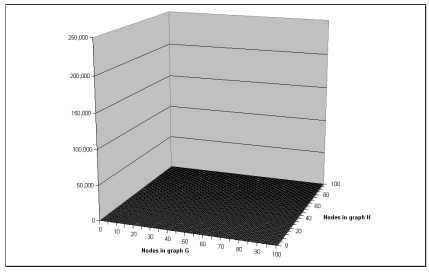
Consider the dynamic threshold function $\epsilon = a|A| + b$ that we use for filtering possible associations. To cope with pattern noise it is necessary to raise the value of b, which is the initial threshold applied to the cost matrix before spatial content of edge-edge associations have been considered, so that not only perfect associations are accepted. At the same time it is necessary to retain the most strict a, which expresses the refinement that an edge-edge association must apply to an association cost, so that the two vertices of an association are required to be spatially identical.



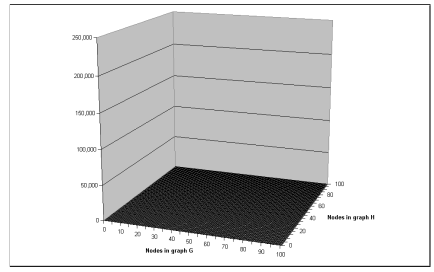
(a) True-Positive. Count 39,600, depth 0.49, recursions 1.12, refinements 4,156.72.



(b) True-Negative. Count 20,000, depth 0.47, recursions 0.87, refinements 3,301.05.



(c) False-Positive. None.



(d) False-Negative. Count 400, depth 0.18, recursions 0.21, refinements 0.00.

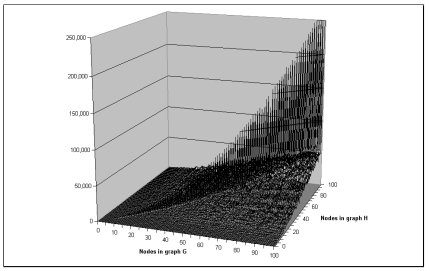
Figure 4.15: Match with 10% pattern noise.

10% pattern noise

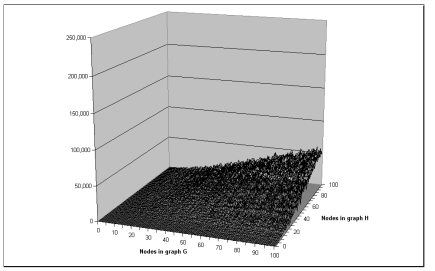
Accuracy	Error	Precision	Recall	F1
0.99	0.01	1.00	0.99	0.99

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{9}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

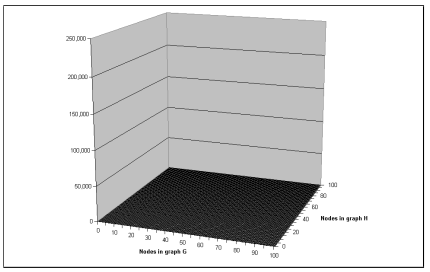
With this little noise applied to the graphs, the results are close to perfect. By the False-Negatives in figure 4.15 it seems that errors only occur when either of the two graphs are sufficiently small (there is no plot on the right-hand side of the diagram). This coincides with the comments on this type of noise above.



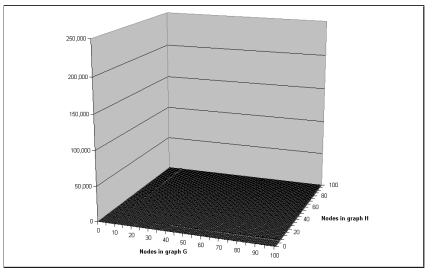
(a) True-Positive. Count 39,400, depth 0.98, recursions 3.52, refinements 11,654.77.



(b) True-Negative. Count 19,800, depth 0.97, recursions 3.32, refinements 10,882.58.



(c) False-Positive. Count 200, depth 0.72, recursions 1.54, refinements 98.80.



(d) False-Negative. Count 600, depth 0.71, recursions 1.63, refinements 240.58.

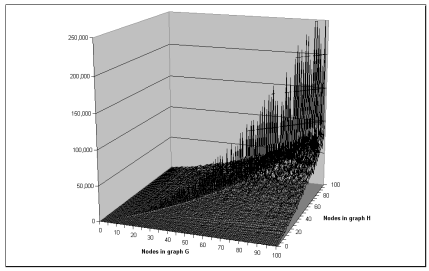
Figure 4.16: Match with 25% pattern noise.

25% pattern noise

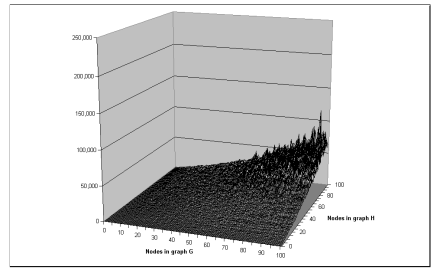
Accuracy	Error	Precision	Recall	F1
0.99	0.01	0.99	0.99	0.99

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{8}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

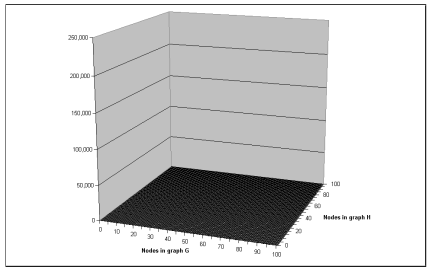
The results are quite similar to that of 10% noise, except that a lot more time is now spent on all four different outcomes. Both the False-Positives and False-Negatives in figure 4.16 indicate that errors only occur on smaller graphs.



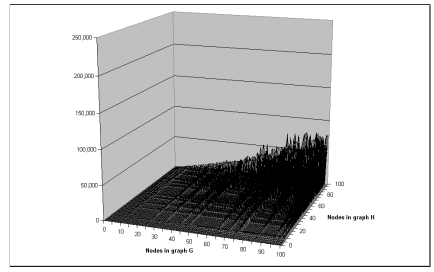
(a) True-Positive. Count 36,000, depth 1.34, recursions 5.17, refinements 15,586.38.



(b) True-Negative. Count 20,000, depth 1.33, recursions 4.98, refinements 14,780.52.



(c) False-Positive. None.



(d) False-Negative. Count 4,000, depth 1.21, recursions 4.43, refinements 12,425.40.

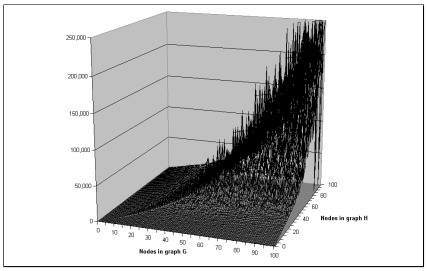
Figure 4.17: Match with 50% pattern noise.

50% pattern noise

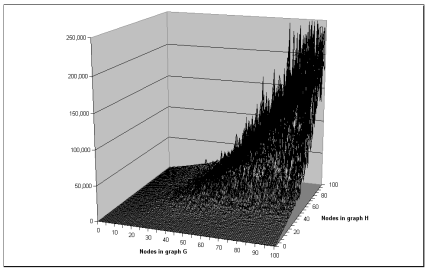
Accuracy	Error	Precision	Recall	F1
0.93	0.07	1.00	0.90	0.95

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{7}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

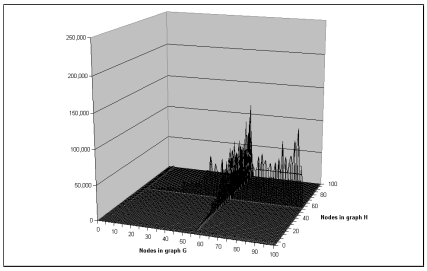
The results remain close to perfect, but again it comes at the cost of increased time. Keep in mind, however, that even the high ridge through the correct matches in the True-Positive diagram is still, at most, only a 6-digit number as opposed to original 156-digit one.



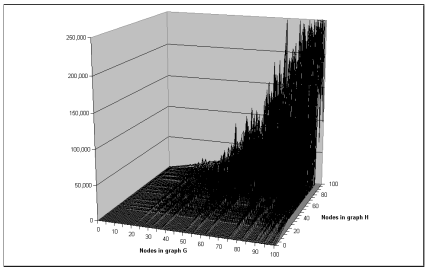
(a) True-Positive. Count 35,000, depth 1.92, recursions 9.34, refinements 31,102.34.



(b) True-Negative. Count 19,600, depth 1.93, recursions 9.22, refinements 31,911.27.



(c) False-Positive. Count 800, depth 1.66, recursions 6.81, refinements 9,625.25.



(d) False-Negative. Count 4,600, depth 1.93, recursions 9.89, refinements 44,514.98.

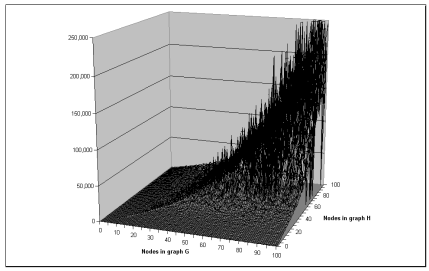
Figure 4.18: Match with 75% pattern noise.

75% pattern noise

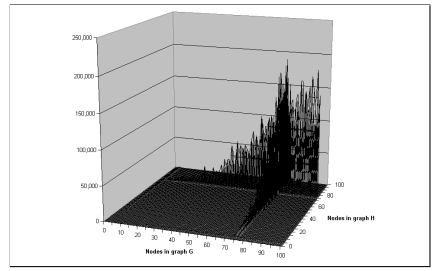
Accuracy	Error	Precision	Recall	F1
0.91	0.09	0.98	0.88	0.93

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{5}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

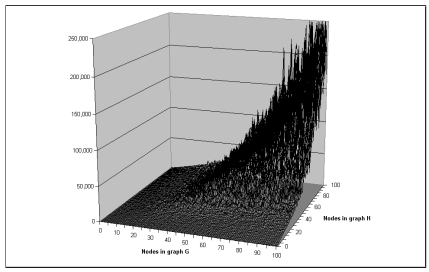
Even at this amount of noise the algorithm is able to come close to perfect results. The trend of increasing time with increasing noise is now obvious, and the average number of refinements for all cases is becoming quite large.



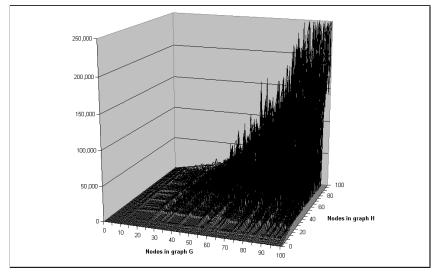
(a) True-Positive. Count 30,000, depth 1.90, recursions 9.04, refinements 29,936.80.



(b) True-Negative. Count 19,600, depth 1.91, recursions 9.01, refinements 30,968.93.



(c) False-Positive. Count 1,200, depth 1.72, recursions 7.72, refinements 20,024.06.



(d) False-Negative. Count 9,200, depth 1.90, recursions 9.52, refinements 37,207.25.

Figure 4.19: Match with 95% pattern noise.

95% pattern noise

Accuracy	Error	Precision	Recall	F1
0.83	0.17	0.96	0.77	0.85

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{5}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

At almost complete distortion of every single pattern primitive in the graph, the algorithm is still able to achieve an incredible F1 of 0.85. The error is primarily due to incorrectly unassigned matches that yield a low Recall. The average amount of refinements in both true outcomes are very similar, while the most time spent is actually in the False-Negatives. This indicates that the algorithm was halted by the low- and high-level cut-offs before it was able to find initial associations that would yield correct traversal of the cost matrix at target depth.

If even more time was allowed for the search it is likely that the algorithm would be able to solve the current False-Negatives, thereby achieving a higher Recall and consequently an even higher F1.

4.3.3 Vertex noise

By "vertex noise" we mean addition or removal of vertices in the candidate graph.

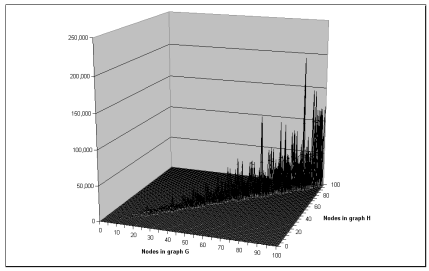
To add a new vertex v' to $G = (V, E)$ it requires a new pattern primitive and a set of $|V|$ new edges. In these experiments, the pattern primitive is a random 3-byte RGB value, found by 3 random values in the byte range $[0, 255]$. The set of edges are (v', v_i) and (v_i, v') for all $v_i \in V$. A spatial relational vector is chosen at random for the first edge, and the others have their vectors derived to keep the graph spatially coherent (see chapter 1.5).

To remove a vertex v_i from G it is simply a matter of removing the vertex from V and all edges in E that connects to it.

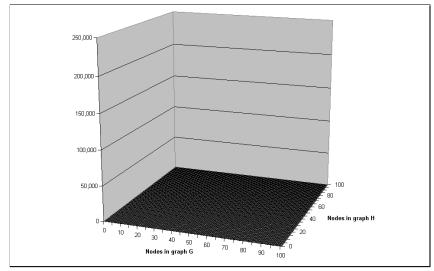
The percentage p is a measurement of how many vertices should be added or removed. As with spatial noise, this value is chosen from the range $[-\frac{1}{2}p, \frac{1}{2}p]$, scaled to the current experiment. Since our experiments handle graphs that have 1 to 100 vertices, our scale factor is 100.

Because there is no noise to neither the pattern primitives of the vertices, nor the spatial information in the edges, these instances are very simple for our algorithm to solve. The original vertices in the candidate graph will be matched correctly to the corresponding prototype graph; excessive vertices in the candidate is understood as the prototype graph being a subgraph of the candidate graph, the opposite is true when vertices have been removed.

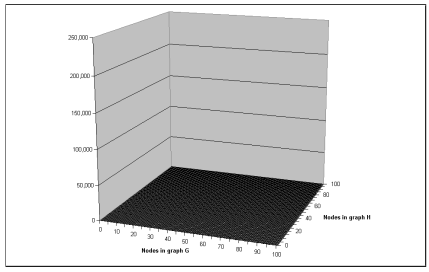
The only reasons for a mismatch to occur under this type of noise is if 1) there exists a prototype graph that is itself a subgraph of another prototype graph, or 2) all additional vertices are identical, both their pattern primitives and their spatial relationships, to some vertices in another prototype graph, such that the new vertices become a larger subgraph to the wrong prototype graph than the original graph to the correct prototype. Instances where the first may occur should be remedied by marking both prototype graphs as being equal, whereas the second is so fantastically improbable that it is not worth considering.



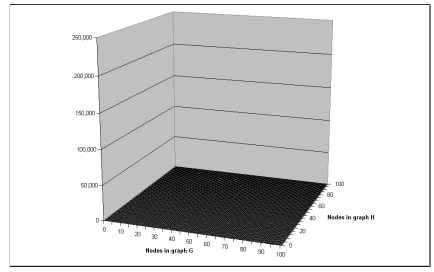
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.24, refinements 821.10.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

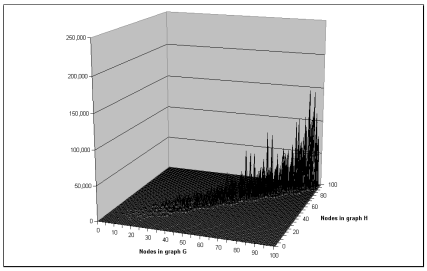
Figure 4.20: Match with 10% vertex noise.

10% vertex noise

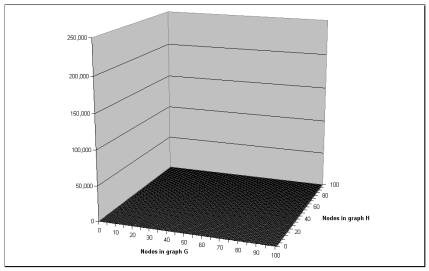
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

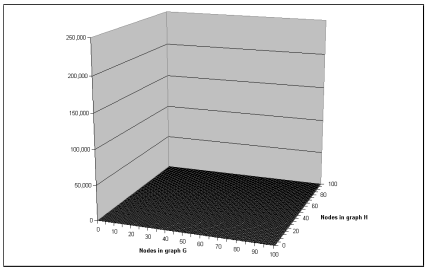
Results are perfect, with no additional time spent on anything except for the True-Positive matches.



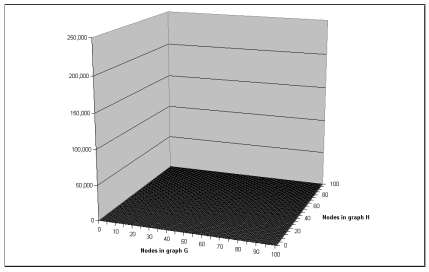
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.23, refinements 820.88.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

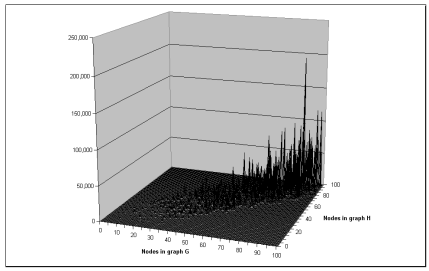
Figure 4.21: Match with 25% vertex noise.

25% vertex noise

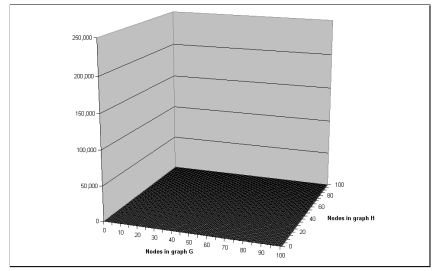
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

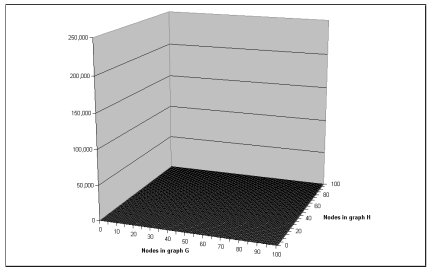
Results are perfect at minimum time.



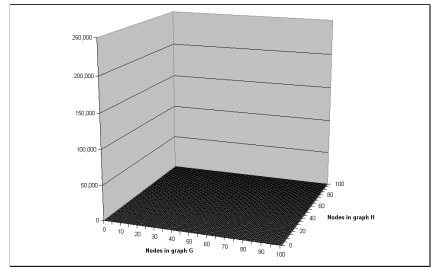
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.22, refinements 822.06.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

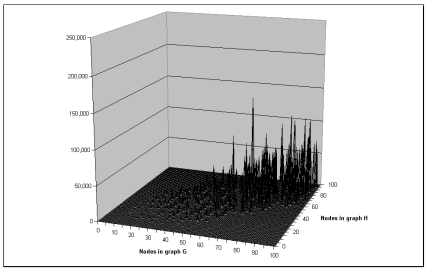
Figure 4.22: Match with 50% vertex noise.

50% vertex noise

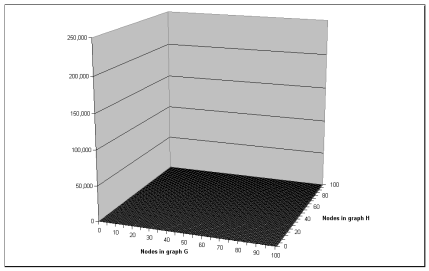
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

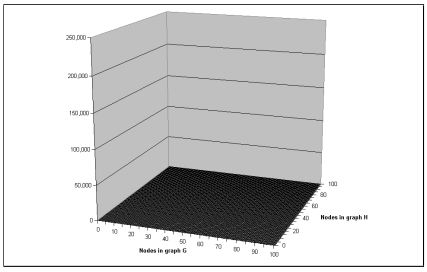
Again, perfect results at minimum time.



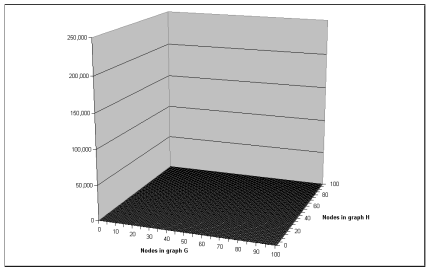
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.21, refinements 820.59.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

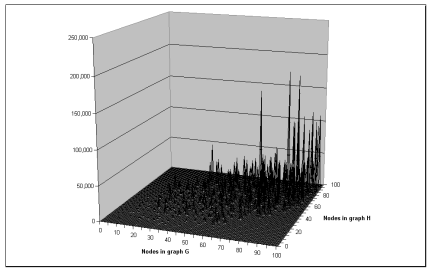
Figure 4.23: Match with 75% vertex noise.

75% vertex noise

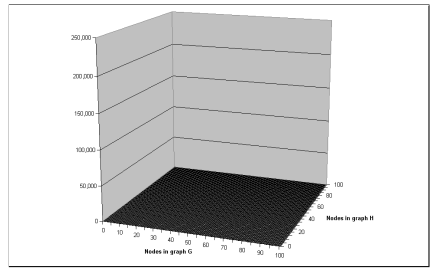
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

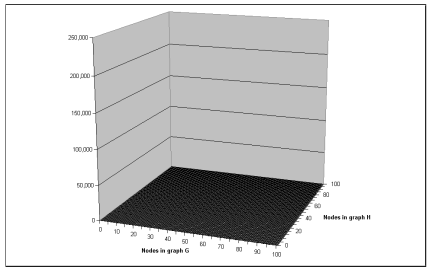
And again, perfect.



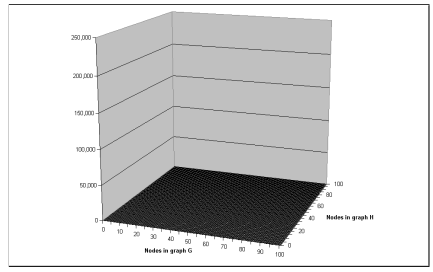
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.20, refinements 822.09.



(b) True-Negative. Count 20,000, depth 0.00, recursions 0.00, refinements 0.00.



(c) False-Positive. None.



(d) False-Negative. None.

Figure 4.24: Match with 95% vertex noise.

95% vertex noise

Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

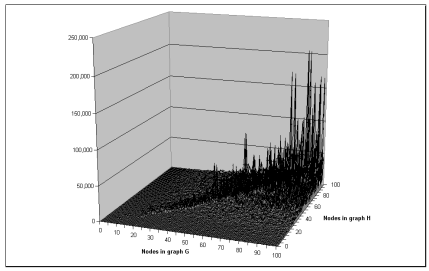
Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

And finally, perfect results even at this amount of noise.

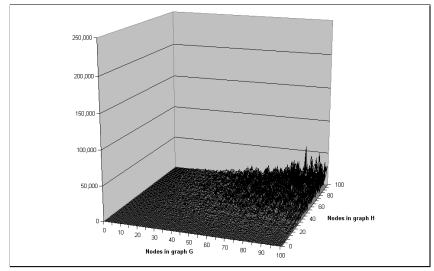
4.3.4 Combinations of noise

We show earlier in this chapter how the application of a single type of noise to a candidate graph can be efficiently resisted by clever manipulation of the controlling parameters. For perfect results, however, it does requires that we know the type of noise applied.

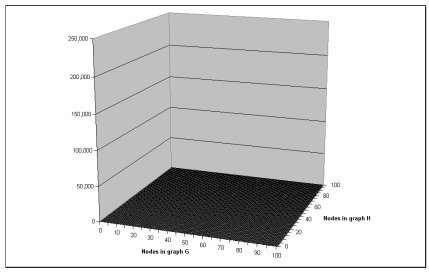
In this chapter we apply different combinations of noise, and then investigate the accuracy of the matcher and the time required. This means that there is no simple way for the algorithm to solve the matching instances, and the controlling parameters need to simultaneously accept both noisy vertex-vertex associations and edge-edge associations.



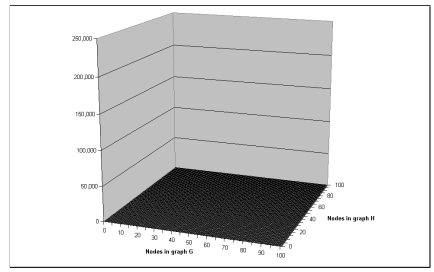
(a) True-Positive. Count 40,000, depth 0.53, recursions 1.17, refinements 4,391.61.



(b) True-Negative. Count 19,800, depth 0.52, recursions 0.95, refinements 3,712.77.



(c) False-Positive. Count 200, depth 0.20, recursions 0.22, refinements 13.87.



(d) False-Negative. None.

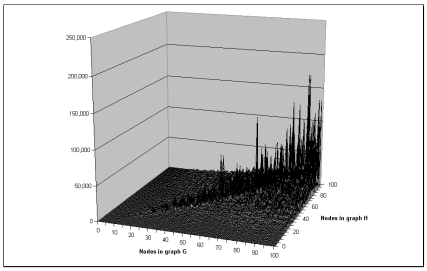
Figure 4.25: Match with 10% spatial and vertex noise.

10% spatial and vertex noise

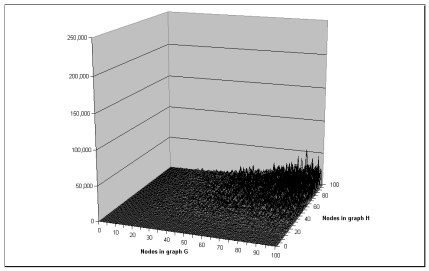
Accuracy	Error	Precision	Recall	F1
1.00	0.00	1.00	1.00	1.00

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{20}|A| - \frac{9}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

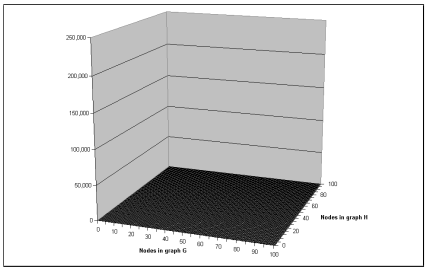
The results are perfect. This agrees with the observation in chapter 4.3.3 that vertex noise is close to trivial, the results equal those in chapter 4.3.1.



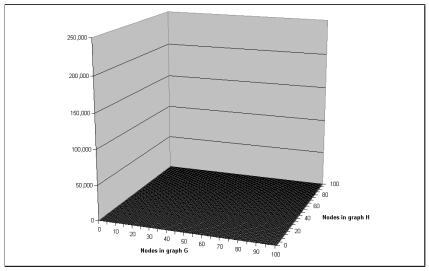
(a) True-Positive. Count 38,600, depth 0.53, recursions 1.18, refinements 4,480.88.



(b) True-Negative. Count 20,000, depth 0.51, recursions 0.90, refinements 3,485.60.



(c) False-Positive. None.



(d) False-Negative. Count 1,400, depth 0.16, recursions 0.17, refinements 5.46.

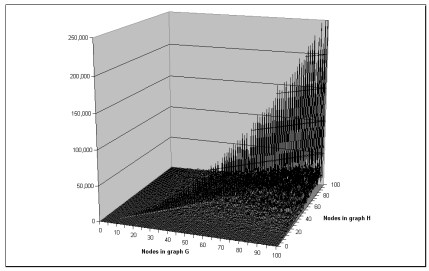
Figure 4.26: Match with 10% pattern and vertex noise.

10% pattern and vertex noise

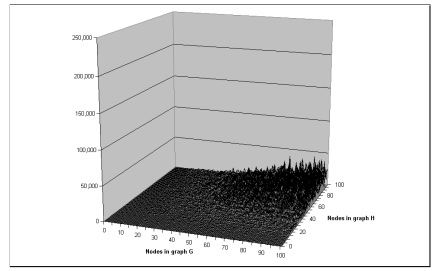
Accuracy	Error	Precision	Recall	F1
0.98	0.02	1.00	0.97	0.98

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{20}|A| - \frac{9}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

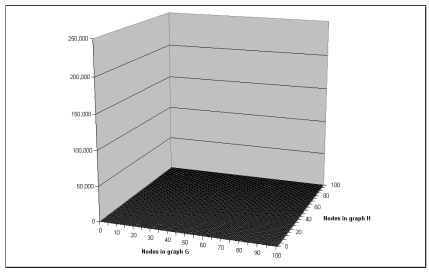
The results are close to perfect, although some incorrect unassigned matches occur. Again, this is similar to the results achieved when disregarding the vertex noise.



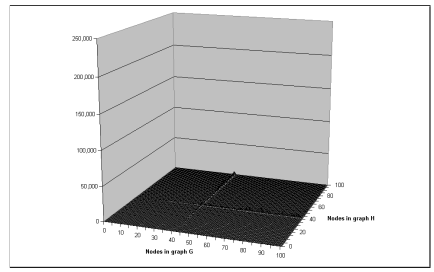
(a) True-Positive. Count 39,400, depth 0.53, recursions 1.15, refinements 4,358.32.



(b) True-Negative. Count 20,000, depth 0.51, recursions 0.91, refinements 3,520.71.



(c) False-Positive. None.



(d) False-Negative. Count 600, depth 0.29, recursions 0.38, refinements 568.27.

Figure 4.27: Match with 10% spatial and pattern noise.

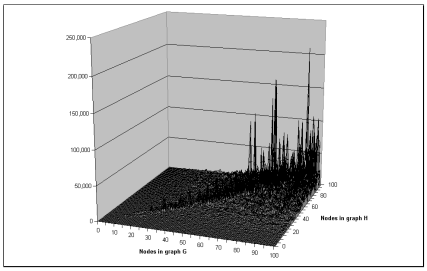
10% spatial and pattern noise

Accuracy	Error	Precision	Recall	F1
0.99	0.01	1.00	0.99	0.99

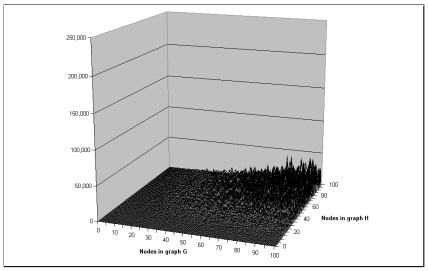
Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{20}|A| - \frac{9}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

These instances are by far the most interesting ones. The spatial content of the edges in the candidate graph, as well as the pattern primitive data in the vertices, are distorted by noise. The algorithm is unable to rely on any type of data as perfect, and the matching becomes far from trivial. Still, the only error that occurs are a few instances of incorrect unassigned matches. By the low average depth, recursion and refinement count we assume that these are instances of smaller graphs.

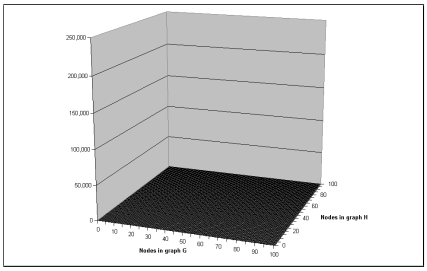
The results are still very close to perfect.



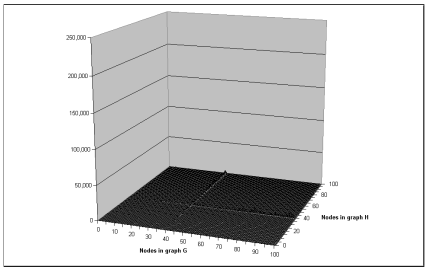
(a) True-Positive. Count 38,200, depth 0.54, recursions 1.17, refinements 4,522.05.



(b) True-Negative. Count 20,000, depth 0.51, recursions 0.91, refinements 3,561.70.



(c) False-Positive. None.



(d) False-Negative. Count 1,800, depth 0.21, recursions 0.26, refinements 186.28.

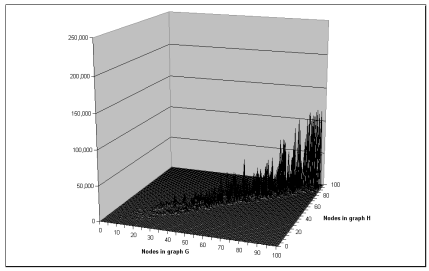
Figure 4.28: Match with 10% spatial, pattern and vertex noise.

10% spatial, pattern and vertex noise

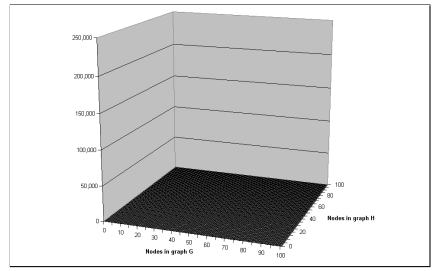
Accuracy	Error	Precision	Recall	F1
0.97	0.03	1.00	0.96	0.98

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{20}|A| - \frac{9}{20}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

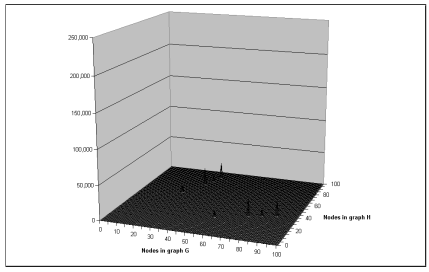
The results are close to perfect. The vertex noise seems to primarily be effecting the smaller graphs, an observation supported by comparison to the results of spatial and pattern noise – the addition of vertex noise only adds a few more instances of False-Negatives with a low average refinement count.



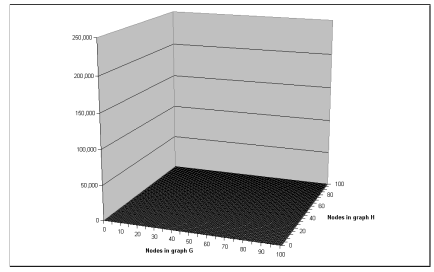
(a) True-Positive. Count 40,000, depth 0.01, recursions 0.23, refinements 835.03.



(b) True-Negative. Count 19,200, depth 0.00, recursions 0.00, refinements 0.26.



(c) False-Positive. Count 800, depth 0.02, recursions 0.08, refinements 180.17.



(d) False-Negative. None.

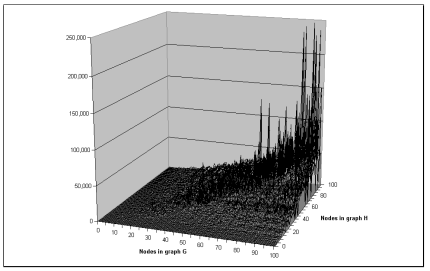
Figure 4.29: Match with 25% spatial and vertex noise.

25% spatial and vertex noise

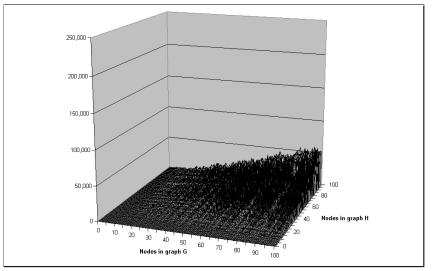
Accuracy	Error	Precision	Recall	F1
0.99	0.01	0.98	1.00	0.99

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = \frac{1}{10}|A| - \frac{1}{2}$, the average cost threshold is $\varepsilon = 0$.

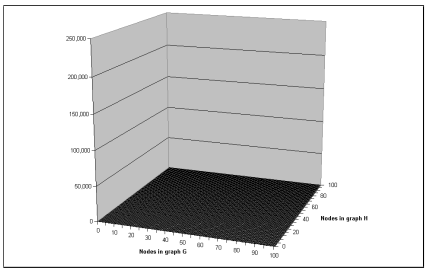
Vertex noise does not contribute to much confusion when combined with spatial noise alone, and the results are very close to perfect.



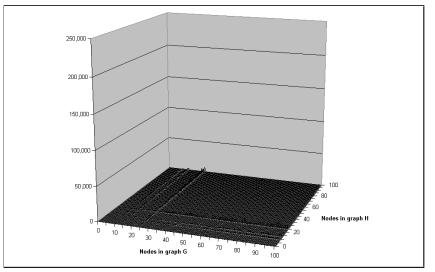
(a) True-Positive. Count 37,600, depth 1.00, recursions 3.62, refinements 12,284.69.



(b) True-Negative. Count 20,000, depth 0.96, recursions 3.26, refinements 10,513.73.



(c) False-Positive. None.



(d) False-Negative. Count 2,400, depth 0.66, recursions 1.37, refinements 452.99.

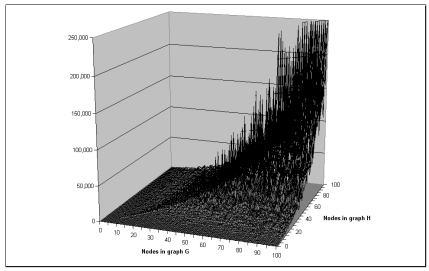
Figure 4.30: Match with 25% pattern and vertex noise.

25% pattern and vertex noise

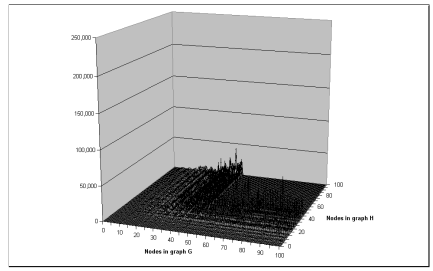
Accuracy	Error	Precision	Recall	F1
0.96	0.04	1.00	0.94	0.97

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The dynamic association threshold is given by the function $\epsilon = -\frac{1}{10}|A| - \frac{4}{10}$, the average cost threshold is $\varepsilon = -\frac{1}{2}$.

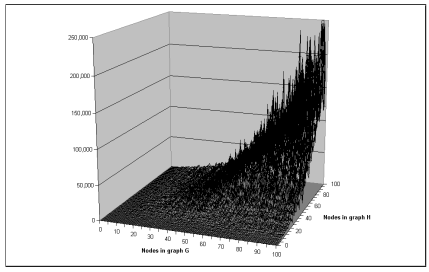
Again, the vertex noise does not shift the results from those in chapter 4.3.2. Precision remains perfect, whereas Recall is down to 0.94, giving a total F1 value of 0.97.



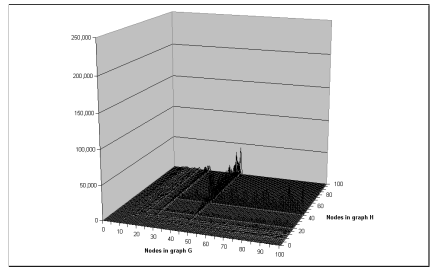
(a) True-Positive. Count 29,400, depth 1.64, recursions 6.09, refinements 24,034.78.



(b) True-Negative. Count 3,200, depth 1.06, recursions 2.86, refinements 3,971.47.



(c) False-Positive. Count 24,800, depth 1.65, recursions 6.21, refinements 27,045.84.



(d) False-Negative. Count 2,600, depth 0.79, recursions 1.89, refinements 1,957.12.

Figure 4.31: Match with 25% spatial and pattern noise.

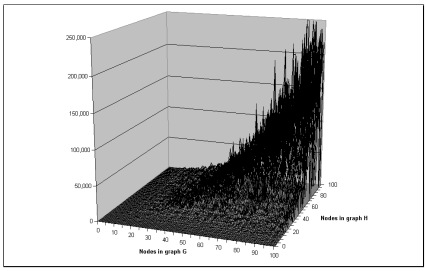
25% spatial and pattern noise

Accuracy	Error	Precision	Recall	F1
0.54	0.46	0.54	0.92	0.68

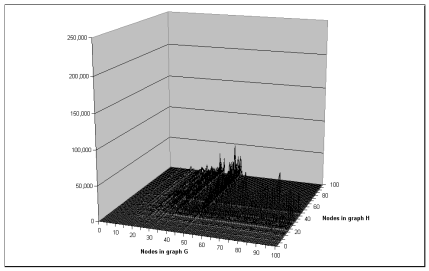
Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The association threshold is $\epsilon = -\frac{2}{5}$, the average cost threshold is $\varepsilon = -\frac{2}{5}$.

By combining both 25% pattern and 25% spatial noise, the algorithm is up against its hardest challenge. The runtimes are among the highest recorded in all our experiments, but the results are very good. In contrast to earlier results Recall is a lot higher than Precision, which would imply that the combined noise has been able to push candidate graphs sufficiently far from their original description into something that more closely resembles another graph in the set of prototypes.

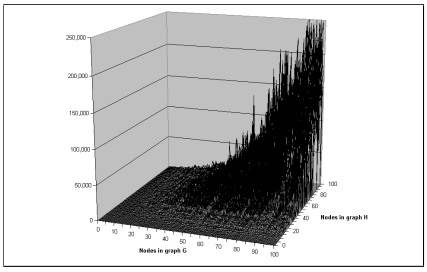
Keep in mind that every single pattern primitive and every single edge has been allowed to move around in 25% of its maximum extent. Furthermore, remember that even the highest peaks of these runtimes are still only a fraction of the equivalent brute-force runtimes.



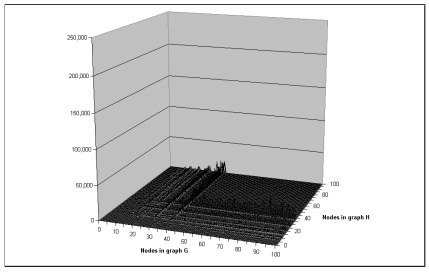
(a) True-Positive. Count 31,200, depth 1.68, recursions 6.36, refinements 26,257.49.



(b) True-Negative. Count 2,200, depth 1.17, recursions 3.35, refinements 5,037.01.



(c) False-Positive. Count 24,200, depth 1.57, recursions 5.80, refinements 24,609.12.



(d) False-Negative. Count 2,400, depth 0.90, recursions 2.28, refinements 2,410.79.

Figure 4.32: Match with 25% spatial, pattern and vertex noise.

25% spatial, pattern and vertex noise

Accuracy	Error	Precision	Recall	F1
0.56	0.44	0.56	0.93	0.70

Target depth is 3, low-level cut-off is 5, high-level cut-off is 50. The association threshold is $\epsilon = -\frac{2}{5}$, the average cost threshold is $\varepsilon = -\frac{2}{5}$.

The results are similar to the instances without the vertex noise; an impressive F1 at the price of "high" runtimes.

Chapter 5

Conclusion, part 1

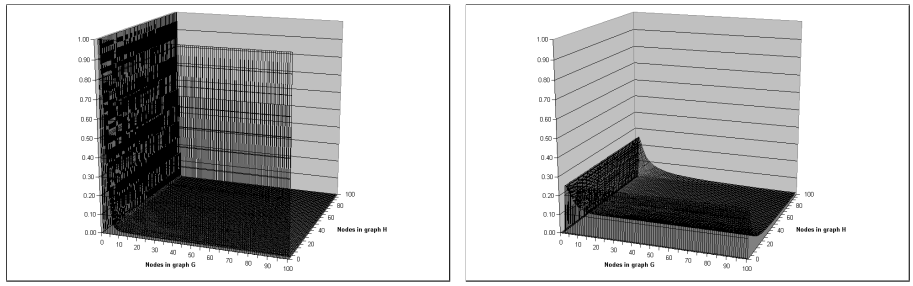
This concludes our second contribution. This part illustrates in detail the benefits of our approach over traditional methods of graph matching. Not only does our algorithm improve on the increasing size of the graphs involved, but it is developed in such a way that future heuristics can quickly and easily be deployed and tested.

Chapter 1 introduces the field of computer vision by a discussion of various applied methods. The attributed relation graph is introduced, and the problem at the heart of our thesis is derived:

> **MINIMUM COST SUBGRAPH ISOMORPHISM**
> INSTANCE: Given graphs $G = (V_1, E_1)$, $H = (V_2, E_2)$ where $|V_1| \leq |V_2|$, a vertex cost metric $Ev(v_i, v_j)$ for associating a vertex $v_i \in V_1$ to a vertex $v_j \in V_2$, and an edge cost metric $Ee(e_k, e_l)$ for associating an edge $e_k \in E_1$ to an edge $e_l \in E_2$.
> QUESTION: Under these metrics, what is the minimum cost subgraph isomorphism from graph G to graph H?

Runtimes are presented as two-fold values (the cost of Ev as α, the cost of Ee as β), and the remainder of the chapter 1 is used to demonstrate how real instances of the problem, as available in vision tasks, can be used to deduce bounds on runtimes which are only a fraction of the originals (see figure 5.1).

In chapter 2 we describe our algorithm for solving instances of MINIMUM COST SUBGRAPH ISOMORPHISM, and in chapter 3 the algorithm is illustrated with complete detail. Then, in chapter 4, we present results from a set of 1,380,000 matches performed on random graphs that are distorted using various types of noise. See figure 5.2 for the incredible improvements achieved by applying



(a) Magnitude of α component over original α.

(b) Magnitude of β component over original β.

Figure 5.1: Improvements by real instances, see chapters 1.5 and 1.6.

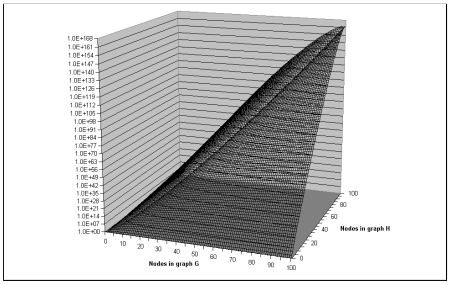
common pruning ideas to our algorithm.

There is almost no loss of accuracy as runtimes are reduced from 158-digit numbers to 3-digit ones, even under heavy noise. Still, our results should be regarded as contemporary, since future techniques and ideas for pruning search space can be readily added to our algorithm as implementation details.

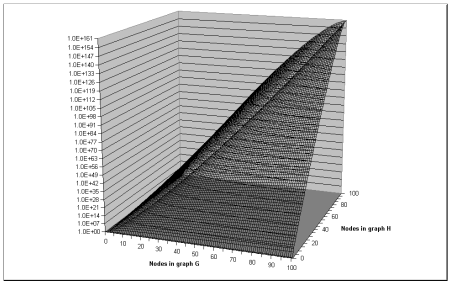
Although the trend in computer vision research seems to favour other approaches than graph matching, our algorithm suggests that ARG's can be used with high accuracy and speed. Contrary to traditional graph matching, our algorithm thrives on large, fully-connected graphs, instances that most other graph matching algorithms fail horribly to solve. The accuracy of our method actually increases as the graphs increase in size, while runtimes only gracefully degenerate.

We have achieved milestone 4; to develop a solution to subgraph isomorphism that manages sufficient accuracy in lower time-complexity than other published methods. The comparison is obviously biased by the fact that we only trial spatially coherent graphs – instances that our algorithm thrives on, while other methods typically avoid to solve them alltogether because of their complexity. However, in light of the trend that we reveal in chapter 1 this is the only correct comparison.

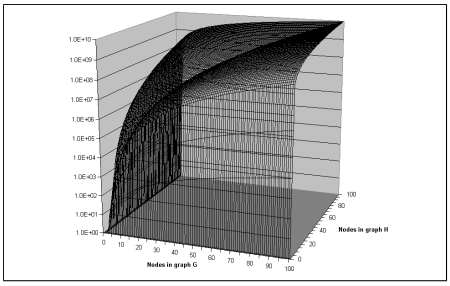
Any future work should includes the development of better vertex- and edge-association heuristics and additional rejection tests.



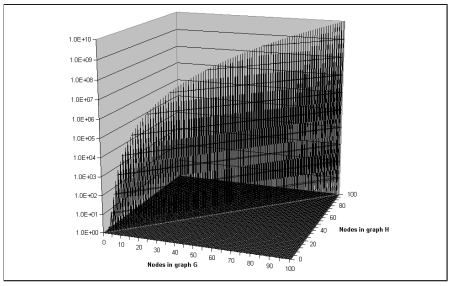
(a) By brute-force. Average number of calls to Ee is 4.08E+158.



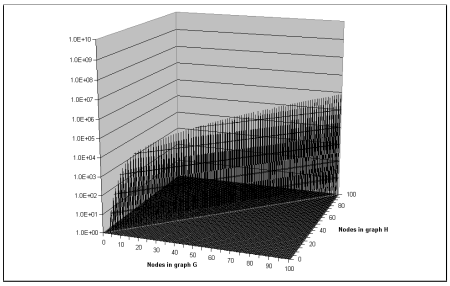
(b) By spatial coherent graphs. Average number of calls to Ee is 4.09E+156.



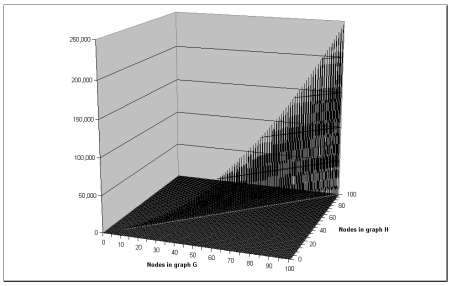
(c) By depth limitation. Average number of calls to Ee is 1.35E+9.



(d) By depth limitation and association threshold. Average number of calls to Ee is 1.58E+7.



(e) By depth limitation, association threshold and low-level cut-off. Average number of calls to Ee is 2,494.40.



(f) By depth limitation, association threshold, low- and high-level cut-off. Average number of calls to Ee is 820.96.

Figure 5.2: Improvements by pruning, see chapters 1 through 4.

Part II

Application to computer vision

Chapter 6

Object definition

In part 1 we introduce the attributed relation graph (ARG) and propose an algorithm to solve the MINIMUM COST SUBGRAPH ISOMORPHISM problem. In this part we apply that solution to a computer vision system.

All papers, as seen by us, detailing nearest neighbour indexing of patterns in computer vision comment on the "curse of dimensionality" as their feature vectors (similar to our pattern primitives) grow to hold more and more information. As more types of pattern primitives are discovered and added to the feature vectors, the faster these methods break down due to a high-order space that becomes near impossible to partition efficiently.

Instead of adding more and more pattern primitives to a single-graph description of an object, we propose the use of a set of graphs that each hold low-complexity primitives.

Let an object $O = (G_1, G_2, .., G_N)$ be a set of N ARG's, where N is the number of different types of pattern primitives available to describe an image of the object. Each graph G_i is a separate and complete description of the object O using pattern primitive type i, or rather; the graph G_i describes the object O in domain i.

By this division of domains the upper-bound complexity of the search space is effectively limited to the highest-order pattern primitive, but it also requires that matching has to be performed separately in each domain. As we detail in chapter 4, this graph matching can be done fast and efficiently by our algorithm.

Although the following example is not accurate, it does let us illustrate the idea.

Let the number of known domains N be 4. This gives four separate ARG's that each describe the same input image, but over separate domains. For simplicity,

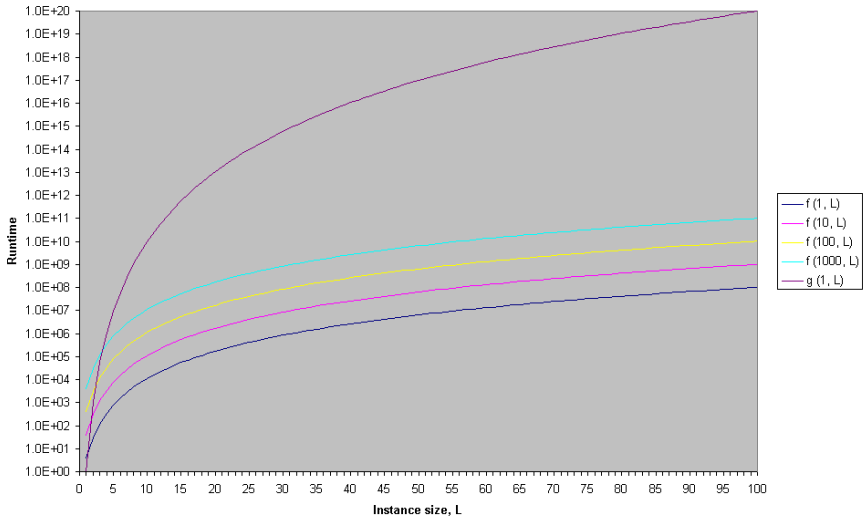


Figure 6.1: Hypothetical runtime comparison.

let us assume that all domains are based on regions detected in the image, thereby making all four ARG's contain the same number of vertices L. Furthermore, let the number of values required to describe a pattern primitive be 1, 2, 3 and 4 for the four domains respectively.

Our method needs to run four separate matches; time is then the sum of 4 separate matches, bound by some constant a and the respective complexities:

$$f(a, L) = a \cdot L^1 + a \cdot L^2 + a \cdot L^3 + a \cdot L^4 \tag{6.1}$$

As opposed to this, an algorithm that combines the corresponding pattern primitives of each domain to a single description is similarly bound by:

$$g(b, L) = b \cdot L^{1+2+3+4} \tag{6.2}$$

See figure 6.1 for the obvious benefits of a function such as $f(a, L)$ over $g(b, L)$. Only when L is small will $g(b, L)$ outperform $f(a, L)$; however, the runtimes in such cases are generally so small that they do not matter anyway.

Keep in mind that this is a demonstration of principle, it is not a set of recorded runtimes.

Chapter 7

Object extractor

In this chapter we present a method of extracting an object (see chapter 6) from any 2-dimensional image. There is some implementation-specific details included in the description of our object extractor, relevant only when read alongside the source-code that accompanies this thesis.

This chapter is based, in part, on the method of feature extraction presented in [6].

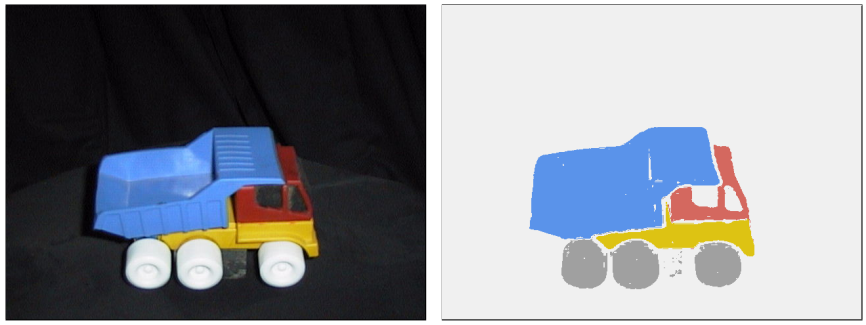
7.1 Image acquisition

Image acquisition is regarded as an implementation detail. This section details how this is achieved using C# code (a proprietary programming language developed by Microsoft, see [54]) running in a Microsoft .Net environment (see [55]).

Loading an image from file uses the Microsoft .Net class $System.Drawing.Bitmap$ capable of loading most known image file types by passing the filename as argument to the class' constructor. Although sampling and quantization has been performed beforehand, the image is moved into a 24-bit surface.

With native Microsoft Windows code called through a few, simple lines C# code, it is possible to capture an image from any digital camera connected to the computer. Both sampling and quantization is determined by the camera, but again the image is available in a 24-bit surface.

With the release of the first managed version (9.0) of the DirectX API (see [56]) in December 2002 it is now possible to use it with any CLR-compliant (see [55] and [57]) language. Even though the OpenGL environment (see [58])



(a) Original image. (b) Classified image.

Figure 7.1: Image classification by YUV space.

boasts platform independence, it does not supply the simplicity of DirectX when running through Microsoft .Net; the namespace $Microsoft.DirectX.Direct3D$ supplies the required classes to load and render any .x mesh file (see [59]). To capture an image of a 3d mesh it is simply a matter of supplying the mesh file and a viewpoint. Spatial resolution can be arbitrarily chosen by the user, while quantization is again done in 24 bits.

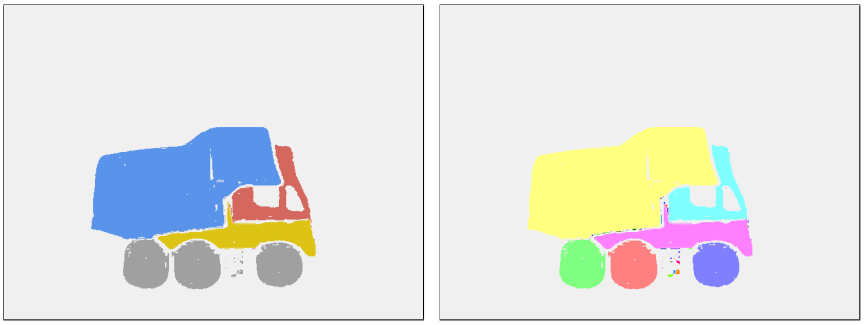
7.2 Preprocessing

In the preprocessing step, the image is being treated with "low-level"-operations. The aim of this step is to do noise reduction on the image (i.e. to dissociate the signal from the noise) and to reduce the overall amount of data. We do this through (1) colour pixel classification, (2) image segmentation, (3) region thresholding, and finally (4) object cropping.

7.2.1 Colour pixel classification

The first step to colour vision is to classify each pixel in an image into one of a discrete number of classes. The approach chosen is to use a set of constant thresholds to define a colour class as a rectangular block in the colour space (see [60]). As noted in [61], RGB space is not particularly suited for rectangular blocks since relations such as "orange" imply a conical volume in that space. In contrast YUV space encodes colour in Y and U, while intensity is encoded in V, making it more appropriate for this method.

Even in YUV there might be issues with ill-formed classes that simple blocks fail to correctly classify. To remedy this issue, each class is instead assigned a hand-



(a) Classified image. (b) Segmented image.

Figure 7.2: Image segmentation by labeling.

picked non-unique RGB value. This RGB value is used as the class identifier, allowing several blocks in YUV space to identify the same class. Arbitrarily shaped classification volumes are thus simply a set of one or more rectangular blocks.

Figure 7.1 shows how this simple, yet powerful classification scheme works.

7.2.2 Image segmentation

Image segmentation is done by scanning the classified image and label each disjoint region with a unique label. Since the image is scanned from top to bottom, left to right, region connectivity might be encountered after already assigning different labels to pixels in the same region. These label equivalences are recorded at discovery, and every pixel is relabeled in a second pass.

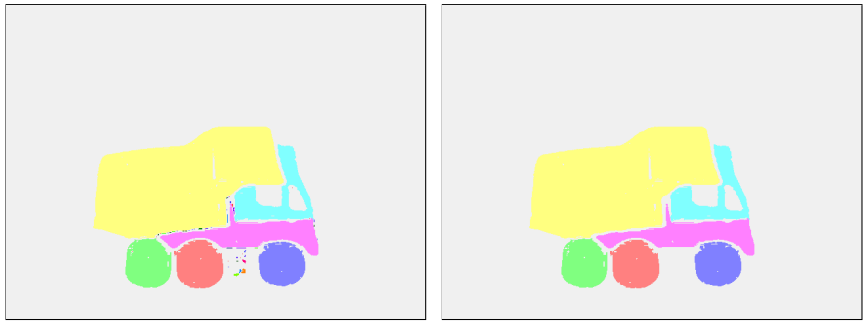
The labeling is performed using an extended implementation of connected-component labeling (see [62]). The algorithm was originally developed for binary images, and works by assigning labels to each pixel such that adjacent pixels of the same class are assigned the same label. The term "adjacency" can be any definable connectivity. We have chosen to use 4-connectivity as neighborhood scheme.

The image is scanned from top to bottom, left to right. When examining a particular pixel A, we know that the cell to its left, B, has already been labelled, as has the cell C directly above A. The steps to label pixel A in a non-binary image is listed as algorithm 4, a slightly modified version of the binary algorithm that appears in [62].

By applying connected-component labeling to the classified images, the system manages to identify any disjoint regions. Figure 7.2 is a pseudo- coloured example of this segmentation.

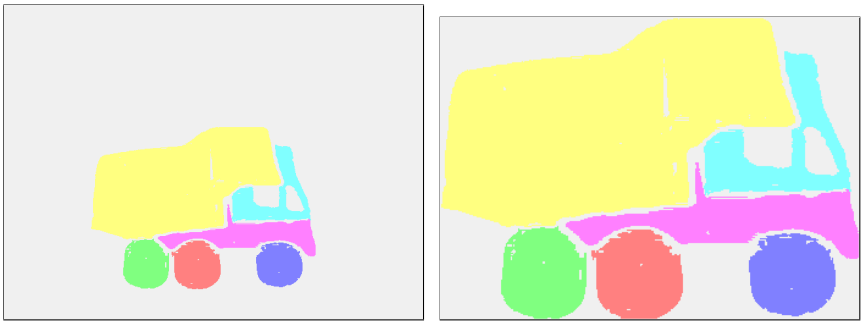
Algorithm 4 Extended connected-component labeling

```
1: if A class = 0 then
2:    Do nothing
3: else if (not B labeled) and (not C labeled) then
4:    Increment label numbering
5:    Label A
6: else if B xor C labeled then
7:    if (B labeled) and (B class = A class) then
8:       Copy label to A
9:    else if (C labeled) and (C class = A class) then
10:      Copy label to A
11:   end if
12: else if B and C labeled then
13:   if (B class = A class) and (C class = A class) then
14:      Copy either B label or C label to A
15:      Record equivalence of labels
16:   else if (B class = A class) then
17:      Copy label to A
18:   else if (C class = A class) then
19:      Copy label to A
20:   else
21:      Increment label numbering
22:      Label A
23:   end if
24: end if
```



(a) Labeled image. (b) Thresholded image.

Figure 7.3: Detail of region thresholding.



(a) Thresholded image. (b) Cropped image.

Figure 7.4: Details of object cropping.

7.2.3 Region thresholding

To remove any noise or disjoint residue from previous steps in the preprocessing we apply thresholding on the labeled regions. Every region is compared to the largest area in the image, and if it is below some fraction of this it is discarded as noise. We avoid thresholding against the actual image size since it has not yet been cropped to the detected object. An example of this thresholding is shown in figure 7.3.

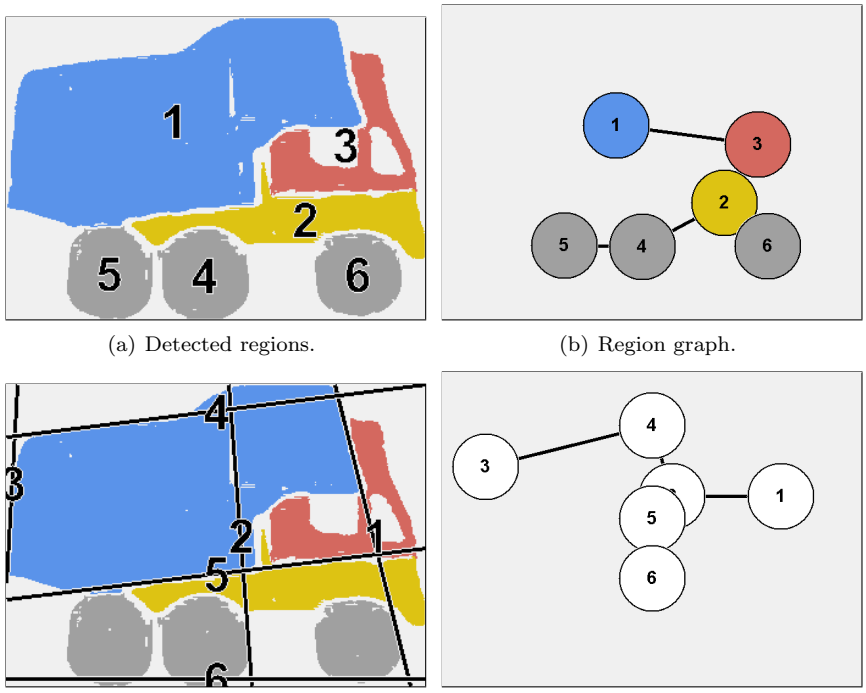
7.2.4 Object cropping

The final step in our preprocessing is to crop the image to the object. Because the previous steps remove noise, residue and uninteresting regions from the input image, the crop is easily able to locate the object. The dimensions after this crop is referred to as the object-dimensions, and can be used to scale extracted features to achieve scale-invariance.

7.3 Graph extraction

The aim of feature extraction is to further reduce the preprocessed images to a set of features. The extraction is run separately for each available domain, and each domain i produces a separate graph $G_i \in O$.

When presenting an object to the reader, we will use a set of four images similar to those in figure 7.5. Because three of the four available domains are concerned with the regions in the image, these three are equally expressed by the first two images. The spatial relationships perceivable in these images are contained in



(a) Detected regions. (b) Region graph.

(c) Detected lines. (d) Line graph.

Figure 7.5: Example of graph illustration.

the edges of the graphs (and therefore spatially coherent). Edges are added to the graphs using a nearest-neighbour scheme until every vertex is connected.

7.3.1 Domain 1: Region mass

The moment of order $(p+q)$ of a discrete function $f(x,y)$ is defined as

$$m_{pq} = \sum_{x,y} x^p y^q f(x,y) \tag{7.1}$$

which is a summation over a domain that includes all nonzero values of the function. To distinguish between the various regions that appear in $f(x,y)$ it can either (1) be separated into one function per region, or (2) wrapped using a function $g(x,y)$ defined as

$$g(x,y) = \begin{cases} 1, & \text{iff } f(x,y) \text{ is labeled as region } l; \\ 0, & \text{otherwise.} \end{cases} \tag{7.2}$$

where l is implied by context.

The total mass of region is then given by the moment m_{00}. The label of the region is assumed as context for $g(x,y)$. To achieve a common scale for all regions in the same image, it is scaled by a constant k given by the image dimensions I_{Width} and I_{Height} as

$$k = \frac{1}{I_{Width} \cdot I_{Height}} \tag{7.3}$$

All regions in the image are represented by separate vertices in the extracted graph, and the regions' mass is assigned as attributes to the corresponding vertices (the value $k \cdot m_{00}$ by the vertex's region-label).

The centroid $(\overline{x}, \overline{y})$ for a vertex is

$$\overline{x} = \frac{1}{m_{00}} \sum x f(x,y) = \frac{m_{10}}{m_{00}} \tag{7.4}$$

$$\overline{y} = \frac{1}{m_{00}} \sum y f(x,y) = \frac{m_{01}}{m_{00}} \tag{7.5}$$

by that vertex's region-label.

7.3.2 Domain 2: Region colour

Region colour is retrieved by a single pass over the labeled image; from top to bottom, left to right. Whenever a new label is encountered, a vertex is added to the graph, defined by the symbolic colour given by that region's class. Because the regions are the same as for mass, the centroid $(\overline{x}, \overline{y})$ is also the same.

7.3.3 Domain 3: Region moments

In [63], the authors describe a method of absolute moment invariants. These expressions are derived from algebraic invariants applied to the moment generating function under a rotation transformation. They consist of groups of nonlinear centralised moment expressions. The result is a set of absolute orthogonal (i.e. rotation) moment invariants, which can be used for scale, position, and rotation invariant pattern identification.

The definition of a discrete centralised moment as described by Hu is:

$$\mu_{pq} = \sum_{x=1}^{M}\sum_{y=1}^{N}(x-\overline{x})^p(y-\overline{y})^q f_{xy} \tag{7.6}$$

This is essentially a translated Cartesian moment, which means that the centralised moments are invariant under translation. To enable invariance to scale, two dimensional scale-normalised centralised moment are used (see [64]), given by:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{7.7}$$

where

$$\gamma = \frac{p+q}{2} + 1 \qquad \forall p+q \geq 2 \tag{7.8}$$

Hu's first four absolute moment invariants are computed from normalised centralised moments up to order three and are shown below:

$$I_1 = \eta_{20} + \eta_{02} \tag{7.9}$$
$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{7.10}$$
$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{7.11}$$
$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{7.12}$$

Each vertex in the moment graph contains all of these four moments. Again, the centroid $(\overline{x}, \overline{y})$ remains the same as for mass.

7.3.4 Domain 4: Dominant lines

Unlike the previous features extracted from the image that relate to the regions of the image, this domain is concerned with dominant lines. This is done by (1) edge detection by application of a Sobel operator, (2) transformation of that edge-image to Hough space, and (3) projection of the peaks of Hough space back into the original image to retrieve scalable parameters.

The Sobel operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or

"smoothly" the image changes at that point, and therefore how likely it is that that part of the image represents an edge, as well as how that edge is likely to be oriented. In practice, the magnitude (likelihood of an edge) calculation is more reliable and easier to interpret than the direction calculation.

Mathematically, the gradient of a two-variable function (here the image intensity function) is at each image point a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction. This implies that the result of the Sobel operator at an image point which is in a region of constant image intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.

The operator uses two 3x3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. These are defined as:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{7.13}$$

At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G}_{xy} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2} \tag{7.14}$$

The magnitude is evaluated and thresholded separately for the red, green, and blue component, then combined to yield a binary edge-image. This image is then transformed into Hough space to fit lines onto it.

A Hough transform is a mapping from an observation space into a parameter space. In computer vision, observation space could be a digital image, an edge map etc. Now assume that a certain structure is thought to be present in image space. For an edge map, this could be a straight line or a circle. The parameters of the structure define parameter space (gradient and intercept for a line, radius and centre coordinates for a circle). In a Hough transform, each point in image space "votes" for that part of parameter space which describes structures which include the point. A part of parameter space receiving a large number of votes corresponds to a possible fit.

In the normal Hough transform approach, parameter space is bounded by setting lower and upper limits on the parameter values, and then divided into blocks in each direction, and an accumulator assigned to each block. The Hough transform proceeds with each point in image space being transformed to an region

in parameter space as described in the previous paragraph. When the region intersects one of the blocks, the corresponding accumulator is incremented. The block whose accumulator has the most votes can then be taken as the best fit of the structure to the image points, the values of the parameters usually being calculated at the centre of the block.

To find the peaks in Hough space we employ a sliding neighborhood that looks for local maxima. As the neighborhood is moved across the accumulator, a local threshold is used to avoid marking the borders of the neighborhood as maxima. The nature of Hough space makes this an important threshold, since it is common to see ridges through the image leading to more global maxima. A global threshold is also used to avoid labeling the low-intensity peaks that tend to exist throughout Hough space.

For each peak found in parameter space, it is projected back into the original image to find the corresponding line's entry- and exit point. The fourth image in figure 7.5 illustrates how these lines transform back in the original image. The projection is necessary to properly represent the information of the peaks in Hough space. The two parameters expressed in Hough space alone are not easily comparable, since the Euclidian distance in Hough space does not necessarily detail the lines' relation in the original image.

To achieve scale invariance for these points they are scaled using the larger of the dimensions of the image; i.e. $\mathbf{max}(I_{Width}, I_{Height})$. This has the added benefit of actually encoding the aspect ratio of the image into the lines.

Each vertex in the line graph contains these scaled entry- and exit points of the referred line.

Chapter 8

Object matcher algorithm

In this chapter we propose a method of finding the closest match to an unknown object $O = (G_1, G_2, .., G_N)$ from a set of known objects L.

Algorithm 5 matches the object O to every object $P_i = (H_{i1}, H_{i2}, .., H_{iN})$ in L by solving MINIMUM COST SUBGRAPH ISOMORPHISM separately in each of the N domains. The solution to each subgraph isomorphism problem solved is stored in a matrix C. Each row i, $c_{i1}..c_{iN}$, corresponds to an object, and each column j, $c_{1j}..c_{|L|j}$, corresponds to a domain.

When every object $P_i \in L$ have been attempted matched to O, each column j is normalized to the range $[0, 1]$. This is analogous to constructing a unit-extent N-dimensional hypercube, in which every object P_i is located by the row-vector $c_{i1}..c_{iN}$, and the unknown object O lies at origin. Therefore; the object P_i whose location lies closest to origin, is the match to O.

This approach to object matching imposes no partitioning or control on the underlying graph matcher. Complete confidence is given to our inexact graph matcher (see part 1) – at this point we simply construct a series of instances of MINIMUM COST SUBGRAPH ISOMORPHISM and record the solutions found for each.

Algorithm 5 Object matcher

1: Allocate a cost matrix C of size $|L| \times N$.
2: **for all** object $P_i \in L$ **do**
3: **for** $j = 1$ to N **do**
4: Solve MINIMUM COST SUBGRAPH ISOMORPHISM for graph G_j and H_{ij}, store minimum cost in c'.
5: $c_{ij} \leftarrow c_{ij} + c'$
6: **end for**
7: **end for**
8: Normalize all N columns of C to the range $[0, 1]$.
9: Find row i_{Min} that is the minimum by $\sqrt{c_{i1}^2 + c_{i2}^2 + \cdots + c_{iN}^2}$.
10: Return P_i as object match.

Step 1 allocates space for a 2-dimensional matrix C of size $|L| \times N$. For each known object $P_i \in L$ there is a corresponding element $c_{ij} \in C$ to hold the minimum cost solution to subgraph isomorphism between graphs G_j and H_{ij}.

Steps 2 - 7 assigns values to each element $c_{ij} \in C$, see algorithm 2 for detail.

Step 8 normalizes each of the N columns in C to the range $[0, 1]$, see algorithm 6 for detail.

Step 9 finds the row i_{Min} whose location in the N-dimensional hypercube lies closest to origin, see algorithm 7 for detail.

Step 10 returns the object $P_{i_{Min}}$ as the solution.

Algorithm 6 Object matcher, step 8

for $j = 1$ to N **do**
  $c_{Min} \leftarrow \infty$
  $c_{Max} \leftarrow -\infty$
  for $i = 1$ to $|L|$ **do**
    if $c_{ij} < c_{Min}$ **then**
      $c_{Min} \leftarrow c_{ij}$
    end if
    if $c_{ij} > c_{Max}$ **then**
      $c_{Max} \leftarrow c_{ij}$
    end if
  end for
  for $i = 1$ to $|L|$ **do**
    $c_{ij} \leftarrow (c_{ij} - c_{Min})/(c_{Max} - c_{Min})$
  end for
end for

Algorithm 7 Object matcher, step 9

$c_{Min} \leftarrow \infty$
$i_{Min} \leftarrow -1$
for $i = 1$ to $|L|$ **do**
  $c' \leftarrow 0$
  for $j = 1$ to N **do**
    $c' \leftarrow c' + c_{ij}^2$
  end for
  $c' \leftarrow \sqrt{c'}$
  if $c' < c_{Min}$ **then**
    $c_{Min} \leftarrow c'$
    $i_{Min} \leftarrow i$
  end if
end for

Chapter 9

Object matcher illustration

In this chapter we illustrate how the object matcher recognizes an unknown object O among a set L of prototype objects P_i. We introduce an implementation of four domain experts, one for each of our known domains (see chapter 7), complete with metrics and the necessary controlling parameters.

The set L of known objects is created by applying the object extractor (see 7) to a series of 19 real images of a toy truck that is being rotated 180 degrees around its vertical axis. The extracted objects are shown in figures 9.1 through 9.19.

For each of the four domains there is a separate expert. Although these cover different pattern primitives, they share a similar structure and they have a set of common control parameters. Notice that the values chosen for each parameter is sufficiently slacked to allow for a somewhat more interesting match; too restrictive values accept only perfect matching graphs in each domain.

First, the edge-metric is equal for all domains, and is a replica of the metric used in chapter 3).

$$Ee(e_i, e_j) = |\vec{e_i} - \vec{e_j}| - \frac{1}{10} \tag{9.1}$$

Second, all experts enforce a target depth of 3 for step 12 of algorithm 3. As we do in chapter 3, we refer to chapter 1.6 for the reasoning of this choice.

Third, the number of iterations through the recursion loop in steps 15 - 27 of algorithm 3 is limited to 3 by a low-level cut-off. Also, the total number of recursions performed in a single match is limited to 9 by a high-level cut-off.

Fourth, a dynamic threshold is used to either accept or reject associations in step 17 of algorithm 3. The value is given by the function $\epsilon = a|A| + b$, but the values

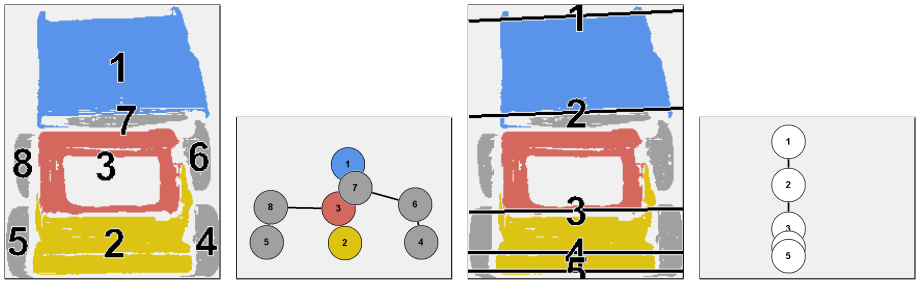


Figure 9.1: Object P_1.

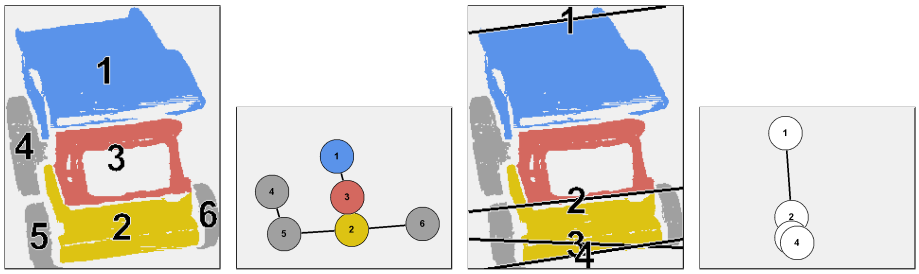


Figure 9.2: Object P_2.

of a and b are separate for each domain.

And finally, an additional threshold ε is employed in step 36 of algorithm 3 to filter isomorphisms whose average cost per vertex- association in A, $\bar{c} = c/|A|$, exceeds it. The value of ε is separate for each domain.

The implementation and values chosen for each domain is done based on both experience and knowledge of the algorithms involved.

Domain 1: Region mass

The vertex-metric for mass-domain allows the cost of an association to exist anywhere in the range $[-\frac{1}{2}, \infty]$. As long as two vertices are reasonably matched, the metric will assign a reasonable cost to the association, but as the two vertices begin to differ, the cost increases rapidly.

$$Ev(v_i, v_j) = \frac{\mathbf{max}(A_i, A_j)}{\mathbf{min}(A_i, A_j)} - \frac{3}{2} \tag{9.2}$$

The association threshold is given by the parameters $a = 0$ and $b = -\frac{1}{4}$. This

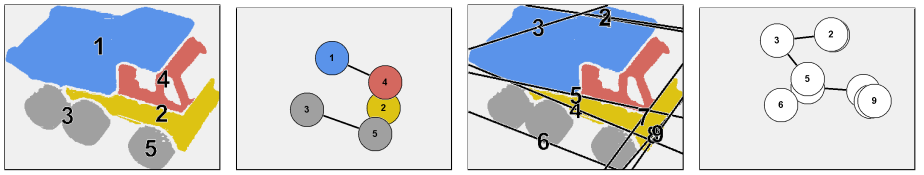


Figure 9.3: Object P_7.

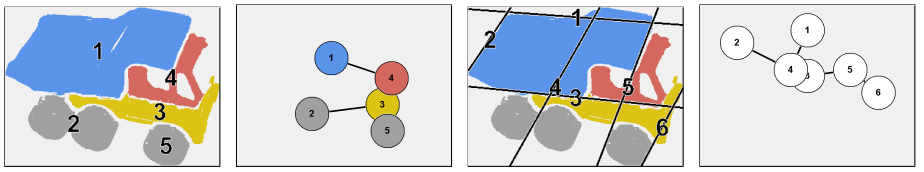


Figure 9.4: Object P_8.

allows the larger of the two regions to be no more than 125% the size of the smaller. As for all the other domains, there is no increased restriction as the recursion depth increases.

The isomorphism threshold for this domain is $\varepsilon = 0$, a very accepting value.

Domain 2: Region colour

The vertex-metric for the colour-domain is covered in chapter 3, and reused here:

$$Ev(v_i, v_j) = \frac{|R_i - R_j| + |G_i - G_j| + |B_i - B_j|}{3 \cdot 255} - \frac{1}{2} \tag{9.3}$$

The association threshold is given by the parameters $a = 0$ and $b = -\frac{9}{20}$; this requires an initial near-match of the region colours, but enforces no further improvement in cost based on the regions' spatial centroids. If there is sufficient mismatch of spatial relations by an association, it will still be filtered by the initial threshold.

Again, the isomorphism threshold for this domain is $\varepsilon = 0$.

Domain 3: Region moments

The vertex-metric for the moment-domain finds the Euclidean distance between the two sets of moments, shifted by the constant $\frac{1}{4}$ to bias association of closely matched vertices.

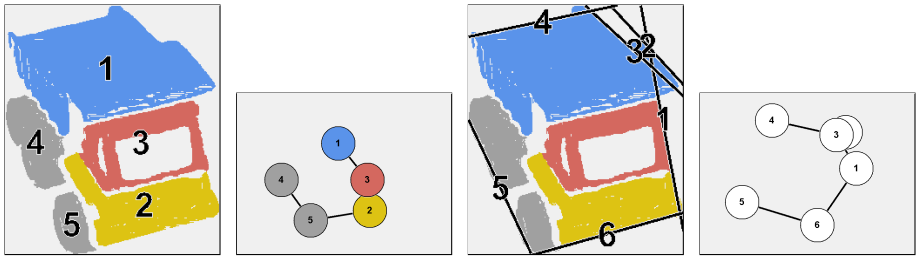


Figure 9.5: Object P_3.

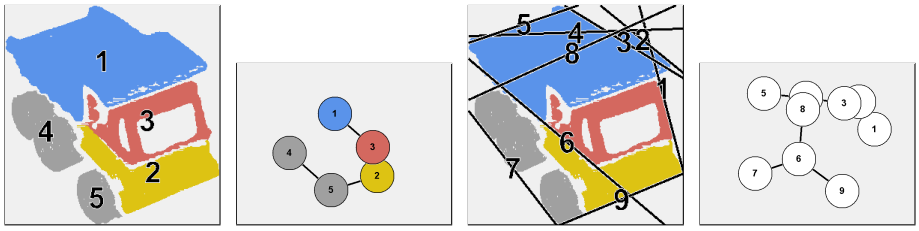


Figure 9.6: Object P_4.

$$Ev(v_i, v_j) = \sqrt{\sum_{m=1}^{4} (I_{im} - I_{jm})^2} - \frac{1}{4} \tag{9.4}$$

The association threshold is given by the parameters $a = 0$ and $b = -\frac{1}{5}$; which does not give the matcher too much slack to accept vertex-associations, but there is no increased restrictions at deeper levels of recursion.

Again, the isomorphism threshold for this domain is $\varepsilon = 0$.

Domain 4: Dominant lines

The vertex-metric for the line-domain is:

$$Ev(v_i, v_j) = \mathbf{min}((|\vec{A}_i - \vec{A}_j| + |\vec{B}_i - \vec{B}_j|), (|\vec{A}_i - \vec{B}_j| + |\vec{B}_i - \vec{A}_j|)) - \frac{1}{4} \tag{9.5}$$

The association threshold is given by the parameters $a = 0$ and $b = 0$; values that allow for a large amount of slack for the matcher. The constant in Ev, $-\frac{1}{4}$, only assigns negative cost to association where the two end-points $\vec{A}$ and $\vec{B}$ of the vertices agree sufficiently.

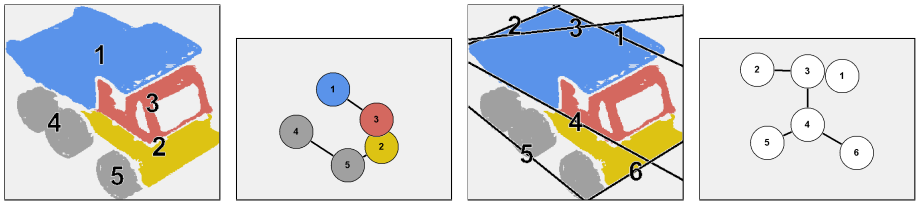


Figure 9.7: Object P_5.

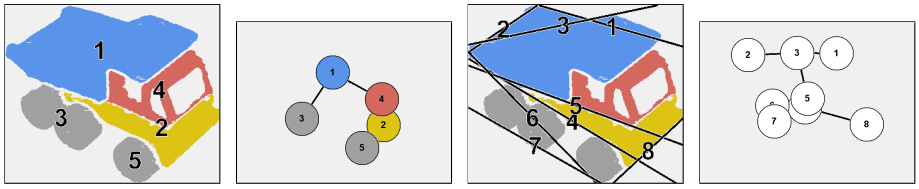


Figure 9.8: Object P_6.

Again, the isomorphism threshold for this domain is $\varepsilon = 0$.

9.1 Illustration 1

For this illustration, the known object P_5 is duplicated as O.

The first part of algorithm 5 creates a cost matrix C that holds the minimum cost of the solution to the MINIMUM COST SUBGRAPH ISOMORPHISM problem that corresponds with each element $c_{ij} \in C$. The row i refers to an object $P_i \in L$, and the column j refers to one of the known domains. If there was no solution found by algorithm 2, the value of c_{ij} is not set (indicated by a hyphen in the following tables).

By the termination of the loop through steps 2 - 7, the elements $c_{ij} \in C$ that correspond to MINIMUM COST SUBGRAPH ISOMORPHISM problems that have solutions, have all been assigned the cost of those solutions. At this point, the matrix C is:

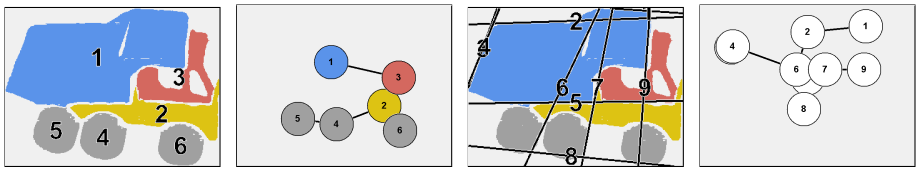


Figure 9.9: Object P_9.

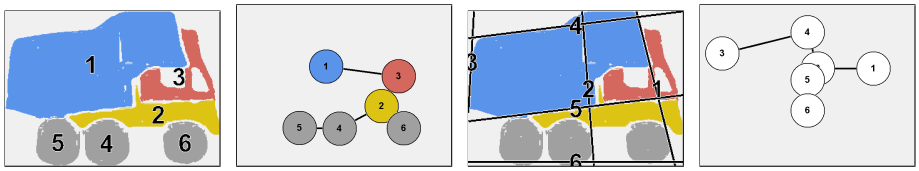


Figure 9.10: Object P_{10}.

Object	Domain			
	Color	Line	Moment	Size
P_1	-0.36	-	-	-
P_2	-	-	-	-
P_3	-2.37	-	-0.96	-1.29
P_4	-3.08	-	-1.76	-2.79
P_5	-3.70	-3.00	-2.45	-3.70
P_6	-3.09	-0.86	-1.71	-2.76
P_7	-2.53	-	-0.90	-1.83
P_8	-1.94	-	-	-
P_9	-1.98	-	-	-
P_{10}	-	-	-	-
P_{11}	-	-	-	-
P_{12}	-1.17	-	-	-
P_{13}	-	-	-	-
P_{14}	-	-	-	-
P_{15}	-	-	-	-
P_{16}	-	-	-	-
P_{17}	-	-	-	-
P_{18}	-	-	-	-
P_{19}	-	-	-	-

Notice from this table how the distribution of solutions are different for each domain. Even by the very unconstrained parameters used for this illustration, there are not many solutions found beyond the closest neighbours of the actual object. The line-domain is very effectively narrowing the acceptable objects to match in L.

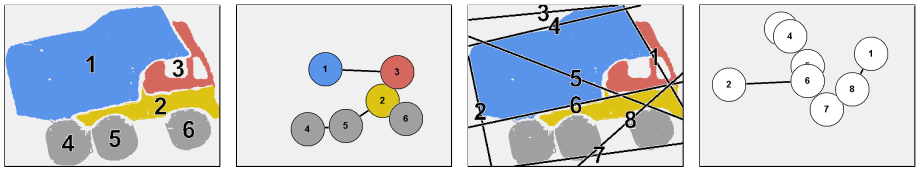


Figure 9.11: Object P_{11}.

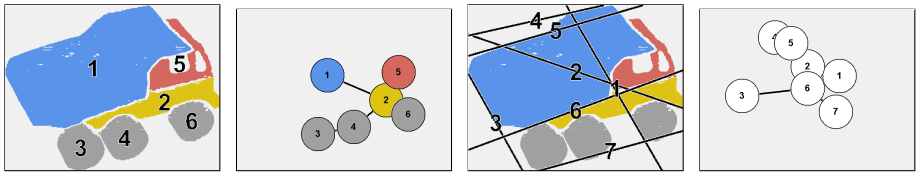


Figure 9.12: Object P_{12}.

The algorithm 6, run as step 8 of algorithm 5, normalizes each column of C individually to the range $[0, 1]$. This process ignores the unset elements of each column. As control returns to algorithm 5, the matrix C is:

		Domain			
		Color	Line	Moment	Size
Object	P_{01}	1.00	-	-	-
	P_{02}	-	-	-	-
	P_{03}	0.40	-	0.96	1.00
	P_{04}	0.19	-	0.45	0.38
	P_{05}	0.00	0.00	0.00	0.00
	P_{06}	0.18	1.00	0.48	0.39
	P_{07}	0.35	-	1.00	0.78
	P_{08}	0.53	-	-	-
	P_{09}	0.51	-	-	-
	P_{10}	-	-	-	-
	P_{11}	-	-	-	-
	P_{12}	0.76	-	-	-
	P_{13}	-	-	-	-
	P_{14}	-	-	-	-
	P_{15}	-	-	-	-
	P_{16}	-	-	-	-
	P_{17}	-	-	-	-
	P_{18}	-	-	-	-
	P_{19}	-	-	-	-

The algorithm 7, run as step 0 of algorithm 5, runs through each row i of C,

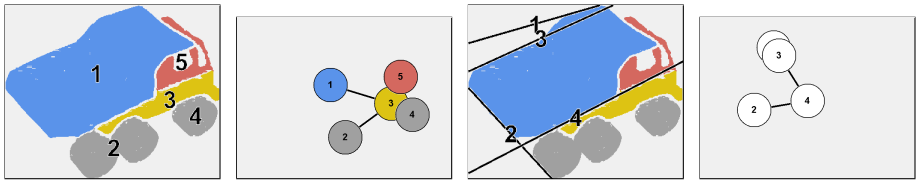


Figure 9.13: Object P_{13}.

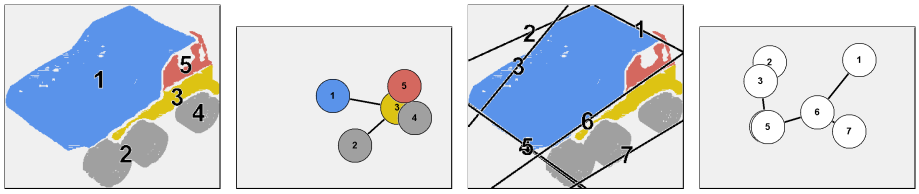


Figure 9.14: Object P_{14}.

and calculates the distance of the vector $[c_{i1}, c_{i2}, c_{i3}, c_{i4}]$ to the origin of the 4-dimensional hypercube, $[0, 0, 0, 0]$, by simple Euclidean distance. As control returns to algorithm 5, the distance to origin for each row in matrix C is:

		Domain				Distance
		Color	Line	Moment	Size	to origin
Object	P_{01}	1.00	-	-	-	-
	P_{02}	-	-	-	-	-
	P_{03}	0.40	-	0.96	1.00	-
	P_{04}	0.19	-	0.45	0.38	-
	P_{05}	0.00	0.00	0.00	0.00	0.00
	P_{06}	0.18	1.00	0.48	0.39	2.05
	P_{07}	0.35	-	1.00	0.78	-
	P_{08}	0.53	-	-	-	-
	P_{09}	0.51	-	-	-	-
	P_{10}	-	-	-	-	-
	P_{11}	-	-	-	-	-
	P_{12}	0.76	-	-	-	-
	P_{13}	-	-	-	-	-
	P_{14}	-	-	-	-	-
	P_{15}	-	-	-	-	-
	P_{16}	-	-	-	-	-
	P_{17}	-	-	-	-	-
	P_{18}	-	-	-	-	-
	P_{19}	-	-	-	-	-

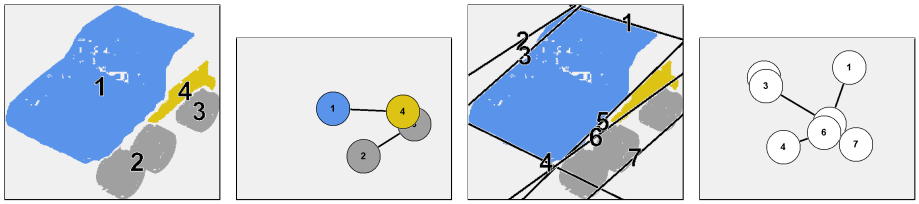


Figure 9.15: Object P_{15}.

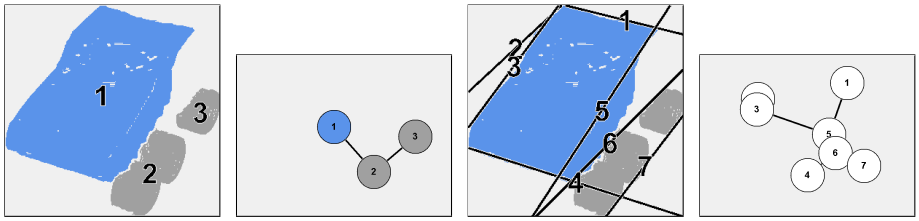


Figure 9.16: Object P_{16}.

The only two objects that the algorithm accepts from L as possible matches to O are P_5, the object itself, and P_6, see figure 9.8. Please take a moment and observe the likeness of the two objects in figures 9.9 and 9.10. By the measurement of distance to origin, the matcher returns P_5 as the match to O.

Since O is P_5, this is a perfect match.

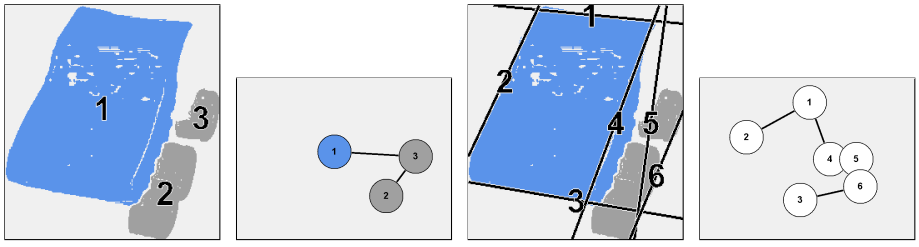


Figure 9.17: Object P_{17}.

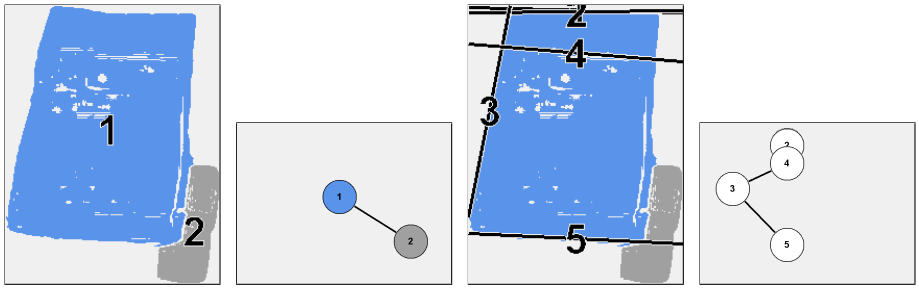


Figure 9.18: Object P_{18}.

9.2 Illustration 2

For this illustration, the known object P_{10} is duplicated as O. At the end of algorithm 5, the matrix C, including distances to origin, is:

		Domain				Distance
		Color	Line	Moment	Size	to origin
Object	P_1	-	-	-	-	-
	P_2	-	-	-	-	-
	P_3	-	-	-	-	-
	P_4	-	-	-	-	-
	P_5	-	-	-	-	-
	P_6	-	-	-	-	-
	P_7	0.69	-	-	-	-
	P_8	0.54	-	-	1.00	-
	P_9	0.19	1.00	0.97	0.34	2.51
	P_{10}	0.00	0.00	0.00	0.00	0.00
	P_{11}	0.19	-	1.00	0.49	-
	P_{12}	0.41	-	-	0.97	-
	P_{13}	0.56	-	-	-	-
	P_{14}	-	-	-	-	-
	P_{15}	-	-	-	-	-
	P_{16}	-	-	-	-	-
	P_{17}	-	-	-	-	-
	P_{18}	1.00	-	-	-	-
	P_{19}	-	-	-	-	-

Since O is P_{10}, this is a perfect match.

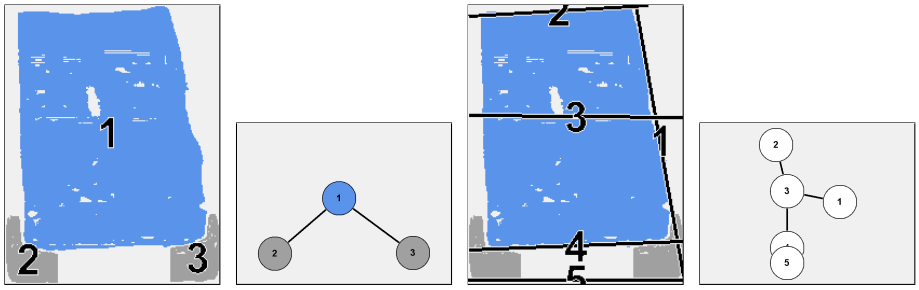


Figure 9.19: Object P_{19}.

9.3 Illustration 3

For this illustration, the known object P_{15} is duplicated as O. At the end of algorithm 5, the matrix C, including distances to origin, is:

		Domain				Distance
		Color	Line	Moment	Size	to origin
Object	P_1	-	-	-	-	-
	P_2	-	-	-	-	-
	P_3	-	-	-	-	-
	P_4	-	-	-	-	-
	P_5	-	-	-	-	-
	P_6	-	-	-	-	-
	P_7	-	-	-	-	-
	P_8	-	-	-	-	-
	P_9	-	-	-	-	-
	P_{10}	-	-	-	-	-
	P_{11}	-	-	-	-	-
	P_{12}	-	-	-	-	-
	P_{13}	0.59	-	1.00	1.00	-
	P_{14}	0.33	-	0.67	0.67	-
	P_{15}	0.00	0.00	0.00	0.00	0.00
	P_{16}	0.75	1.00	0.69	0.78	3.22
	P_{17}	1.00	-	-	-	-
	P_{18}	-	-	-	-	-
	P_{19}	-	-	-	-	-

Since O is P_{15}, this is a perfect match.

Chapter 10

Object matcher results

10.1 Overview

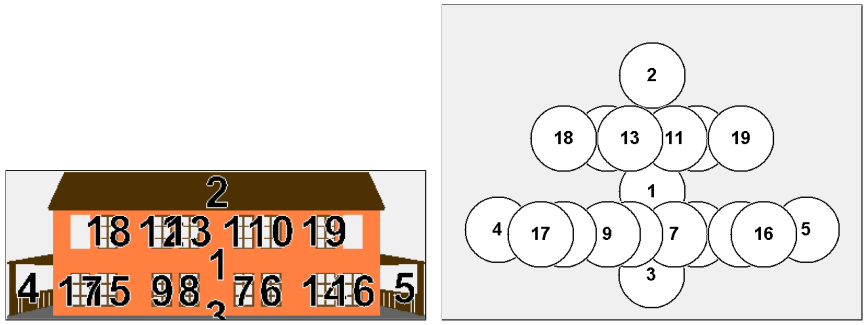
In this chapter we demonstrate how our matcher is able to recognize objects under heavy noise. Because our work is more concerned with the inexact graph matching than the object extraction, we allow ourselves to run the system on computer-generated images of 3d models. This simplifies the extraction process (see chapter 7), since there is no noise other than what we deliberately introduce.

For these experiments we use a set of 100 house models. The first 75 of these are used to build a set of prototype objects by rendering the models without noise and applying the object extractor to these images. For each experiment all 100 models are rendered, under noise determined by the experiment, and the candidate objects are extracted from those images.

In chapter 4 we improve the results of the graph matcher by modifying the controlling parameters by the knowledge of what noise is applied. In this chapter, however, we use a single set of parameters for all experiments. Instead of trying to achieve the best possible statistics for each individual experiment, these results are intended to show how efficient and robust our approach is under various noise.

The metrics Ee and Ev for each of the four domains are given in chapter 9. For all domains target depth is 3, low-level cut-off is 10, high-level cut-off is 1000, and the average cost threshold is $\varepsilon = 0$. The association threshold for region colour is $\epsilon = -\frac{1}{2}$, for region mass $\epsilon = -\frac{1}{4}$, for region moment invariants $\epsilon = -\frac{1}{10}$, and for dominant lines $\epsilon = -\frac{1}{10}$.

By the results in part 1, we know that our inexact graph matcher works better on larger graphs with no repeated patterns.



(a) Detected regions. (b) Extracted region graph.

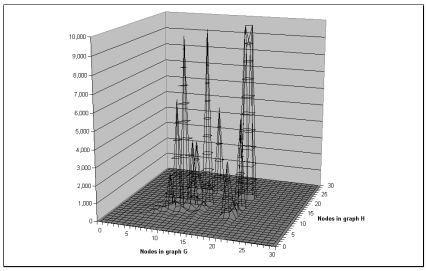
Figure 10.1: Example of an ARG extracted from a rendered house.

The cost matrix C (see algorithm 3), and the corresponding contribution matrix C', of larger graphs are expected to be less ambiguous because each vertex-associations implies many more edge-associations. Larger graphs hold a lot more syntactic information that ultimately converges C'.

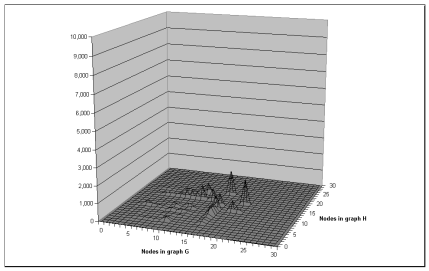
Repeated pattern primitives disrupt this convergence – although the matcher is likely in its first recursions to attempt the most unambiguous vertex-associations, it is left to choose haphazardly between the remaining associations. The refinements to C are intended to solve ambiguity by the graph's spatial relationships, but when the spatial layout of similar pattern primitives is also similar, even more ambiguity arises.

Our 3d models are quite simple, giving smaller graphs, and for most houses its windows are likely produce repeating patterns with very similar spatial layout (see figure 10.1).

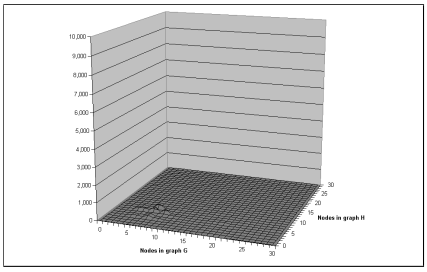
These issues make our 3d models quite a challenge to the matcher, and we expect to see some mismatches even under very little noise. In fact, when sufficient noise is introduced some pattern primitives might be distorted just enough to reduce the number of ambiguous associations, and thereby improve the results.



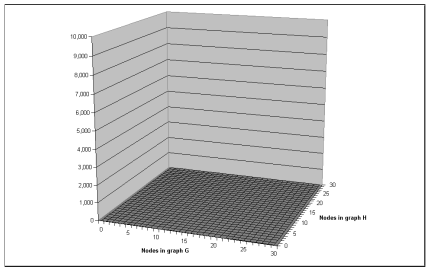
(a) True-Positive. Count 22,500, depth 0.61, recursions 2.29, refinements 107.48.



(b) True-Negative. Count 6,300, depth 0.47, recursions 1.30, refinements 25.73.



(c) False-Positive. Count 1,200, depth 0.57, recursions 1.43, refinements 22.99.



(d) False-Negative. None.

Figure 10.2: Match with 25% centroid-, pattern-, and vertex noise.

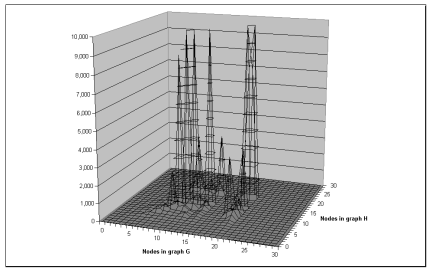
10.2 Improvement by domains

10.2.1 25% centroid-, pattern-, and vertex noise

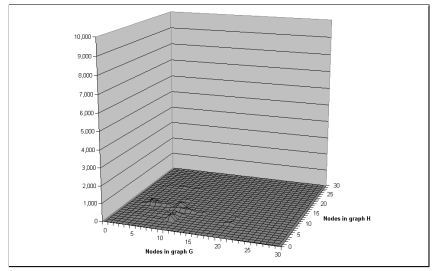
Accuracy	Error	Precision	Recall	F1
0.96	0.04	0.95	1.00	0.97

The noise applied in this example is equal to that in the last section of chapter 4, where the matcher was able to achieve an F1 value of 0.70. Through the use of several domains, both Precision and Recall has been significantly raised under the same noise. Under the very general controlling parameters given above, we achieve close to perfect results.

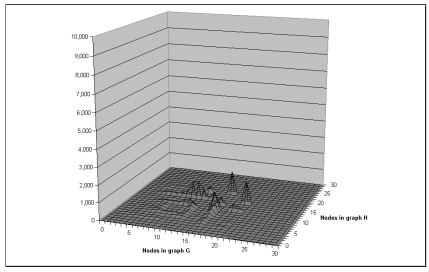
These runtimes are different from the corresponding graph matches in chapter 4 because these experiments run on the house models (i.e. smaller, repeating graphs), not on the random 1- to 100-vertex graphs.



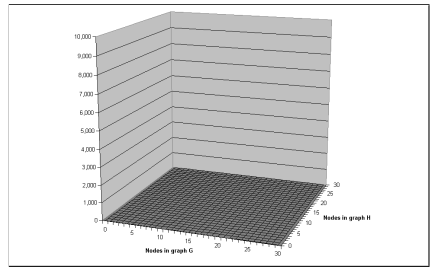
(a) True-Positive. Count 22,500, depth 0.71, recursions 2.82, refinements 143.05.



(b) True-Negative. Count 3,000, depth 0.53, recursions 1.53, refinements 29.66.



(c) False-Positive. Count 4,500, depth 0.70, recursions 1.98, refinements 49.41.



(d) False-Negative. None.

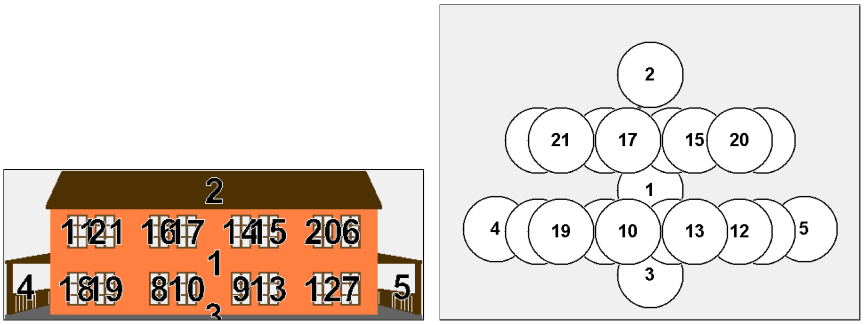
Figure 10.3: Match with no noise.

10.2.2 No noise

Accuracy	Error	Precision	Recall	F1
0.85	0.15	0.83	1.00	0.91

For reference we include the results from matching the candidate objects completely undistorted by noise. Although the results are good, one would expect perfect results when there is no noise. We still accept the controlling parameters as they are because they perform well under all the noise we apply in this chapter.

The number of average recursions by brute-force is 1.35E+30, whereas our parameters reduce this to a single digit number. Likewise, the number of average refinements is down from 3.75E+31 to a 2-3 digit number.



(a) Detected regions. (b) Extracted region graph.

Figure 10.4: Example of an ARG extracted under scale noise.

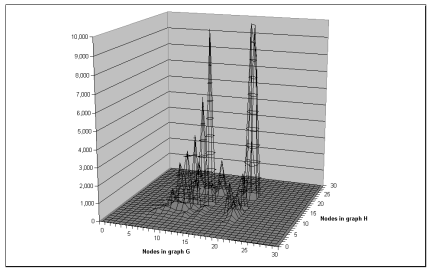
10.3 Performance under noise

10.3.1 Scale noise

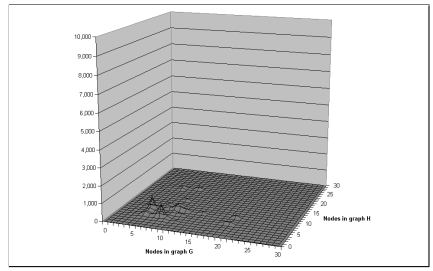
By "scale noise" we mean changes to the size of the rendered images of the 3d models. As the scale changes of these images, regions or lines may appear or disappear. An example of additional regions is illustrated in figure 10.4, where the left- and right-most windows on the upper floor of the house are now considered regions instead of noise (see figure 10.1). Similarly, by reducing the size of the rendered image most windows will be thresholded as insignificant.

The percentage p is a measurement of the applied scaling. The aspect-ratio of the images are retained, so the noise is determined by a single value chosen at random from the range $[1 - \frac{1}{2}p, 1 + \frac{1}{2}p]$. Both the width and the height of the image is scaled by this value before rendering.

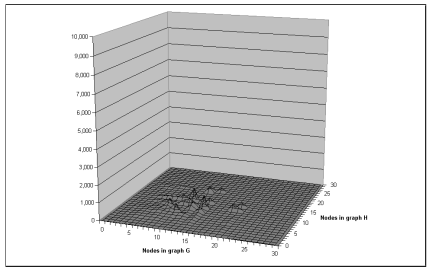
This noise is similar to the vertex-noise in chapter 4, and should, therefore, not be a problem for the matcher. It is an important type of noise to consider because it is likely to occur in systems that process real images.



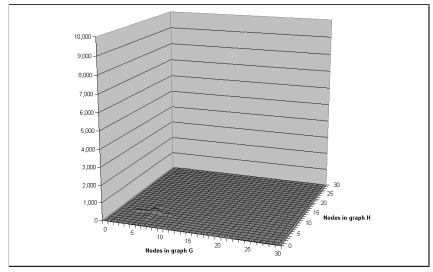
(a) True-Positive. Count 21,900, depth 0.71, recursions 2.53, refinements 107.33.



(b) True-Negative. Count 3,000, depth 0.56, recursions 1.77, refinements 37.59.



(c) False-Positive. Count 4,800, depth 0.72, recursions 2.16, refinements 51.03.



(d) False-Negative. Count 300, depth 0.57, recursions 1.40, refinements 21.00.

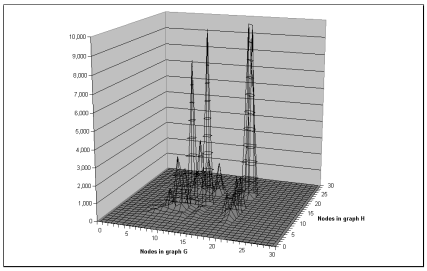
Figure 10.5: Match with 10% scale noise.

10% scale noise

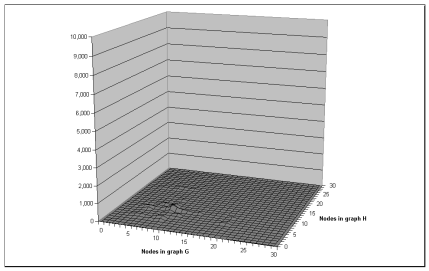
Accuracy	Error	Precision	Recall	F1
0.83	0.17	0.82	0.99	0.90

All controlling parameters are given at the start of this chapter.

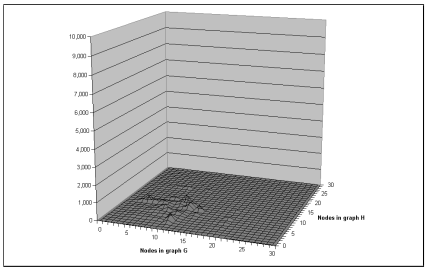
The results are close to that of no noise. By figure 10.5 it seems that the mismatches occur only on the smaller graphs. As opposed to the results under no noise, where Recall was perfect, some instances are now also incorrectly unmatched. Both Recall and Precision is down by 0.01, which forces F1 down by 0.01.



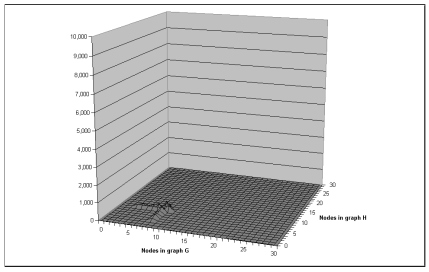
(a) True-Positive. Count 22,200, depth 0.70, recursions 2.47, refinements 97.88.



(b) True-Negative. Count 3,600, depth 0.57, recursions 1.67, refinements 32.07.



(c) False-Positive. Count 3,900, depth 0.72, recursions 2.24, refinements 52.87.



(d) False-Negative. Count 300, depth 0.63, recursions 2.03, refinements 46.03.

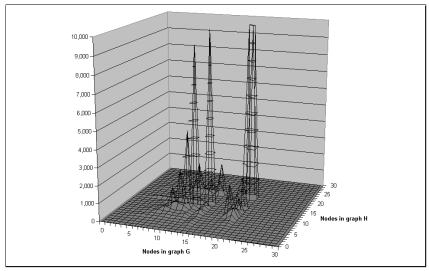
Figure 10.6: Match with 25% scale noise.

25% scale noise

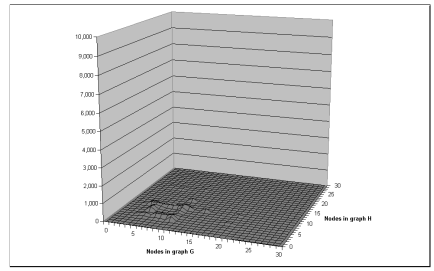
Accuracy	Error	Precision	Recall	F1
0.86	0.14	0.85	0.99	0.91

All controlling parameters are given at the start of this chapter.

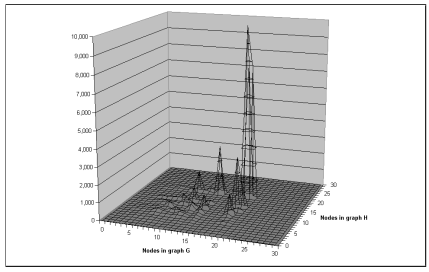
At more than twice the amount of noise of the previous experiment, results are better. Although Recall is still not perfect, Precision has increased beyond the result of the no noise experiment, and F1 equals it. Most of the runtimes equal those of the first experiment, but for the correct matches less refinements are applied.



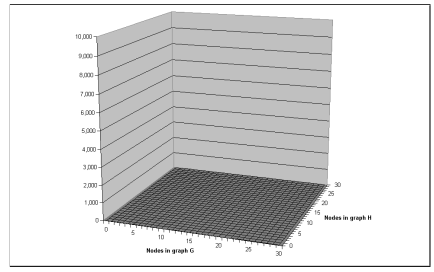
(a) True-Positive. Count 22,200, depth 0.71, recursions 2.46, refinements 90.13.



(b) True-Negative. Count 3,300, depth 0.57, recursions 1.80, refinements 36.81.



(c) False-Positive. Count 4,500, depth 0.70, recursions 2.24, refinements 98.35.



(d) False-Negative. None.

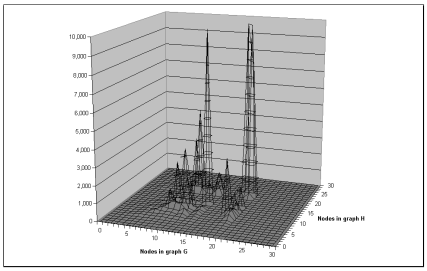
Figure 10.7: Match with 50% scale noise.

50% scale noise

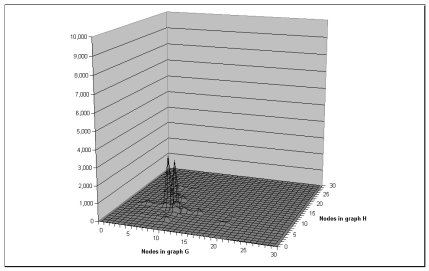
Accuracy	Error	Precision	Recall	F1
0.85	0.15	0.83	1.00	0.91

All controlling parameters are given at the start of this chapter.

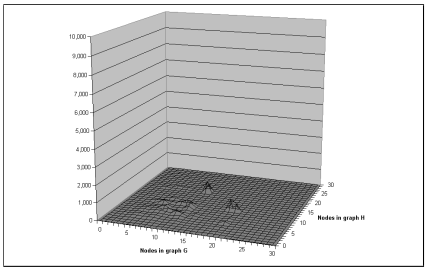
The results are equal to the no noise experiment. The number of refinements applied in the correct matches is down to almost half of the original, for the correct unassigned matches it is slightly higher, but for the incorrect matches it has actually doubled. By figure 10.7 there seems to be some errors even when the graphs are larger.



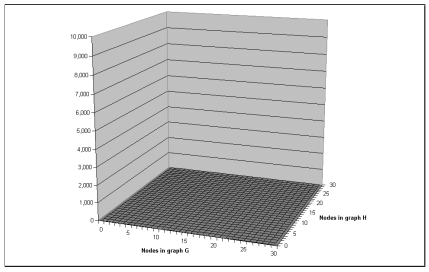
(a) True-Positive. Count 22,200, depth 0.71, recursions 2.48, refinements 98.47.



(b) True-Negative. Count 4,200, depth 0.57, recursions 1.79, refinements 35.83.



(c) False-Positive. Count 3,300, depth 0.71, recursions 2.12, refinements 48.15.



(d) False-Negative. Count 300, depth 0.46, recursions 0.81, refinements 6.17.

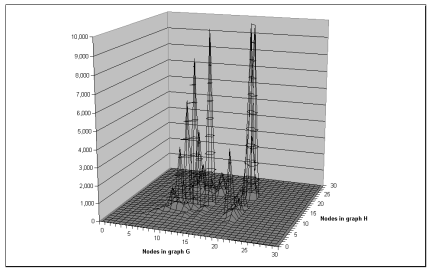
Figure 10.8: Match with 75% scale noise.

75% scale noise

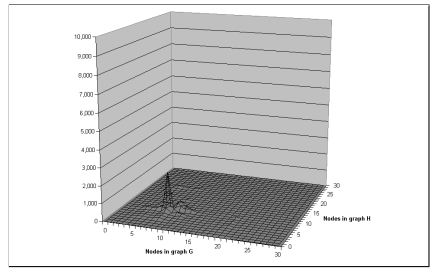
Accuracy	Error	Precision	Recall	F1
0.88	0.12	0.87	0.99	0.93

All controlling parameters are given at the start of this chapter.

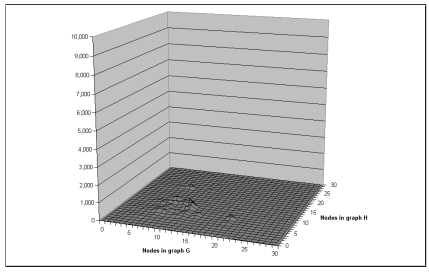
Recall is again down from the perfect results achieved under no noise, but Precision has increased enough to push F1 beyond the no noise results. The only runtimes that differ from the other are the correct matches, which are down to almost half of the original ones. The only explanation of these results is some combination of 1) ambiguity between prototypes being resolved by the noise, and 2) the randomness of the noise favouring beneficial values in those instances where it is critical.



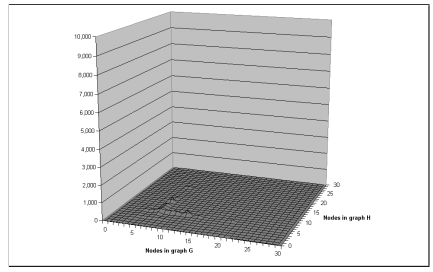
(a) True-Positive. Count 21,900, depth 0.70, recursions 2.44, refinements 103.83.



(b) True-Negative. Count 3,600, depth 0.56, recursions 1.75, refinements 36.43.



(c) False-Positive. Count 4,200, depth 0.70, recursions 1.94, refinements 43.35.



(d) False-Negative. Count 300, depth 0.78, recursions 2.49, refinements 75.65.

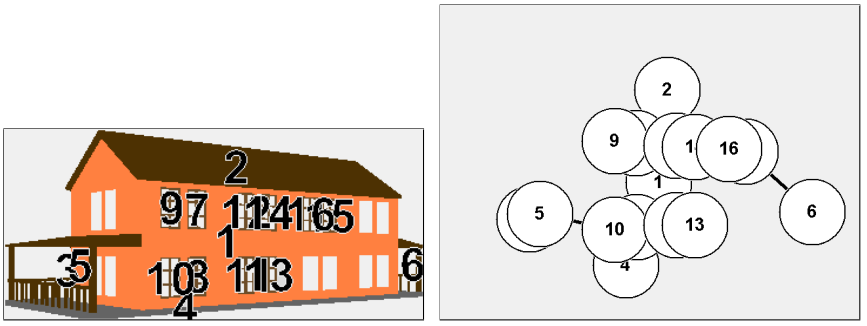
Figure 10.9: Match with 95% scale noise.

95% scale noise

Accuracy	Error	Precision	Recall	F1
0.85	0.15	0.84	0.99	0.91

All controlling parameters are given at the start of this chapter.

Although runtimes are lower than that of the no noise experiment, the results are equal. The trend through scale noise has been a reduction in runtimes without compromising the results. This can partly be explained by the fact that scale noise would easier remove than add regions, since the original scale is able to capture most regions – scaling the image up from figure 10.1 to figure 10.4 adds 2 regions, while scaling down is likely to remove all 14 window- regions. By this rationale the size of the average graph, under this type of noise, goes down, and therefore also the average runtime.



(a) Detected regions. (b) Extracted region graph.

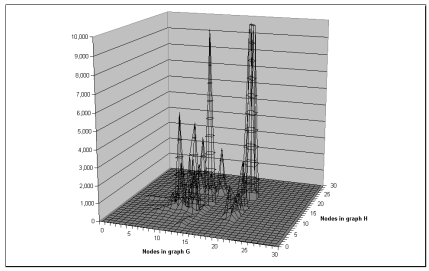
Figure 10.10: Example of an ARG extracted under pose noise.

10.3.2 Pose noise

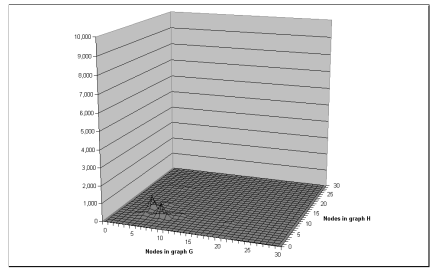
By "pose noise" we mean a change to the viewpoint used when rendering the 3d models. Because we only allow the viewpoint to move in a perfect circle around the model, we are, in fact, only altering the pose of it. As with scale noise, one should expect to see this noise in systems that process real images – attempting to always capture images of real objects from some specific pose is in itself a difficult task.

An example of this type of noise can be seen in the candidate object in figure 10.10, as opposed to the corresponding prototype in figure 10.1. The implications of this noise is quite severe; many regions have their pattern primitive altered, some regions disappear, and some new regions appear. In addition, the dominant lines may change significantly too.

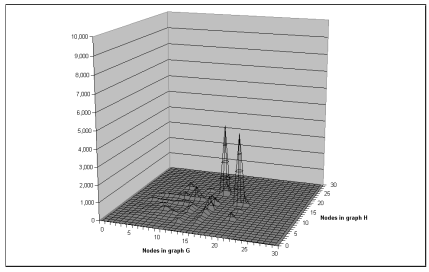
The percentage p is a measurement of how much the viewpoint is allowed to move. For simplicity, rotation is restricted to occur only around the vertical-axis of the model itself. Furthermore, rotation beyond $\frac{1}{2}\pi$ to either side of the original is pointless because that will cause all of the original features of the object to rotate out of view. Rotation is, therefore, determined by a random value in the range $[-\frac{1}{2}\pi p, \frac{1}{2}\pi p]$.



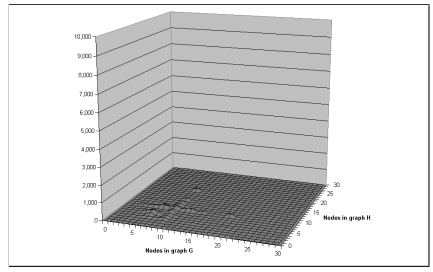
(a) True-Positive. Count 21,000, depth 0.69, recursions 2.37, refinements 98.18.



(b) True-Negative. Count 4,200, depth 0.58, recursions 1.68, refinements 31.09.



(c) False-Positive. Count 3,900, depth 0.69, recursions 2.05, refinements 59.37.



(d) False-Negative. Count 900, depth 0.70, recursions 2.05, refinements 55.74.

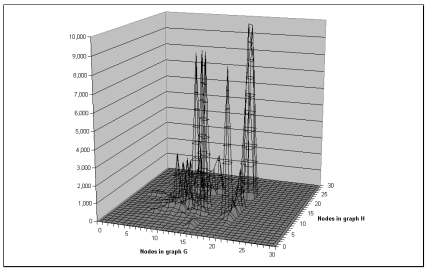
Figure 10.11: Match with 10% pose noise.

10% pose noise

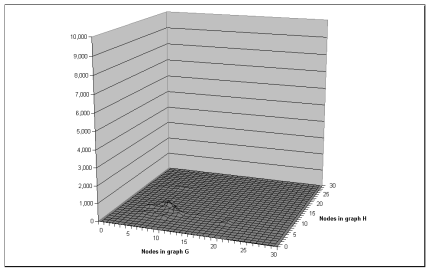
Accuracy	Error	Precision	Recall	F1
0.84	0.16	0.84	0.96	0.90

All controlling parameters are given at the start of this chapter.

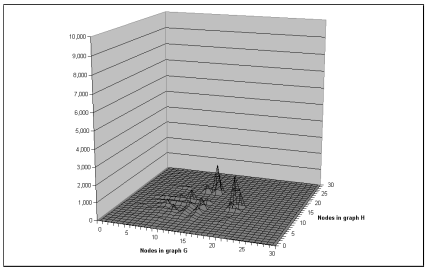
This level of pose noise should be expected in a system that processes real images. Results are close to the instances without noise. Runtimes are also similar, but since the perfect matches are now less perfect, the average amount of refinement applied to those cases are down by one third. Recall is down from the perfect 1.0, but Precision is up.



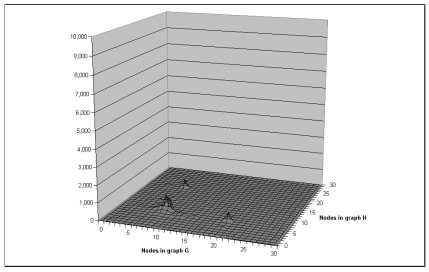
(a) True-Positive. Count 19,500, depth 0.69, recursions 2.34, refinements 110.99.



(b) True-Negative. Count 4,200, depth 0.57, recursions 1.47, refinements 29.95.



(c) False-Positive. Count 4,500, depth 0.69, recursions 1.86, refinements 55.88.



(d) False-Negative. Count 1,800, depth 0.55, recursions 1.53, refinements 39.25.

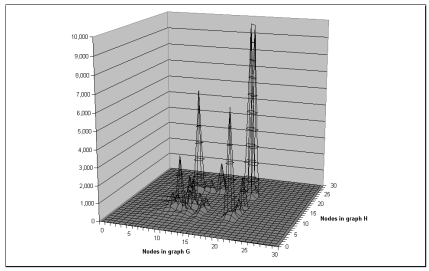
Figure 10.12: Match with 25% pose noise.

25% pose noise

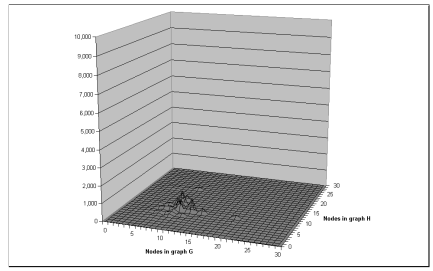
Accuracy	Error	Precision	Recall	F1
0.79	0.21	0.81	0.92	0.86

All controlling parameters are given at the start of this chapter.

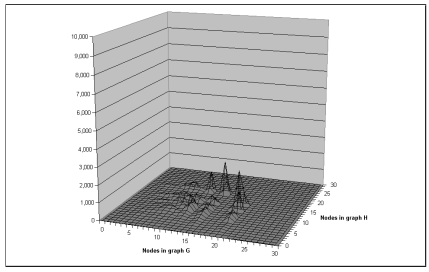
Just as 10% pose noise should be expected in real images, one should not expect more than this level of noise. Both Precision and Recall has dropped as the noise increased, but the F1 score is still a very good 0.86. With more noise comes some higher runtimes, but the number of applied refinements are more or less the same as for 10% noise.



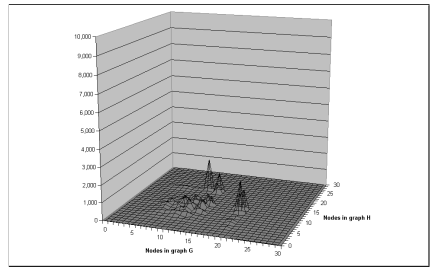
(a) True-Positive. Count 12,900, depth 0.70, recursions 2.21, refinements 91.75.



(b) True-Negative. Count 3,900, depth 0.53, recursions 1.17, refinements 25.38.



(c) False-Positive. Count 7,200, depth 0.67, recursions 1.82, refinements 57.76.



(d) False-Negative. Count 6,000, depth 0.52, recursions 1.40, refinements 43.79.

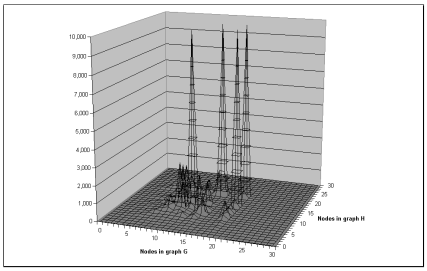
Figure 10.13: Match with 50% pose noise.

50% pose noise

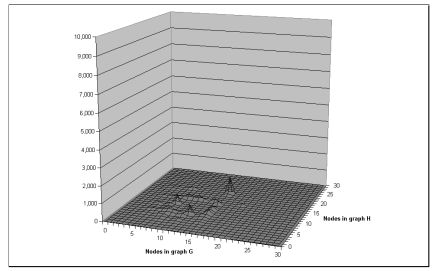
Accuracy	Error	Precision	Recall	F1
0.56	0.44	0.64	0.68	0.66

All controlling parameters are given at the start of this chapter.

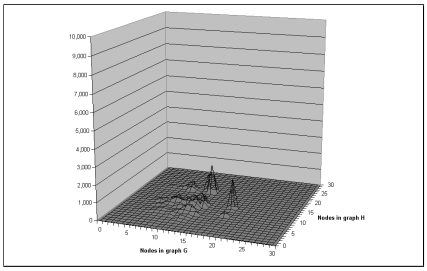
As pose noise increases to 50% and beyond the distortions to the source image, and therefore the candidate object, are so severe that the results are mostly a curiosity. The controlling parameters were chosen to attempt to overcome this, however, and the results are not disappointing. F1 is an impressive 0.66. Runtimes are equal to what we have already seen under this type of noise.



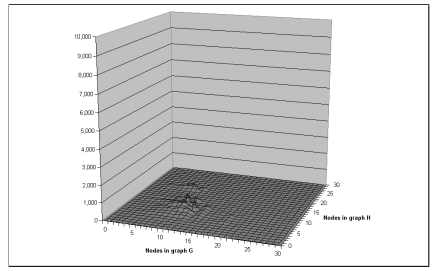
(a) True-Positive. Count 9,000, depth 0.67, recursions 2.14, refinements 94.52.



(b) True-Negative. Count 4,800, depth 0.50, recursions 1.02, refinements 25.08.



(c) False-Positive. Count 6,300, depth 0.64, recursions 1.71, refinements 50.94.



(d) False-Negative. Count 9,900, depth 0.56, recursions 1.47, refinements 45.40.

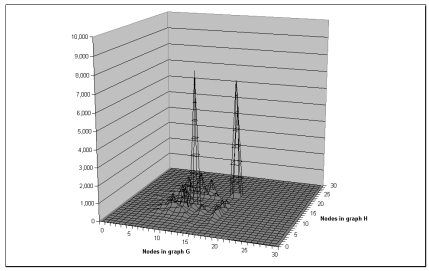
Figure 10.14: Match with 75% pose noise.

75% pose noise

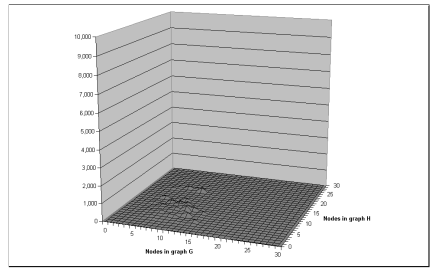
Accuracy	Error	Precision	Recall	F1
0.46	0.54	0.59	0.48	0.53

All controlling parameters are given at the start of this chapter.

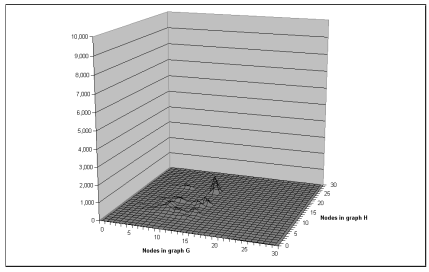
Again, results are very good. The step from 25% to 50% noise reduced the F1 value by 0.20, whereas the step from 50% to 75% only reduced it by 0.13. The system is still able to successfully match half of the distorted candidate objects to the corresponding prototype objects, but we are seeing a substantial amount of incorrectly assigned matches (i.e. False-Positives).



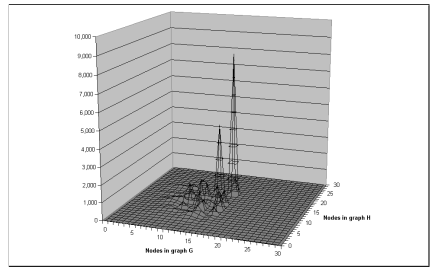
(a) True-Positive. Count 9,000, depth 0.67, recursions 2.13, refinements 75.44.



(b) True-Negative. Count 5,100, depth 0.57, recursions 1.21, refinements 28.01.



(c) False-Positive. Count 4,800, depth 0.70, recursions 1.85, refinements 50.11.



(d) False-Negative. Count 11,100, depth 0.51, recursions 1.26, refinements 42.49.

Figure 10.15: Match with 95% pose noise.

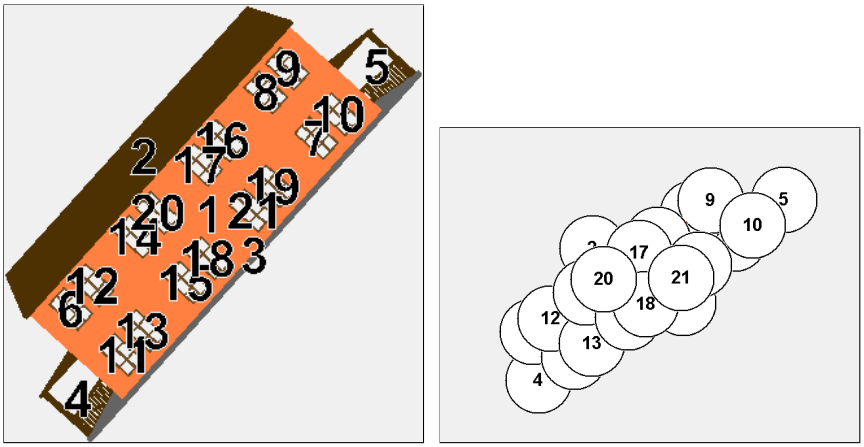
95% pose noise

Accuracy	Error	Precision	Recall	F1
0.47	0.53	0.65	0.45	0.53

All controlling parameters are given at the start of this chapter.

At 95% pose noise we allow rotations in the range $[-\frac{19}{40}\pi, \frac{19}{40}\pi]$, which is extremely close to a full $\frac{1}{2}\pi$ in either direction. As noted under the introduction to this type of noise, at that point the camera is set at either side of the object, removing all pattern primitives visible in the image processed for the prototype object. (This is mostly true, but since a few features on some houses actually extend towards the camera in the prototype, they remain at full rotation as extensions to either side.)

The results are nothing short of amazing, there is no degeneration from 75% noise, F1 remains at 0.53. The runtimes are all similar except for the perfect matches, where the amount of applied refinements are down by one fourth.



(a) Detected regions. (b) Extracted region graph.

Figure 10.16: Example of an ARG extracted under roll noise.

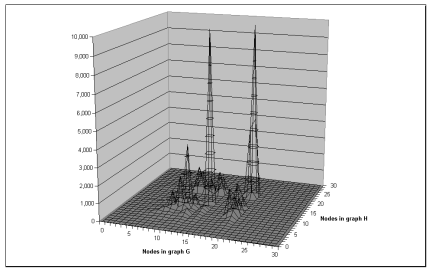
10.3.3 Roll noise

By "roll noise" we mean rotation of the rendered image before it is processed by the object extractor. This is analogous to the camera or the object itself being rotated around the depth-axis of the camera (see figure 10.16). To some extent such noise will always appear in real images.

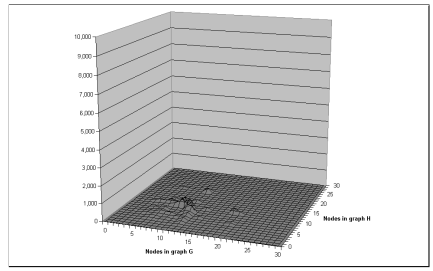
If it is possible to determine the object's primary axes of elongation in the source image, this noise can be minimized by rotating the image back to some reference axes. We consider this minimization a strength of the pre-processor, and not the matcher itself, and therefore choose not to implement it. These experiments are intended to research the resilience of our system to this noise.

The percentage p is a measurement of how much the image is allowed to roll. Rotation beyond π leads to symmetries, and the roll is therefore determined by a random value in the range $[-\frac{1}{2}\pi p, \frac{1}{2}\pi p]$.

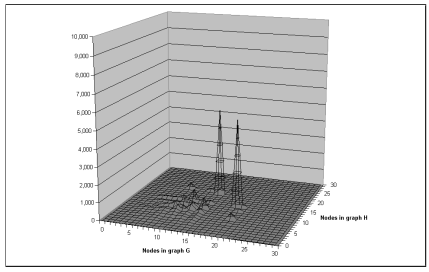
This noise is similar to the spatial-noise in chapter 4. Although the matcher is able to overcome that noise by manipulation of the controlling parameters, that is not an option for these experiments. We expect this noise to be difficult under our very general parameters.



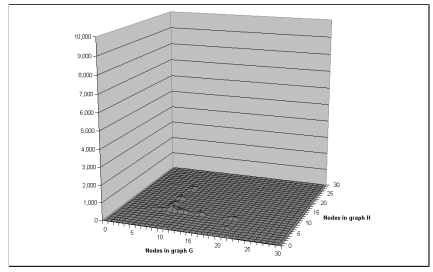
(a) True-Positive. Count 19,800, depth 0.63, recursions 1.96, refinements 74.56.



(b) True-Negative. Count 3,900, depth 0.51, recursions 1.39, refinements 36.26.



(c) False-Positive. Count 5,100, depth 0.62, recursions 1.63, refinements 45.21.



(d) False-Negative. Count 1,200, depth 0.49, recursions 1.16, refinements 30.89.

Figure 10.17: Match with 10% roll noise.

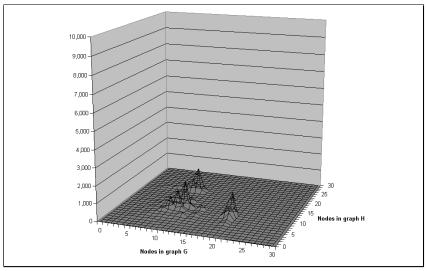
10% roll noise

Accuracy	Error	Precision	Recall	F1
0.79	0.21	0.80	0.94	0.86

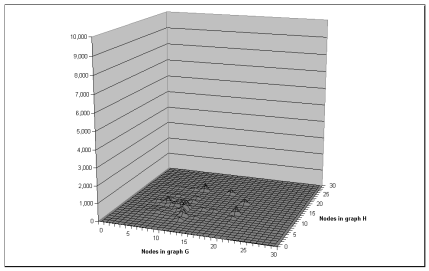
All controlling parameters are given at the start of this chapter.

At 10% noise the results are good, Recall and Precision are both down from the experiments without noise, but the overall F1 is only down by 0.05. Runtimes are also similar, but perfect matches only require half the amount of refinements.

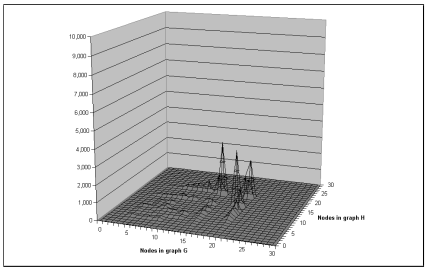
If the pre-processor was allowed to rotate the source image by an objects' primary axes of elongation, we expect all roll noise to appear only in this limited extent.



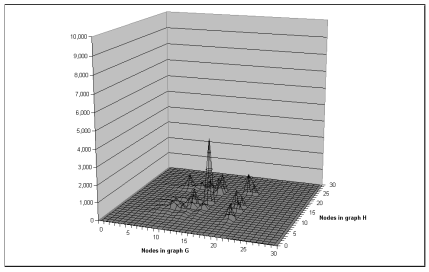
(a) True-Positive. Count 12,000, depth 0.54, recursions 1.73, refinements 60.50.



(b) True-Negative. Count 5,400, depth 0.30, recursions 0.71, refinements 20.85.



(c) False-Positive. Count 3,300, depth 0.63, recursions 1.85, refinements 65.52.



(d) False-Negative. Count 9,300, depth 0.34, recursions 0.98, refinements 35.01.

Figure 10.18: Match with 25% roll noise.

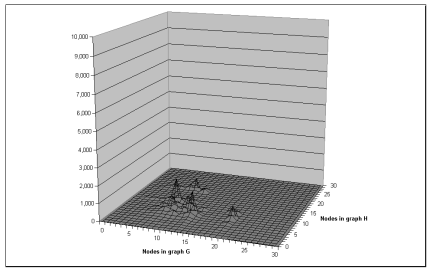
25% roll noise

Accuracy	Error	Precision	Recall	F1
0.58	0.42	0.78	0.56	0.66

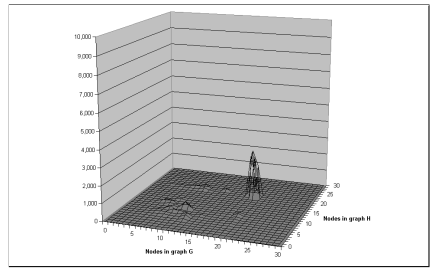
All controlling parameters are given at the start of this chapter.

At this amount of noise the results starts to rapidly degenerate.

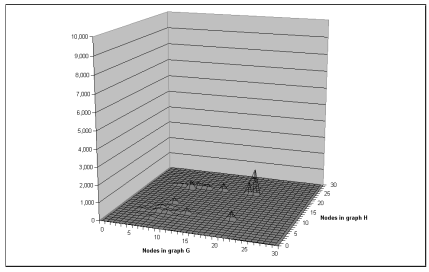
If we were to allow more slack in the spatial relations in the objects, the whole refinement of C would eventually be void, and one would be better off matching purely on the vertex-metric Ev. It was possible in chapter 4 because the noise appeared without any changes to the pattern primitives, whereas this roll modifies region mass, region moment invariants and dominant lines at the same time.



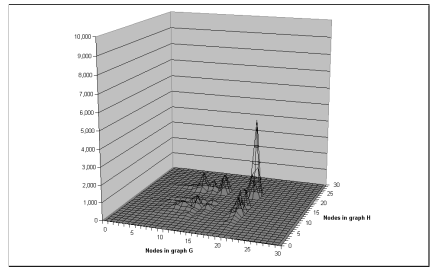
(a) True-Positive. Count 6,300, depth 0.52, recursions 1.62, refinements 57.82.



(b) True-Negative. Count 6,000, depth 0.27, recursions 0.67, refinements 24.01.



(c) False-Positive. Count 2,100, depth 0.49, recursions 1.06, refinements 35.56.



(d) False-Negative. Count 15,600, depth 0.32, recursions 0.96, refinements 38.12.

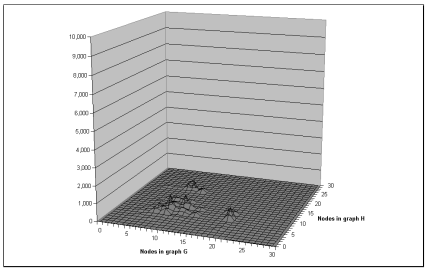
Figure 10.19: Match with 50% roll noise.

50% roll noise

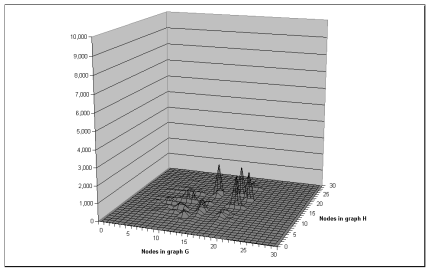
Accuracy	Error	Precision	Recall	F1
0.41	0.59	0.75	0.29	0.42

All controlling parameters are given at the start of this chapter.

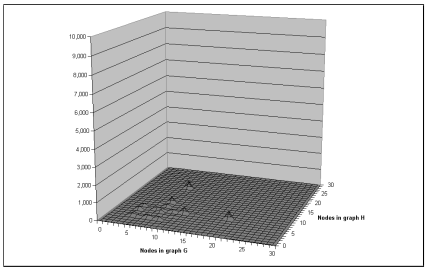
Results degenerate further.



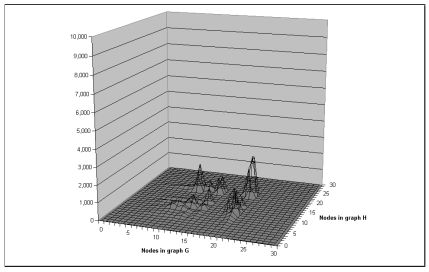
(a) True-Positive. Count 4,200, depth 0.54, recursions 1.73, refinements 60.27.



(b) True-Negative. Count 6,600, depth 0.25, recursions 0.64, refinements 24.11.



(c) False-Positive. Count 1,200, depth 0.45, recursions 1.22, refinements 24.46.



(d) False-Negative. Count 18,000, depth 0.32, recursions 0.98, refinements 38.26.

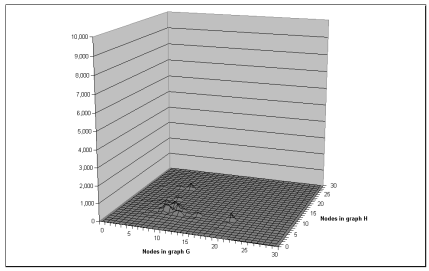
Figure 10.20: Match with 75% roll noise.

75% roll noise

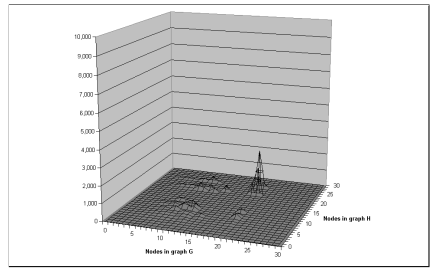
Accuracy	Error	Precision	Recall	F1
0.36	0.64	0.78	0.19	0.30

All controlling parameters are given at the start of this chapter.

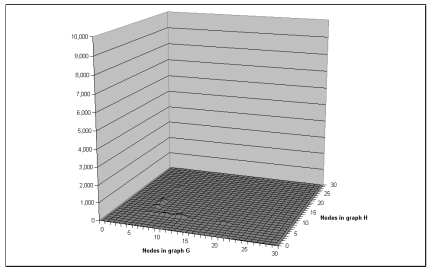
Again, more degeneration of results.



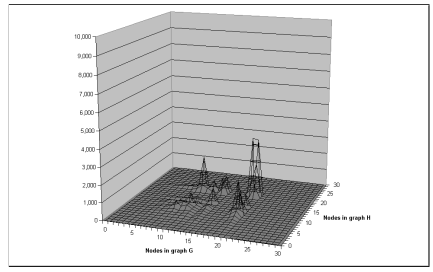
(a) True-Positive. Count 2,700, depth 0.60, recursions 1.88, refinements 61.00.



(b) True-Negative. Count 6,900, depth 0.25, recursions 0.63, refinements 25.23.



(c) False-Positive. Count 1,200, depth 0.48, recursions 1.19, refinements 27.26.



(d) False-Negative. Count 19,200, depth 0.32, recursions 0.96, refinements 39.69.

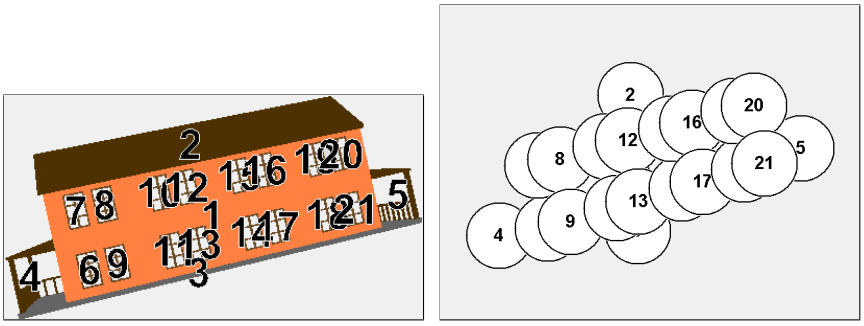
Figure 10.21: Match with 95% roll noise.

95% roll noise

Accuracy	Error	Precision	Recall	F1
0.32	0.68	0.69	0.12	0.21

All controlling parameters are given at the start of this chapter.

And finally, the results are down to the lowest measured F1 at 0.21.



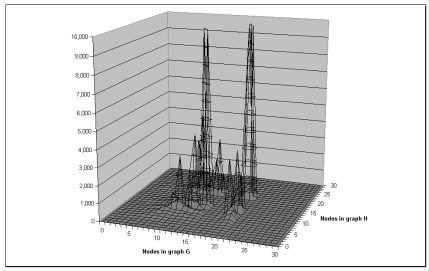
(a) Detected regions. (b) Extracted region graph.

Figure 10.22: Example of an ARG extracted under a combination of noise.

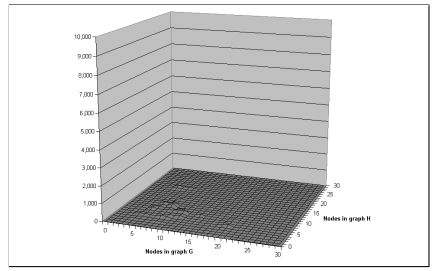
10.3.4 Combinations of noise

In chapter 4 the combination of noise was the only changes to the candidate graph that simultaneously affected both the vertices and the edges, whereas the noise we apply earlier in this chapter always affect both. By combinations of scale, pose and roll noise the distortions to the candidate graphs become quite severe. In this whole chapter we have only used a single set of controlling parameters for the matcher, and we will here examine how well they hold up under the worst possible noise.

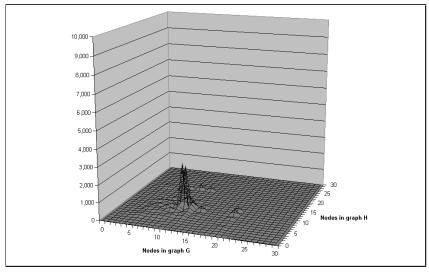
Figure 10.22 is an example of a candidate graph extracted under a combination of noise. The scale noise has made all windows of the house dominant enough to pass the object extractor as separate regions, the pose noise has twisted the house so that the left side is closer to the camera (noticeable by the skew of the roof), and the roll noise has rotated the whole image counter-clockwise.



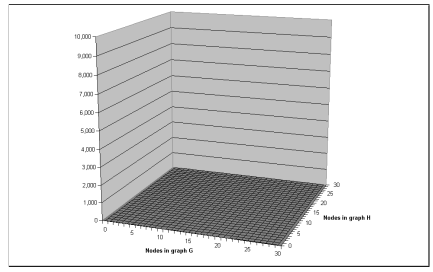
(a) True-Positive. Count 21,900, depth 0.68, recursions 2.29, refinements 103.26.



(b) True-Negative. Count 3,300, depth 0.62, recursions 1.71, refinements 33.11.



(c) False-Positive. Count 4,800, depth 0.67, recursions 1.91, refinements 51.49.



(d) False-Negative. None.

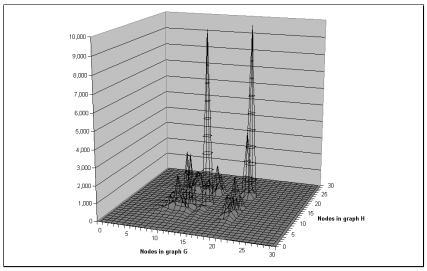
Figure 10.23: Match with 10% pose and scale noise.

10% pose and scale noise

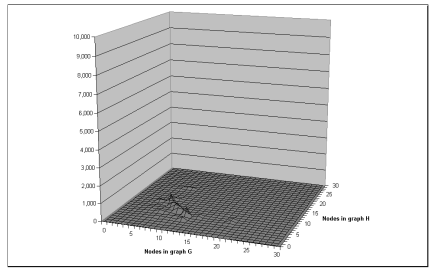
Accuracy	Error	Precision	Recall	F1
0.84	0.16	0.82	1.00	0.90

All controlling parameters are given at the start of this chapter.

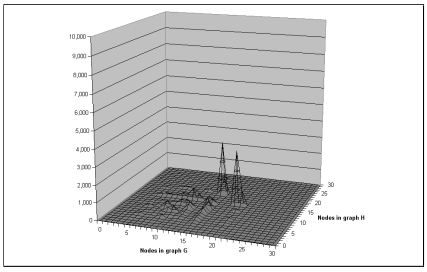
In accordance with the results of pose-only or scale-only noise, these instances are solved very close to those. As we have seen earlier, the runtimes of perfect matches are down, whereas the others are about the same. Since the controlling parameters in use only achieve 0.91 for F1 under no noise, this 0.90 is considered very good.



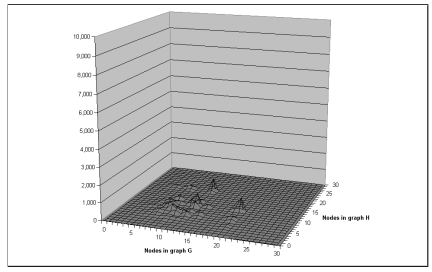
(a) True-Positive. Count 19,500, depth 0.61, recursions 1.87, refinements 70.11.



(b) True-Negative. Count 4,500, depth 0.51, recursions 1.18, refinements 27.83.



(c) False-Positive. Count 3,900, depth 0.60, recursions 1.51, refinements 42.71.



(d) False-Negative. Count 2,100, depth 0.56, recursions 1.48, refinements 46.52.

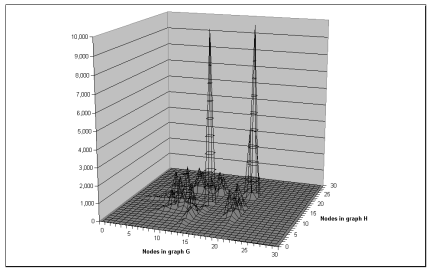
Figure 10.24: Match with 10% roll and pose noise.

10% roll and pose noise

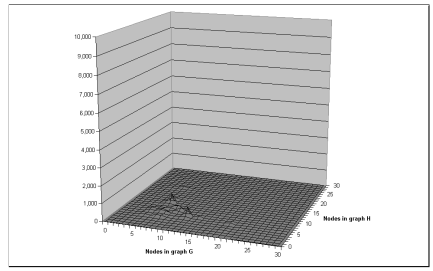
Accuracy	Error	Precision	Recall	F1
0.80	0.20	0.83	0.90	0.87

All controlling parameters are given at the start of this chapter.

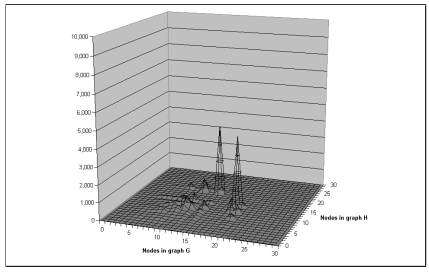
Similar to what we have seen earlier, roll noise has quite an impact on Recall. The results are still good, but Recall is down by 0.10. The time required to solve the perfect matches are down to half of those under no noise.



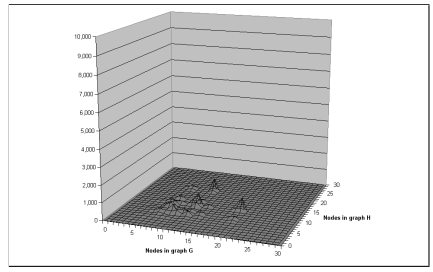
(a) True-Positive. Count 19,200, depth 0.61, recursions 1.91, refinements 73.82.



(b) True-Negative. Count 3,600, depth 0.50, recursions 1.22, refinements 25.74.



(c) False-Positive. Count 5,700, depth 0.63, recursions 1.75, refinements 51.17.



(d) False-Negative. Count 1,500, depth 0.55, recursions 1.50, refinements 51.75.

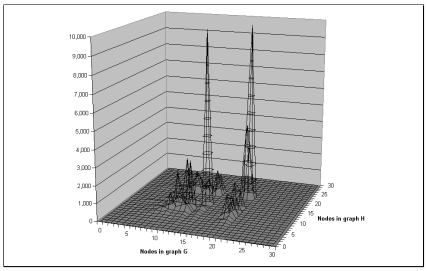
Figure 10.25: Match with 10% roll and scale noise.

10% roll and scale noise

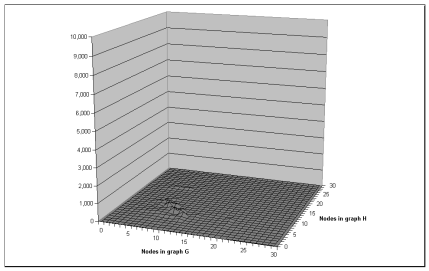
Accuracy	Error	Precision	Recall	F1
0.76	0.24	0.77	0.93	0.84

All controlling parameters are given at the start of this chapter.

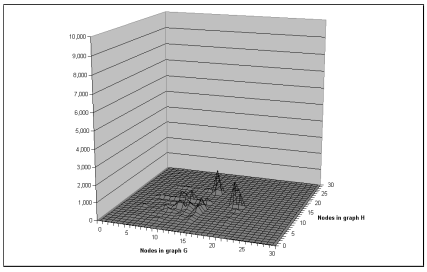
This combination of roll and scale noise achieve results that are just below those under roll and pose noise. This is contrary to what we expected, since we have seen earlier how little scale noise impacts the results. Runtimes are about the same.



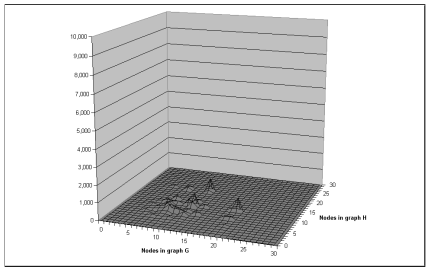
(a) True-Positive. Count 19,500, depth 0.61, recursions 1.90, refinements 72.24.



(b) True-Negative. Count 5,400, depth 0.52, recursions 1.20, refinements 25.48.



(c) False-Positive. Count 3,000, depth 0.59, recursions 1.52, refinements 46.99.



(d) False-Negative. Count 2,100, depth 0.51, recursions 1.18, refinements 36.06.

Figure 10.26: Match with 10% roll, pose and scale noise.

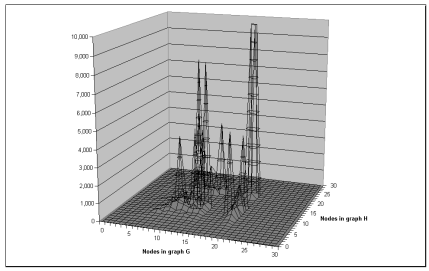
10% roll, pose and scale noise

Accuracy	Error	Precision	Recall	F1
0.83	0.17	0.87	0.90	0.88

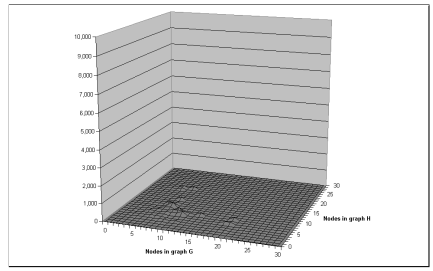
All controlling parameters are given at the start of this chapter.

The results are close to those without noise, an F1 of 0.88. Although Recall is down to 0.90, Precision is at an impressive 0.87. Runtimes are about the same as we have seen earlier, with perfect matches requiring on average only half the time compared to that without noise.

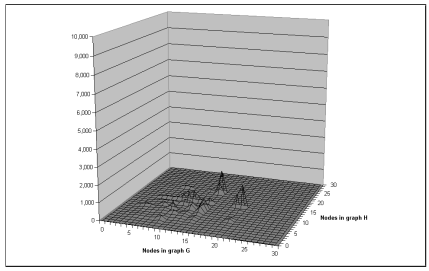
In a vision system that processes real images, this is the amount of noise that it should be able to handle. Beyond this point there should either be more recorded poses for each object, or there should be better pre-processing applied before object extraction.



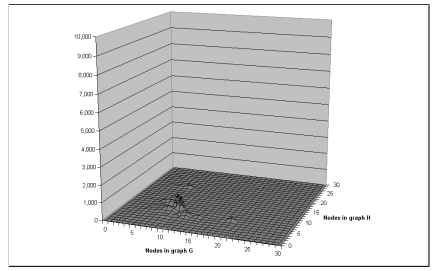
(a) True-Positive. Count 18,900, depth 0.71, recursions 2.34, refinements 97.49.



(b) True-Negative. Count 3,300, depth 0.55, recursions 1.40, refinements 29.72.



(c) False-Positive. Count 4,800, depth 0.70, recursions 2.08, refinements 54.37.



(d) False-Negative. Count 3,000, depth 0.57, recursions 1.46, refinements 33.65.

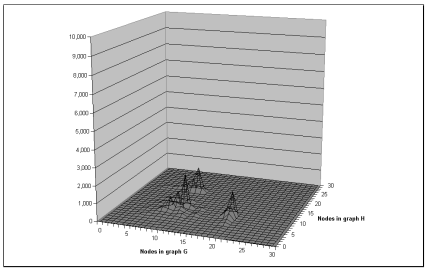
Figure 10.27: Match with 25% pose and scale noise.

25% pose and scale noise

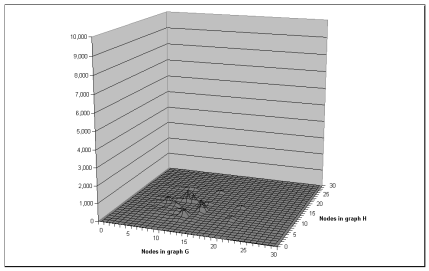
Accuracy	Error	Precision	Recall	F1
0.74	0.26	0.80	0.86	0.83

All controlling parameters are given at the start of this chapter.

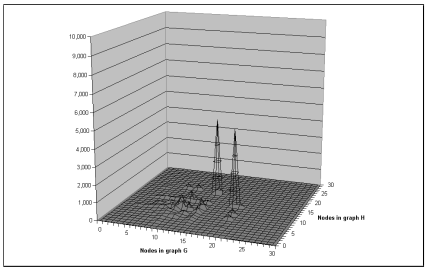
By comparison to the 25% pose noise instances, the scale noise has moved F1 down by 0.03.



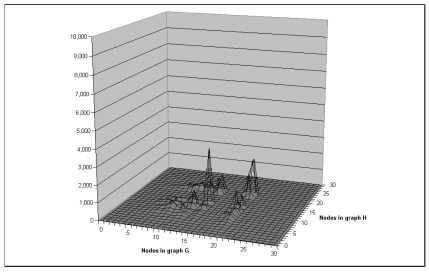
(a) True-Positive. Count 11,400, depth 0.54, recursions 1.68, refinements 59.37.



(b) True-Negative. Count 6,000, depth 0.33, recursions 0.79, refinements 22.75.



(c) False-Positive. Count 2,700, depth 0.62, recursions 1.80, refinements 67.12.



(d) False-Negative. Count 9,900, depth 0.34, recursions 0.98, refinements 38.38.

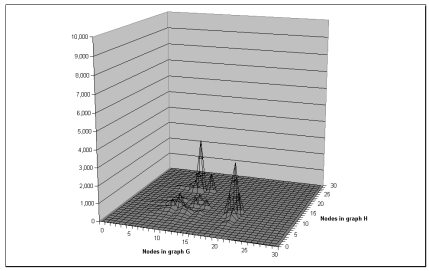
Figure 10.28: Match with 25% roll and scale noise.

25% roll and scale noise

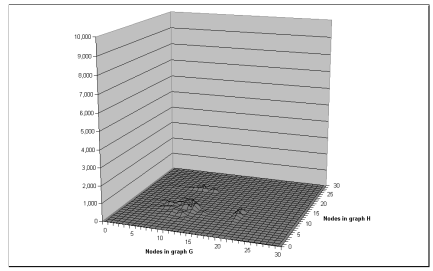
Accuracy	Error	Precision	Recall	F1
0.58	0.42	0.81	0.54	0.64

All controlling parameters are given at the start of this chapter.

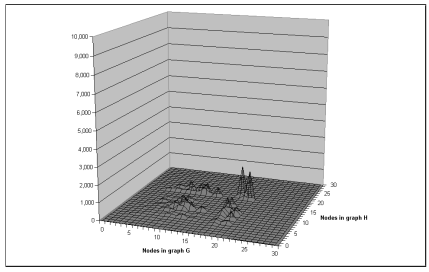
The 25% roll noise instances only achieve an F1 that is 0.02 higher, so again the scale noise seems negligible. This is exactly what we expected, and in accordance with the results under scale noise alone.



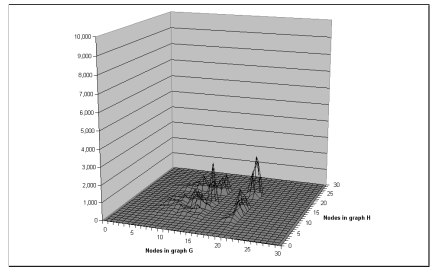
(a) True-Positive. Count 8,100, depth 0.54, recursions 1.63, refinements 59.44.



(b) True-Negative. Count 6,000, depth 0.35, recursions 0.86, refinements 25.78.



(c) False-Positive. Count 3,000, depth 0.59, recursions 1.63, refinements 60.67.



(d) False-Negative. Count 12,900, depth 0.37, recursions 1.02, refinements 38.83.

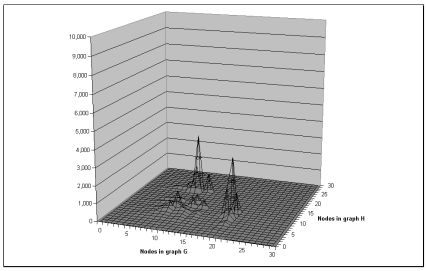
Figure 10.29: Match with 25% roll and pose noise.

25% roll and pose noise

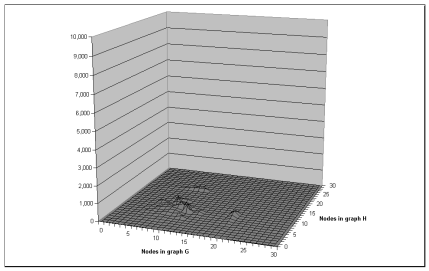
Accuracy	Error	Precision	Recall	F1
0.47	0.53	0.73	0.39	0.50

All controlling parameters are given at the start of this chapter.

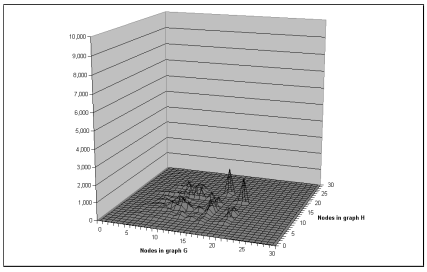
By the combination of this much roll and pose noise, the results suffer severely. At 0.50 F1 it is not applicable to any vision system that suffers under this much noise and requires reliable results.



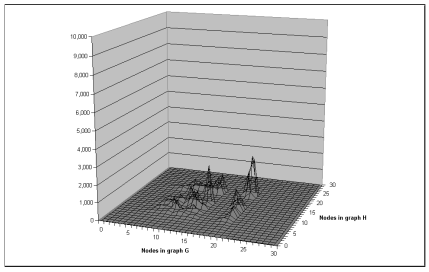
(a) True-Positive. Count 8,100, depth 0.52, recursions 1.53, refinements 56.78.



(b) True-Negative. Count 6,300, depth 0.36, recursions 0.92, refinements 26.37.



(c) False-Positive. Count 3,000, depth 0.64, recursions 1.91, refinements 67.44.



(d) False-Negative. Count 12,600, depth 0.36, recursions 0.98, refinements 37.40.

Figure 10.30: Match with 25% roll, pose and scale noise.

25% roll, pose and scale noise

Accuracy	Error	Precision	Recall	F1
0.48	0.52	0.73	0.39	0.51

All controlling parameters are given at the start of this chapter.

These results are equally poor to those without the applied scale noise. As we have already argued, this much noise requires different means to be overcome. Roll noise can be reduced through objects' primary axes of elongation in the source images, and pose noise can be solved by storing multiple poses for each object.

Chapter 11

Conclusion, part 2

This concludes our third contribution. This part illustrates in detail our proposition of object matching by combination of matching results from multiple graph domains.

In part 1 we introduce the attributed relation graph (ARG) and an efficient solution to solve the problem MINIMUM COST SUBGRAPH ISOMORPHISM when applied to ARG's in computer vision. In this part we use the inexact graph matcher to solve object recognition tasks on synthetic data.

Chapter 6 suggests that data contained in ARG's should remain of low complexity, and instead an "object" should be a set of such graphs that all describe the same image in separate domains. In chapter 7 we present an extractor that is able to find such objects in any 2-dimensional image, and in chapter 8 we describe a method of matching these by repeated application of the graph matcher algorithm.

Chapter 9 contains 3 separate illustrations of how the object matcher works, and finally chapter 10 contains results from a set of 172,500 object matches under a wide range of noise. The results achieved under the type of noise that is likely to appear in real vision systems (10% combinations of noise) are close to perfect.

The difficulty of solving instances of repeated patterns in a graph is somewhat overcome by the spatial information of the graph's edges, but where there is simultaneous similarity between both pattern primitive and spatial layout, the problem remains. Unlike other methods, our solution performs best when the graphs are large – the larger instances produce significantly more refinements to the cost matrix C because we are able to treat all graphs as fully connected.

Experiments show that the most difficult noise to overcome is what we refer to as rotation noise; rotation around the camera's z-axis. We suggest that any

application of our method should first rotate the unknown image by the object's primary axes of elongation after object-cropping to minimize this type of noise. One would also be able to improve performance by storing each prototype object using a set of poses, as opposed to our single pose, since this would minimize pose noise.

Because there is no interaction between matches being performed within a graph domain, there is nothing to prevent an implementation from doing this in complete parallel. The same is also true for the graph domains, since the results are only aggregated when all the domains have completed.

Any future work should investigate improvements to the graph matcher itself, as noted in chapter 5, but also the possibility of adding more graph domains when available. The most significant improvement, however, is expected to be contained in what currently allows parallelization – interaction between graph matches and graph domains to limit the choice of prototypes. Of course, this improvement is void if there is enough parallel processors to attempt all matches simultaneously.

Appendix A

Source code

A.1 Graph matcher

```
// Inititates the control structures needed in the recursion based graph matcher.
private CMorphism Initiate(CGraph oGraphG, CGraph oGraphH)
{
        // Allocate a cost matrix C of size —VR— x —VU—, where —VA— denotes the number of vertices in graph A.
        double[,] adCost = new double[oGraphG.Vertices.Length,
              oGraphH.Vertices.Length];

        // Allocate an exhaustion matrix.
        m_oExhausted = new Hashtable();

        // For each vertex VGi in VG do
        for(int i = 0; i < oGraphG.Vertices.Length; i++)
        {
                CVertex oVertexGi = oGraphG.Vertices[i];

                // For each vertex VHj in VH do
                for(int j = 0; j < oGraphH.Vertices.Length; j++)
                {
                        CVertex oVertexHj = oGraphH.Vertices[j];

                        // Request Domain Expert for cost C[i,j] of assigning VGi to VHj.
                        adCost[i, j] = m_oExpert.Ev(oVertexGi, oVertexHj);
                }
        }

        // Reset iteration marker to control overall cut-off.
        m_nMatchNum = 0;

        // Allocate a morphism M for graph G, graph H, an empty A, and c = 0.
        CMorphism oMorphism = new CMorphism(oGraphG, oGraphH, 0);

        // Call Recursion for morphism M and the cost matrix C, storing the returned morphism in M.
        oMorphism = Recurse(oMorphism, adCost);

        // Return M to calling process.
        return oMorphism;
}

// Perform a recursion step for the graph matcher.
private CMorphism Recurse(CMorphism oMorphism, double[,] adCost)
{
        // Mark recursion as exhausted.
        if(oMorphism.Associations.Count > 0)
                m_oExhausted.Add(GetExhaustKey(oMorphism.Associations), true);

        // Reference graphs.
        CGraph oGraphG = oMorphism.GraphG;
        CGraph oGraphH = oMorphism.GraphH;

        // Branch as implementation detail.
```

```
if(oMorphism.Associations.Count > 0)
{
        // For the last vertex-tuple (VGk, VHl) in A, do
        CAssociation oAssociation =
              (CAssociation)oMorphism.Associations[oMorphism.Associations.Count
               - 1];

        // For each vertex VGi in VG, do
        for(int i = 0; i < oGraphG.Vertices.Length; i++)
        {
                CVertex oVertexGi = oGraphG.Vertices[i];
                if(i == oAssociation.VertexG)
                        continue; // Ignore reflection.

                // For each vertex VHj in VH, do
                for(int j = 0; j < oGraphH.Vertices.Length; j++)
                {
                        CVertex oVertexHj = oGraphH.Vertices[j];
                        if(j == oAssociation.VertexH)
                                continue; // Ignore reflection.

                        // Request Domain Expert for cost adjustment to C[i,j] using the relation between
                              the edges (VGi, VGk) and (VHj, VHl).
                        adCost[i, j] += m_oExpert.Ee(oGraphG.FindEdge(i,
                              oAssociation.VertexG), oGraphH.FindEdge(j,
                              oAssociation.VertexH));
                }
        }
}

// Solve the assignment problem.
double[] adAvgG = new double[oGraphG.Vertices.Length];
double[] adAvgH = new double[oGraphH.Vertices.Length];
for(int i = 0; i < oGraphG.Vertices.Length; i++)
{
        for(int j = 0; j < oGraphH.Vertices.Length; j++)
        {
                adAvgG[i] += adCost[i, j] / oGraphG.Vertices.Length;
                adAvgH[j] += adCost[i, j] / oGraphH.Vertices.Length;
        }
}

ArrayList oCost = new ArrayList();
for(int i = 0; i < oGraphG.Vertices.Length; i++)
{
        for(int j = 0; j < oGraphH.Vertices.Length; j++)
        {
                oCost.Add(new CAssociation(i, j, adCost[i, j] - (adAvgG[i] +
                      adAvgH[j]) / 2));
        }
}
oCost.Sort();

// Recurse to required depth.
if((m_oExpert.RecursionDepth > 0 && oMorphism.Associations.Count >=
      m_oExpert.RecursionDepth) ||
        oMorphism.Associations.Count == oMorphism.GraphG.Vertices.Length) //
              Graph G is always the smaller.
{
        // Duplicate M and store as MP.
        CMorphism oMorphismP = new CMorphism(oMorphism);

        // In order of increasing value, for C[i,j] do
        foreach(CAssociation oAssociation in oCost)
        {
                // If (VGi, *) not in AMP and (*, VHj) not in AMP, do
                if(oMorphismP.FromG(oAssociation.VertexG) != null ||
                      oMorphismP.ToH(oAssociation.VertexH) != null)
                        continue;

                // Add tuple (VGi, VHj) to AMP.
                oMorphismP.Associations.Add(oAssociation);
        }

        // Assign to cMP the sum of C[i,j] by all tuples in AMP.
        foreach(CAssociation oAssociation in oMorphismP.Associations)
        {
                oMorphismP.Cost += adCost[oAssociation.VertexG,
                      oAssociation.VertexH];
        }

        // Request Domain Expert for cost adjustment to cMP.
        oMorphismP.Cost += m_oExpert.Em(oMorphismP);

        // If average assiciation cost exceeds match threshold, reject MP.
        if(oMorphismP.Cost / oMorphismP.Associations.Count >
              m_oExpert.MatchThreshold)
```

```
                return null;

            // Return MP to calling process.
            return oMorphismP;
        }
        else
        {
            // Allocate an empty morphism set M.
            ArrayList oMorphisms = new ArrayList();

            // Allocate a variable to count low-level recursions.
            int nRecursionNum = 0;

            // In order of increasing value for C[i,j], do
            foreach(CAssociation oAssociation in oCost)
            {
                // If (VGi, *) not in A and (*, VHj) not in A, do
                if(oMorphism.FromG(oAssociation.VertexG) != null ||
                    oMorphism.ToH(oAssociation.VertexH) != null)
                        continue;

                // If C[i,j] does not exceed association threshold, do
                if(adCost[oAssociation.VertexG, oAssociation.VertexH] >
                    m_oExpert.RecursionThresholdA *
                    oMorphism.Associations.Count +
                    m_oExpert.RecursionThresholdB)
                        break; // Keep in mind that traversal is ordered.

                // If A + (VGi, VHj) is not exhausted, do
                if(m_oExhausted.ContainsKey(GetExhaustKey(oMorphism.Associations,
                     oAssociation)))
                        continue; // Solves thrashing behaviour (see Mackworth77 and Shapiro81).

                // If active Domain Expert does not oppose, do
                if(!m_oExpert.R(oMorphism, oAssociation))
                        continue;

                // If recursion is not cut off, do
                if(m_oExpert.RecursionCutOff > 0 && ++nRecursionNum >
                    m_oExpert.RecursionCutOff)
                        break; // Again; traversal is ordered.

                // If match is not cut off, do
                if(m_oExpert.MatchCutOff > 0 && ++m_nMatchNum >
                    m_oExpert.MatchCutOff)
                        break; // Again; traversal is ordered.

                // Duplicate M and store as MR.
                CMorphism oMorphismR = new CMorphism(oMorphism);

                // Add tuple (VGi, VHj) to AMR.
                oMorphismR.Associations.Add(oAssociation);

                // Duplicate C and store as CR.
                double[,] adCostR = new double[adCost.GetLength(0),
                    adCost.GetLength(1)];
                for(int i = 0; i < oGraphG.Vertices.Length; i++)
                {
                        for(int j = 0; j < oGraphH.Vertices.Length; j++)
                        {
                                adCostR[i, j] = adCost[i, j];
                        }
                }

                // Recurse for morphism MR and cost matrix CR, storing the returned morphism in MR.
                oMorphismR = Recurse(oMorphismR, adCostR);

                // Add morphism MR to set M, if any.
                if(oMorphismR != null)
                        oMorphisms.Add(oMorphismR);
            }

            // If morphism set M is not empty, do
            if(oMorphisms.Count == 0)
                return null;

            // Sort morphism set.
            oMorphisms.Sort();

            // Return min(M) to calling process.
            return (CMorphism)oMorphisms[0];
        }
}
```

Listing A.1: Graph matcher in C#

Bibliography

[1] Royal Military Academy Signal & Image Centre. Contribution of rma/sic to hispars project - euclid rtp6.2. `http://www.sic.rma.ac.be/Projects/Hispars/`, 1993.

[2] H. Shao, T. Svoboda, T. Tuytelaars, and L. Van Gool. Hpat indexing for fast object/scene recognition based on local appearance. In Michael Lew, Thomas Huang, Nicu Sebe, and Xiang (Sean) Zhou, editors, computer lecture notes on Image and video retrieval, pages 71–80. Springers, July 2003.

[3] M. A. Eshera and K. S. Fu. An image understanding system using attributed symbolic representation and inexact graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(5):604–617, 1986.

[4] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(5):530–535, May 1997.

[5] R. E. Blake. The use of scott's lattice theory as a basis for combining items of evidence. Pattern Recognition Letters, 7:151–155, 1988.

[6] R. E. Blake and P. Boros. The extraction of structural features for use in computer vision. In Proceedings of the Second Asian Conference on Computer Vision, Singapore, December 1995.

[7] D. G. Lowe. Object recognition from local scale-invariant features. In Proceedings of the Seventh International Conference on Computer Vision (ICCV'99), pages 1150–1157, Washington, DC, USA, 1999. IEEE Computer Society.

[8] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(4):377–388, 1996.

[9] H. T. Hajiaghayi and N. Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth, 2002.

[10] L. G. Shapiro and M. Haralick. Structural descriptions and inexact matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, 3(5):504–519, September 1981.

[11] T. Pavlidis. Structural Pattern Recognition. Springer, New York, NY, USA, 1977.

[12] K. S. Fu. Syntactic Pattern Recognition and Applications. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[13] K. S. Fu. A step towards unification of syntactic and statistical pattern recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 5(2):200–205, March 1983.

[14] C. Soanes and S. Hawker. Compact Oxford English Dictionary of Current English. Oxford University Press, 2005.

[15] P. Foggia, R. Genna, and M. Vento. Introducing generalized attributed relational graphs (gargs) as prototypes of args. In Proceedings of the 2nd IAPR Workshop on Graph-based Representations (GbR99), Haindorf, Austria, 1999.

[16] J. Rocha and T. Pavlidis. A shape analysis model with applications to a character recognition system. IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(4):393–404, 1994.

[17] H. Nishida. Shape recognition by integrating structural descriptions and geometrical/statistical transforms. Computer Vision and Image Understanding, 64:248–262, 1996.

[18] J. Hsu and S. Wang. A machine learning approach for acquiring descriptive classification rules of shape contours. Pattern Recognition, 30(2):245–252, 1997.

[19] R. S. Michalski. Pattern recognition as rule-guided inductive inference. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2(4):349–361, 1980.

[20] W. H. Tsai and K. S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Transactions on Systems, Man, and Cybernetics, 9(22):757–768, 1979.

[21] W. H. Tsai and K. S. Fu. Subgraph error-correcting isomorphisms for syntactic pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics, 13(1):48–62, 1983.

[22] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An efficient algorithm for the inexact matching of arg graphs using a contextual transformational

model. In Proceedings of the 13th International Conference on Pattern Recognition, pages 180–184, Washington, DC, USA, 1996. IEEE Computer Society.

[23] S. A. Cook. The complexity of theorem-proving procedures. In Proc. 3rd Ann. ACM Symp. on Theory of Computing, pages 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[24] R. M. Karp. Reducibility among combinatorial problems. Complexity of Computer Computations, pages 85–103, 1972.

[25] M. R. Garey and D. S. Johnson. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1979.

[26] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In Proc. 13th Ann. Symp. on Switching and Automata Theory, pages 125–129, Long Beach, CA, USA, 1972. IEEE Computer Society.

[27] L. Stockmeyer. The polynomial-time hierarchy. Theoretical Computer Science, 3(1):1–22, October 1977.

[28] L. G. Valiant. The complexity of computing the permanent. Theoretical Computer Science, 8(2):189–201, April 1979.

[29] S. Z. Li. Matching: invariant to translations, rotations and scale changes. Pattern Recognition, 25:583–594, 1992.

[30] W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17:749–764, August 1995.

[31] A. Rosenfeld and A. Kak. Digital Picture Processing, volume 2. Academic Press, Inc., Orlando, FL, USA, 1982.

[32] L. S. Davis. Shape matching using relaxation techniques. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1(1):60–72, January 1979.

[33] A. Rosenfeld, R. A. Hummel, and S. W. Zucker. Scene labeling by relaxation operations. IEEE Transactions on Systems, Man, and Cybernetics, 6(6):420–433, June 1976.

[34] R. E. Blake. A partial ordering for relational graphs applicable to varying levels of detail. Pattern Recognition Letters, 11:305–312, 1990.

[35] R. E. Blake. Partitioning graph matching with constraints. Pattern Recognition, 27(3):439–446, 1994.

[36] R. E. Blake and A. Juozapavicius. Convergent matching for model-based computer vision. Pattern Recognition, 36:527–534, 2003.

[37] M. A. Eshera and K. S. Fu. A graph distance measure for image analysis. IEEE Transactions on Systems, Man, and Cybernetics, 14(3):398–408, May/June 1984.

[38] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Graph matching: A fast algorithm and its evaluation. In Proceedings of the 14th International Conference on Pattern Recognition, pages 1582–1584, Washington, DC, USA, 1998. IEEE Computer Society.

[39] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In Conference on Computer Vision and Pattern Recognition, pages 1000–1006, Puerto Rico, 1997.

[40] Y. Lamdan and H. J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In Proceedings of the 2nd International Conference on Computer Vision, pages 238–249, Washington, DC, USA, 1988. IEEE Computer Society.

[41] D. Forsyth, J. L. Mundy, A. Zisserman, and C. M. Brown. Invariance - a new framework for vision. In Proceedings of the 3rd International Conference on Computer Vision, pages 598–605, Washington, DC, USA, 1990. IEEE Computer Society.

[42] Y. Lamdan, J. T. Schwartz, and H. J. Wolfson. Affine invariant model-based object recognition. IEEE Transactions on Robotics and Automation, 6(5):578–589, 1990.

[43] D. Clemens and D. Jocobs. Space and time bounds on indexing 3-d models from 2-d images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(10):1007–1017, 1991.

[44] C.A. Rothwell, A. Zisserman, J.L. Mundy, and D.A. Forsyth. Efficient model library access by projectively invariant indexing functions. In Proceedings of Computer Vision and Pattern Recognition '92, pages 109–114, Long Beach, CA, USA, 1992. IEEE Computer Society.

[45] F. Stein and G. Medioni. Structural indexing: efficient 3d object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2):125–145, December 1992.

[46] F. Stein and G. Medioni. Structural indexing: efficient 2d object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(12):1198–1204, December 1992.

[47] J. H. Freidman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw., 3(3):209–226, 1977.

[48] L. Van Gool, P. Kempenaers, and A. Oosterlinck. Recognition and semi-differential invariants. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 454–460, Washington, DC, USA, 1991. IEEE Computer Society.

[49] Z. Zhang, R. Deriche, O. Faugeras, and Q. T. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. Artificial Intelligence Journal, 78:87–119, 1995.

[50] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. Annals of Mathematical Statistics, 35:876–879, 1964.

[51] J. S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. Advances in neural information processing systems 2, pages 211–217, 1990.

[52] C. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[53] C. Peterson and B. Soderberg. A new method for mapping optimization problems onto neural networks. International Journal of Neural Systems, 1:3–22, 1989.

[54] Microsoft Corporation. Microsoft visual c# developer center. `http://msdn.microsoft.com/csharp/`.

[55] Microsoft Corporation. Microsoft .net. `http://www.microsoft.com/net/`.

[56] Microsoft Corporation. Microsoft directx. `http://www.microsoft.com/windows/directx/`.

[57] Microsoft Corporation. The common language runtime (clr). `http://msdn.microsoft.com/netframework/programming/clr/`.

[58] Inc. Silicon Graphics. Opengl. `http://www.sgi.com/products/software/opengl/`.

[59] Microsoft Corporation. X file reference. `http://msdn.microsoft.com/library/default.asp?HOWPUBLISHED=/library/en-%us/directx9_c/dx9_graphics_reference_d3dx_x_file.asp`.

[60] R. Jain, R. Kasturi, and B. G. Schunck. Machine Vision. McGraw-Hill, New York, NY, USA, 1995.

[61] James Bruce. Realtime machine vision perception and prediction. `http://www.cs.cmu.edu/~jbruce/cmvision/`, 2000.

[62] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. Journal of the Association for Computing Machinery, 13(4):471–494, October 1966.

[63] M-K. Hu. Visual pattern recognition by moment invariants. IRE Transactions on Information Theory, 8(2):179–187, 1962.

[64] J. Wood. Invariant pattern recognition : A review. Pattern Recognition, 29(2):1–17, January 1996.