AN IMPROVED GRAPH MINING TOOL AND ITS APPLICATION TO OBJECT DETECTION IN REMOTE SENSING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÜMİT RUŞEN AKTAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2013

Approval of the thesis:

AN IMPROVED GRAPH MINING TOOL AND ITS APPLICATION TO OBJECT DETECTION IN REMOTE SENSING

submitted by **ÜMİT RUŞEN AKTAŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** ______________

Prof. Dr. Fatoş T. Yarman Vural
Supervisor, **Computer Engineering Department, METU** ______________

Examining Committee Members:

Prof. Dr. Faruk Polat
Computer Engineering Department, Bilkent University ______________

Prof. Dr. Fatoş T. Yarman Vural
Computer Engineering Department, METU ______________

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU ______________

Assist. Prof. Dr. Sinan Kalkan
Computer Engineering Department, METU ______________

Dr. Onur Pekcan
Civil Engineering Department, METU ______________

Date: ______________

ABSTRACT

AN IMPROVED GRAPH MINING TOOL AND ITS APPLICATION TO OBJECT DETECTION IN REMOTE SENSING

AKTAŞ, ÜMİT RUŞEN
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Fatoş T. Yarman Vural

September 2013, 66 pages

In many graph-based data mining tools, the use of numeric values as attributes in graphs is very limited. Most algorithms require pre-processing of the attributes, which often involves discretization into bins and embedding group names in the input graph(s). In this thesis, we tackle this problem by utilizing all attributes as is, and directly incorporating them into the pattern mining process. In order to implement our method, we modify an existing graph-based knowledge discovery algorithm, SUBDUE, by adding it the capability of working with continuous and discrete data vectors of any dimension. In addition, we propose an object detection framework using improved SUBDUE in its object matching step. This system detects repetitive objects such as buildings and airplanes in satellite images, once the user specifies a sample target by drawing a bounding box around it. Experiments on artificial and real datasets show that our contributions result from a robust and flexible approach that can generalize over a vast number of problems.

Keywords: Graph Mining, Data Mining, Substructure Discovery

ÖZ

GELİŞTİRİLMİŞ BİR GRAFİK MADENCİLİĞİ ARACI VE UYDU GÖRÜNTÜLERİNDE NESNE TESPİTİNE UYGULANMASI

AKTAŞ, ÜMİT RUŞEN
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Fatoş T. Yarman Vural

Eylül 2013 , 66 sayfa

Pek çok grafik madenciliği uygulamasında, sayısal değerlerin grafiklerin içinde özellik olarak kullanılması oldukça zor bir problem olarak karşımıza çıkmaktadır. Çoğu algoritma, bu bilginin, örneklerin gruplanmasını ve grup isimlerinin grafiğe eklenmesini gerektiren bir ön işleme evresinden geçirilmesini zorunlu kılmaktadır. Bu tezde, sayısal değerleri olduğu gibi bırakıp desen arama işleminde kullanarak bu probleme bir çözüm getirmeyi hedeflemekteyiz. Yöntemimizi gerçeklemek için, halihazırda var olan SUBDUE isimli bir grafik tabanlı bilgi keşfetme algoritmasını, onu herhangi bir boyuttaki sayısal değerler dizisi ile çalışacak hale getirerek değiştirmiş bulunmaktayız. Ek olarak, geliştirilmiş SUBDUE'yi nesne eşleme aşamasında kullanan bir nesne tespit etme sistemi önermekteyiz. Bu sistemde, kullanıcı belirlediği hedefin çevresine bir kutu çizdikten sonra, uydu görüntüsündeki uçak ve bina gibi nesneler tespit edilebilmektedir. Yapay ve gerçek veri setlerindeki deneyler, alana sunduğumuz katkıların, pek çok probleme genellenebilecek gürbüz ve esnek bir yaklaşımın sonucu olduğunu ortaya koymaktadır.

Anahtar Kelimeler: Grafik Madenciliği, Veri Madenciliği, Altyapı keşfi

To those who resist for a better world.

#direngezi

ACKNOWLEDGMENTS

It would have been impossible to write this thesis for me if I was not surrounded by the awesome people I have in my life. While I would like to mention some of the names here, this is far from a complete list.
To begin with, I would like to express my deepest graditude to my supervisor Prof. Dr. Fatoş T. Yarman Vural for her guidance and effort in the preparation of this thesis. I thank her for providing me the motivation and discipline in every way possible.
I would like to thank my darling Başak Meral for her continuous support and patience. She has been the absolute source of joy and light in my life.
I feel very privileged to have my dear sister Berfin for helping me out in the hardest of times. I, certainly, would not be the person I am without her help, kindness, and compassion.
I am very lucky to have my beloved family always supporting me. I would like to thank my dear mother Güllü Aktaş and my dear father Ali Haydar Aktaş for their love, sacrifices and care.
I thank Havelsan and Turkcell Akademi for their vast support to the study presented in this document. In addition, I would like to acknowledge the financial support by TÜBİTAK 2210 Scholarship Program.
Finally, I would like to thank my professors, colleagues and friends for their mere existence in my life. In addition, I appreciate the encouragement and insightful comments provided by the jury members, Prof. Dr. Faruk Polat, Assoc. Prof. Dr. Pınar Karagöz, Assist. Prof. Dr. Sinan Kalkan and Dr. Onur Pekcan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

SI	Subgraph Isomorphism
GI	Graph Isomorphism
AGM	Apriori-based Graph Mining
FSG	Frequent Subgraph Discovery
gSpan	Graph-based Substructure Pattern Mining
DFS	Depth-First Search
GASTON	GrAph/Sequence/Tree extractiON
FFSM	Fast Frequent Subgraph Mining
MoFa	Molecular Fragment Miner
SeuS	Structure extraction using Summaries
ARG	Attributed Relational Graphs
SIFT	Scale Invariant Feature Transform
MDL	Minimum Description Length
ExtSUBDUE	Extended SUBDUE
GODFREY	Generalized Object Detection Framework for Remotely Sensed Images
Sub	Substructure, Subgraph

CHAPTER 1

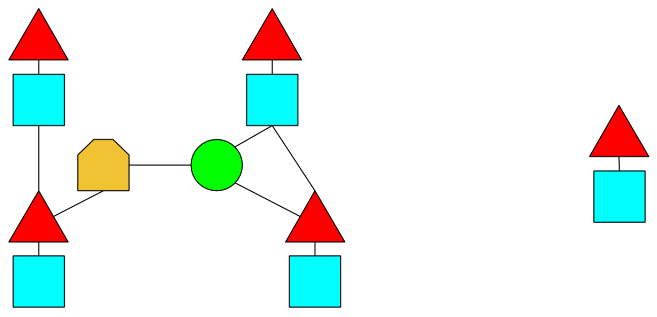
INTRODUCTION

Data mining is the process of learning previously unknown knowledge from raw datasets. It lies at the heart of unsupervised or semi-supervised learning on large databases, and has a vast number of applications on many domains. The goal is to extract the regularities from the data, which define the dataset's characteristics in a machine-readable format. While data mining has many applications unstructured domains such as independent measurements, the focus of this thesis is on structural domains. DNA and The Internet are such examples where the basic entities (molecules and computers) are semantically connected.

In order to uncover the hidden knowledge in structural data, one prominent option is to represent the data with a graph, and then search for frequent subgraphs (pattern, substructure) in it. As a result, data mining on structural data has gained momentum with the emerging of graph mining algorithms [1, 2]. These frequent patterns are the essence of the input graph, i.e. they model important partial structural relationships. For example, the abstract graph G in figure 1.1a has the frequent pattern P given in figure 1.1b. This pattern occurs four times throughout G.

A graph consists of a set of vertexes which may correspond to basic objects, and a set of edges which define pair-wise relations of these nodes. The advantage of using graphs as the basic representation is that complicated relationships in the data can be modeled with these simple building blocks. DNA, for instance, is an inherently structural data that can be modeled with graphs. The four nucleobases (Adenine, Guanine, Thymine, Cytosine) can correspond to vertices, while the bonds between them are represented with edges. To extract the valuable knowledge from the input graph(s), repetitive substructure search is a popular method implemented extensively in graph mining algorithms [1, 3, 4, 2, 5].

The problem with most of the existing approaches [1, 3, 4, 5] is that they generally are not capable of detecting patterns that occur throughout the input graph(s) with small variations. The available algorithms are highly optimized towards finding all frequent patterns. Additionally, they do not tolerate any changes in the structure of their instances. SUBDUE [2] is a knowledge discovery algorithm that aims to solve



(a) Input graph G (b) Pattern P

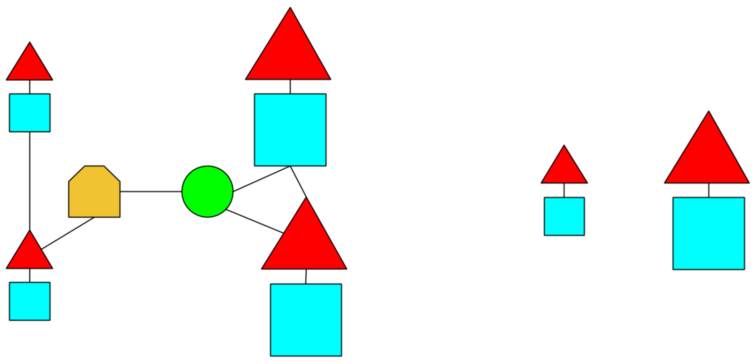
Figure 1.1: Frequent subgraph example 1.

this problem by introducing tolerance matching in its discovery step. SUBDUE can find frequent substructures even though a few of their vertices/edges are missing in some instances.

The problem we address in this thesis is that most of the approaches in the literature are not able to work with real data, and are limited to abstract labels in the graphs. For example, the pattern in figure 1.1b should be interpreted as "a triangle connected with a square" in many applications. This interpretation does not include any numeric (size/length/area) information about neither object. We believe that accommodating such numeric knowledge in the graph leads to a more powerful representation. For example, consider the slightly different graph G' in figure 1.2a. For a standard graph miner to discover these two subgraphs, the objects on the right side of G' should manually be labeled as "Big", and the ones on the left as "Small". Otherwise, only one repetitive subgraph given in figure 1.1b will be found. In our methodology, a numeric feature of each object can be calculated and embedded within the object's label. For example, a "Triangle" with area "32" can be represented with the label "Triangle 32". With no further modification, the proposed algorithm will come up with the two frequent patterns in figure 1.2b.

The main contribution of this work is that numeric data vectors of any length can be used directly in the input graph to define entities, allowing the pattern mining procedure to work on these labels. We propose ExtSUBDUE (Extended SUBDUE) as an improved version of SUBDUE that can work with numeric labels. While this improvement comes with increased complexity, it is more accurate than alternative methods, as shown in chapter 5. In addition, we propose an object detection framework for remote sensing images. Our method can detect instances of a target object class when the user specifies a sample pattern. We solve the object matching problem efficiently with ExtSUBDUE, using a graph abstraction of the input image.

In order to show the effectiveness of ExtSUBDUE, we implement a graph-based object detection framework for remotely sensed images, GODFREY. This system is capable



(a) Input graph G' (b) Patterns $P'_1 and P'_2$

Figure 1.2: Frequent subgraph example 2.

of detecting small objects in satellite images such as cars, airplanes, buildings, trees, once the user specifies a template pattern. The target pattern is searched for in the image using ExtSUBDUE, which works over a graph abstraction of the input image.

In the following chapter, we discuss features of the existing approaches in the literature, followed by theoretical basis and the variations of SUBDUE, which is the tool our graph mining tool builds on. In chapter 3, we give a detailed overview of the SUBDUE algorithm. Then, we propose ExtSUBDUE (Extended SUBDUE), which is a modified version of SUBDUE. In chapter 4, we describe our graph-based object detection framework, GODFREY in detail. Experiments on both artificial and real datasets are presented in chapter 5. Finally, a conclusion and our further intentions to improve ExtSUBDUE and GODFREY are given in chapter 6.

CHAPTER 2

RELATED WORK

Due to their expressive power, graphs have been increasingly used in data mining applications. Their wide use is a result of their natural occurrences in many domains including web mining [6], chemical compound analysis [5], security threat detection [7] and more. Perhaps the most intuitive interpretation of graph mining and analysis is frequent subgraph (substructure, pattern) discovery. Based on the nature of the dataset, the problem can be attacked in two different ways:

- Detection of frequent subgraphs across a graph database. This version of the problem can be formulated as follows:

  Given a graph dataset $DS = G_1, G_2, ..., G_n$, we aim to find any subgraph sg s.t. $support(sg) \geq minimumSupport$.

  $support(sg) = \frac{\text{number of graphs in } DS \text{ including an instance of } sg}{\text{total number of graphs in } DS}$.

- Detection of frequent subgraphs within a single graph G. $support(sg)$ is then considered as number of instances (embeddings) of sg in G. $minimumSupport$ is the minimum number of instances for a subgraph to be deemed important.

While the two problems look alike, the methodologies developed for the former problem generally do not scale well to the latter. On the other hand, latter algorithms are able to solve first problem with slight modifications. In both cases, detecting the existence of a subgraph within a graph essentially reduces to Subgraph Isomorphism problem (SI), which is known to be NP-Complete. As a result, the main effort in efficient algorithms is focused on reducing the number of subgraph isomorphism tests, and only resort to run SI check on highly ambiguous cases. Variants of these problems include discovery of overlapping/non-overlapping instances and and inexact matching, i.e. tolerating slight changes in the form of the substructure in SI tests.

2.1 Frequent Pattern Discovery

Since graphs are used to model complex relationships with basic building blocks such as nodes, edges and labels, frequent substructure discovery has been one of the hot topics in the last decade. Frequent pattern discovery algorithms aim to find all subgraphs that have a minimum support value within a graph or a set of graphs. Many algorithms have been proposed for the purpose of developing efficient and scalable solutions. Based on how they approach the problem, they can be divided into two groups: complete and greedy approaches. While complete methods are guaranteed to find all frequent subgraphs, they tend to report too many final substructures [8, 9, 10, 11, 4, 12, 1]. On the other hand, greedy methods [2, 13] may miss some frequent substructures as a result of their constrained search strategies.

A naive approach in frequent pattern mining in graphs is to enumerate all possible subgraphs having up to k vertices. First level includes one-vertex subgraphs, second level consists of two-vertex patterns, etc. Then, each candidate subgraph can be searched for in the graph database to decide whether it has enough support. This approach is problematic and needs improvement for a number of reasons. From a practical standpoint, such an enumeration scheme requires time and space in exponential order with k. Therefore, a pruning mechanism is needed to filter out nodes with low support values from the search tree. In addition to this, at any given level, redundant substructures will be generated. Checking whether that subgraph has already been generated requires Graph Isomorphism (GI) tests with previous candidates, for which no known polynomial-time algorithm exists [14]. Allowing redundancies in the candidate generation step leads to many unnecessary SI tests over the whole dataset, which are also costly. Due to these reasons, complete algorithms either attempt to reduce number of redundant subgraphs generated [10], or they elaborate depth-first search to fully avoid candidate generation [1]. Heuristic algorithms attack the problem by limiting number of candidates generated at each level [2].

Inokuchi et al. proposed an algorithm called Apriori-based Graph Mining (AGM) in 2000 [15], which can be considered as an early example of frequent pattern miners. AGM uses level-wise expansion of subgraphs in the candidate generation step. At each level of this process, two compatible subgraphs of size k are merged to form new subgraphs of size $k + 1$. Similar to the work of Agrawal et al. [16], AGM adopts the idea underlying market basket analysis by generating a coding mechanism to keep the subgraphs in lexicographical order. Another work that embraces level-wise Apriori candidate generation idea is presented in [10] by Kuramochi et al. Their algorithm, Frequent SubGraph (FSG), uses canonical labeling and subgraph joining at candidate generation step. In addition, it scales to large graphs in linear order with graph size.

gSpan (graph-based Substructure pattern mining) [1] is an efficient graph-based frequent pattern mining tool developed by Yan et al. in 2002. It avoids candidate generation by utilizing a depth-first procedure to enumerate possible substructures. For each substructure to be tested against support, a *minimum DFS code* is generated. This unique code is the canonical label of the substructure, and is the same for two isomorphic graphs, reducing graph isomorphism test to the much simpler DFS code comparison. Although the worst-case complexity of canonical label generation is exponential, it is sufficiently efficient for many types of real graphs. Variants of gSpan have been developed and applied to domains such as procedural abstraction in embedded systems [12]. CloseGraph [17] is another expansion of gSpan. It is a modified version of gSpan that only reports *closed* substructures, therefore reducing the output size. A subgraph is *closed* if there is no supergraph of it with the same support.

In 2002, Borgelt and Berthold proposed an algorithm which outperforms gSpan on a number of synthetic and real datasets [5]. MoFa, Molecular Fragment Miner, mines patterns in graphs by means of an extension-based search procedure. In addition to support-based and size-based pruning rules which are used extensively by its competitors [1, 2], MoFa applies a form of structural pruning to reduce its search space. The purpose of the latter rule is to consider each node set only once to prune already considered substructures. This efficient enumeration plan helps their algorithm to process substantially lower number of nodes in the search tree. Possible extensions at each level, on the other hand, are calculated from the embeddings of the parent substructure. This design choice results in very high space requirements, as each embedding (instance) of the currently considered subgraphs should be stored. Similar to the idea of quick-start in GASTON [4], and pre-defined substructures in SUBDUE [2], MoFa first discovers the basic units in the database. An initial compression is performed with these units, allowing the algorithm to perform actual discovery faster. When compared to GASTON [4], FFSM [3] and gSpan [1], MoFa is specifically optimized to work on molecular fragment mining, as [18] verifies.

An interactive graph-mining algorithm called SeuS is proposed in [8]. In this work, Ghazizadeh et al. provide a way to refine the threshold parameter by showing the user approximate, intermediate structures. This procedure allows the user to optimize the support threshold without the expensive exact discovery process. They use summaries of the data to limit disk access to the actual database, therefore scaling well to large datasets. Fast, approximated versions of final substructures are extracted from these summaries. When the user sets the final threshold based on these intermediate results, the expensive discovery process is started.

A novel subgraph mining framework, FFSM, is presented by Huan et al. in [3]. The contributions of FFSM, Fast Frequent Subgraph Mining, are: 1) a new canonical form and two efficient candidate generation functions, FFSM-Join and FFSM-Extension, 2) graph enumeration rules that generate a low number of redundant subgraphs, 3) use of embeddings to avoid exponential SI tests. As opposed to minimal canonical codes in [15, 10], FFSM utilizes maximal codes. FFSM enumerates all possible subgraphs using FFSM-Join and FFSM-Extension functions. Similarly with [5], canonical matrices of each embedding is stored to quickly match substructures. Experiments on both synthetic and real datasets reveal a large speed-up over gSpan [1], due to largely a more precise coding mechanism and an efficient subgraph isomorphism function. However, as the size of the substructures decreases, the positive effect of using embeddings is reduced, and gSpan and FFSM tend to perform at a similar level [18]. While use of embeddings and canonical matrices clearly pays off in terms of time performance, it may yield memory problems as the number of stored embeddings are bound to explode for larger graphs.

In [4], Nijssen et al. introduce the idea of enumerating subgraphs based on their types. Their algorithm, GASTON, enumerates paths, trees and cyclic graphs separately. This allows them to incorporate effective methods for simple graphs such as paths and trees, for which polynomial-time isomorphism tests exist. The **Quickstart** principle in their context means starting candidate generation with simple structures such as paths and trees, leading to an efficient pattern growth plan. For isomorphism tests on general graphs, they use Nauty [19], which is the current state-of-the-art in graph isomorphism. A head-to-head comparison with gSpan suggests that GASTON outperforms gSpan due to its ability to enumerate paths and trees without duplicates [18].

In addition to the performance constraints, the number of substructures reported by algorithms poses a potential problem. While previously mentioned programs except CloseGraph [17] report all substructures having enough support, Huan et al. presented a novel method that mines only *maximal* frequent subgraphs [9]. A *maximal* subgraph is not a part of any other frequent subgraph. This is in accordance with the concept of *interesting* substructures in SUBDUE, the ones which may be more significant to the user. SUBDUE [2] defines the quality of a subgraph by its ability to compress the input graph(s). The number of substructures in the output is drastically reduced, while keeping the most important ones intact.

While working on a slightly different problem, Kuramochi et al. developed two algorithms to find frequent substructures in a large single graph setting [11]. Their algorithms, depth-first *VSIGRAM* and breadth-first *HSIGRAM* can cope with graphs with more than 100.000 vertexes. Except SUBDUE, the methods discussed up to now are optimized toward working on a database of graphs, rather than a single graph. In a one large graph setup, it is necessary to count the possibly non-overlapping instances of a subgraph to determine its support. In a multiple-graph database, the support is formulated as number of graphs having an instance of the subgraph, regardless of the number of instances in each. In [11], edge-disjoint instances considered for the sake of efficient enumeration. For two subgraphs to be edge-disjoint, they must not have any common edges. In the subgraph isomorphism test, they use canonical labels and vertex invariants to get a unique code for each pattern. To determine the number of non-overlapping instances, they create a graph in which embeddings map to the vertices, and edges link overlapping ones. Frequency counting is thus reduced to the Maximum Independent Set (MIS) problem. Given a graph $G = (V, E)$ and a subset

$S \subset V$, S is **independent** if $\forall v_i, v_j \in S, (v_i, v_j) \notin E$. S is a maximum independent set if it includes as many vertices from V as possible. Since this problem is NP-Complete, they propose two approximate solutions in addition to the exact solution.

Following the footsteps of gSpan, Hellal and Romdhane proposed the NODAR [20] algorithm that successfully embodies minimum DFS codes in its candidate generation step. NODAR, Non-Overlapping embeDding based grAph mineR, works in a single large graph to find non-overlapping instances of frequent substructures. To prune the depth-first search tree, they use the anti-monotonic SMNOES (Size of Maximum Non-Overlapping Embedding Set) heuristic. Anti-monotonicity of a measure in this context means that support of a pattern can not be higher than that of its subpatterns. In their search tree, they only allow subgraphs with minimum DFS codes to be extended, to avoid redundancies in the pattern growth step. The core contribution of this paper lies in its backwards frequent substructure discovery mechanism: After a frequent pattern is found, all of its sub-patterns are extracted from this so-called *supergraph*, without any frequency computation.

2.2 Graph-Subgraph Isomorphism

Subgraph Isomorphism (SI) and Graph Isomorphism(GI) problems are closely related in that GI can be seen as a special case of SI problem. The question whether two graphs G and S are isomorphic reduces to whether they have the same number of vertices, and SI test for G and S is true. From a mathematical point of view, the complexity class of GI is unknown; however, no polynomial-time algorithms have been proposed to date. While the worst-case complexity of existing solutions to both remains to be exponential, much of the effort has been devoted to reduce the number of such cases to improve average time complexity [14, 21, 19]. It is also worth mentioning that in many real world applications, graphs in largely similar forms can be considered as isomorphic. Inexact matching algorithms can cope with slight differences across the data, making them an ideal fit when error-tolerance is of importance [22, 23, 2].

While most of the substructure discovery methods aim to reduce the number of costly subgraph isomorphism tests, reducing the cost of the test is another option worth considering. Not surprisingly, endeavors to attack this NP-Complete problem use the same ideas as those that aim to reduce the number of tests. For example, Olmos et al. [24] use a coding scheme to define a lexicographical ordering of vertices. This concept is very similar to DFS coding of gSpan[1]. Edges are assigned codes based on their labels and features of the vertices at both ends, including degree and label information. A unique DFS code is built for each subgraph, with an aim to reduce the number of operations. They compare their algorithm with *sgiso()*, the subgraph isomorphism function of SUBDUE, which itself has not been fully optimized. In many cases, their method concludes much faster than *sgiso()* function, suggesting a potential improvement to the SUBDUE system.

Nauty[19], the current state-of-the art of graph isomorphism, creates canonical labeling for graphs to find a bijection between their vertex sets. The graphs to be matched are transformed into their canonical forms. It retains completeness, i.e. is guaranteed to find the optimal and exact solution. It is inevitable that even for Nauty, there is a set of graphs that require exponential number of steps; however, it is very fast for many others of practical use. The VF2 algorithm by Cordella et al. [21] is comparable to Nauty in terms of performance. In addition to graph isomorphism, it can also perform subgraph isomorphism tests. Its authors define five feasibility rules to prune the search space, including 1-look ahead and 2-look-ahead operations. In addition to these, VF2 can also work with Attributed Relational Graphs (ARG), fully utilizing the semantic information in the attributes of the nodes. The fact that it takes continuous values in attributes into account to estimate a match cost for each node in the search space helps it to further prune nodes not having the best match cost.

A potential use of SI/GI algorithms in the world of computer vision is object recognition and localization in a scene. In [14], Abdulrahim et al. use a graph isomorphism algorithm they develop for object class recognition. Like many other SI/GI methods, they exploit neighborhood structure of the vertices to make quick rejections. Since they use node labels and their neighborhood information, they work in linear time for many non-isomorphic graph classes. For a small subset of all possible graph pairs, their algorithm can not make a decision in polynomial-time, and prompts the user to resort to an exponential matching algorithm. In their experiments, they recognize protein molecules by matching an unknown sample to a database of concepts to determine its type. The object classes are represented with graphs, and the tests show that their method works well for highly structured object types such as molecules. In this thesis, we propose a methodology that can be used for localization, not just recognition of objects. Our methodology uses a more robust SI approach, as opposed to GI, which requires objects to be isolated from background clutter so that their representations only include information from the objects themselves.

The following section is devoted to SUBDUE, a greedy graph mining tool. It forms the basis to the suggested algorithm, and the reader is advised to read the section carefully as the rest of this thesis will build on top of it.

2.3 SUBDUE

SUBDUE [2] is a data mining tool that mines frequent patterns in graphs. Designed as a greedy algorithm, it employs a computationally constrained search mechanism to enumerate and evaluate possible candidates. differs from its contemporaries in that it aims to mine patterns that best compress the dataset according to the Minimum Description Length (MDL) [25] principle. While many frequency-based graph miners tend to output too many structures that may or may not interest the user, SUBDUE detects fewer substructures which are more interesting. The search methodology in SUBDUE is a level-wise search which starts with single vertices, and expands them by one edge and possibly one vertex at each level. When expanding each substructure, only a limited number of children are generated, reducing exponential search space and therefore the computation.

2.3.1 Minimum Description Length Principle

The quality of each substructure found in SUBDUE is evaluated using MDL principle, introduced by Rissanen in 1978 [26]. MDL suggests that regularities in the data can be learned by finding out the hypotheses that lead to the best compression of the data. Rissanen has an interesting approach to the learning problem in that it neither assumes a *true* distribution underlying the data samples as in the Bayesian paradigm, nor it relies on a purely frequentist approach. It basically implements a trade-off between a model's fitting error and its complexity.

The idea behind MDL is to consider learning as finding the repetitive patterns and regularities in the data to compress it. More formally, given a set of hypotheses $\mathbf{H} = \{H_1, H_2, ..., H_n\}$ and data D, the purpose is to find an hypothesis or a family of hypotheses that compress the data most. A valid hypothesis is a part of a family of hypotheses (a model), while the set of all models is the sum of all available means to express the data. Mathematically, this two-fold MDL principle can be stated as follows:

Definition 1. *Two-Part Minimum Description Length Principle*
Let $\mathbf{M} = \mathbf{H}^1 \cup \mathbf{H}^2 \cup ... \cup \mathbf{H}^N$*, where* $\mathbf{H}^x$ *is* x^{th} *family of available hypotheses (model), and* $\mathbf{M}$ *is the union of all valid models. The best hypothesis* $H \epsilon \mathbf{M}$ *minimizes* $DL(H) + DL(D|H)$*. Here,* $DL(H)$ *is the description length of* H*, and* $DL(D|H)$ *is the description length of* D *after being compressed by* H*.*

Definition of an hypothesis in this context is no further than a mechanism to compactly represent the data. It can be a language dictionary, a codebook, a binary/decimal encoding scheme, a family of polynomials of degree k or a programming language, etc. In fact, data compression using computer languages as an encoding mechanism has long been paid special attention. In 2.3.2 the background of MDL is presented, followed by SUBDUE's interpretation of MDL principle in 2.3.3.

2.3.2 Kolmogorov Complexity

In 1965, Kolmogorov [27] formulated complexity of a given sequence of data as follows:

Definition 2. *Kolmogorov Complexity*
Kolmogorov Complexity of a sequence of data D is equal to the shortest program P that prints D and halts.

The intuition suggests that such a representation is highly dependent on the chosen computer language, which is not the case. Kolmogorov [27] has proven that given a sequence long enough, the length of two programs written in languages A and B can differ by no more than a constant c. Kolmogorov Complexity of a sequence thus refers to its regularity. This definition is in close accordance with Occam's Razor principle, which states that among competing hypotheses, the one with the fewest assumptions should be selected. In other words, the best hypothesis is also the simplest one, while the term *simple* does not imply any loss in the representation's power. Similarly, Kolmogorov Complexity (KC) refers to the simplest method able to print the data.

While any of the current programming languages such as Java, C or Python can be selected for the underlying encoder in the estimation of Kolmogorov Complexity, it is also possible to speak of Turing Machines in the same context. For example, a Turing Machine $M =< Q, \Gamma, \epsilon, \Sigma, \delta, s, F >$ where

- Q is the set of states,
- Γ is the alphabet,
- ϵ is the blank symbol,
- $\Sigma = \Gamma - \epsilon$ is the set of input symbols,
- δ is the transition function,
- s is the start state,
- F is the set of final states,

takes a string w as input to generate output string o. The string "$<M>w$", which encodes M and w in bits, is a representation for o. $KC(o)$, Kolmogorov Complexity of o, is the shortest pair "$<M>w$" such that when M is fed with input w, it outputs o. As it can be inferred, KC is invariant to the underlying encoding mechanism (programming language). However, this invariance property assumes a very long and possibly infinite data string, which is hardly the case in real situations.

Example 2.3.1. *Kolmogorov Complexity example. Let a string*

$$x = abcabcabcabc...abc$$

where x consists of abc pattern repeating for 200 times. A simple program that can output this sequence can be written in C as follows:

```
for (i=0; i<200, i++){
    printf("abc");
}
```

whereas another, longer alternative is given below:

```
printf("abcabcabcabc ... abc");
```

■

From these two code snippets in example 2.3.1, it is easy to infer that the first program is indeed the shortest one. However, Kolmogorov Complexity (KC) can not be used as a perfect MDL solution due to the two reasons below, as stated in [28]:

- For many real cases, finding the most optimal solution for KC is non-trivial. In fact, in Li and Vitanyi [29], it is shown that it is impossible to write an algorithm that returns the shortest program printing a given data sequence D. Since KC is uncomputable, it is not of practical use on real data.
- The data strings confronted in daily life are much shorter than *long enough*, so the syntax or language choice affects the results. Since certain languages are optimized toward certain paradigms, the program lengths creating the same sequence in languages A and B may vary drastically.

As opposed to the Kolmogorov Principle, Rissanen has chosen a more flexible and less idealistic approach. Even then, the Two-Part MDL fails to put forth a guideline towards design of models, or hypotheses. In 1996, Rissanen [30] re-formulated his metric to create a more concrete basis for the model selection problem, leading to a second MDL definition:

Definition 3. *One-part Minimum Description Length Principle*
Given a family of hypotheses $\mathbf{H^x}$ *and data sequence* D*, we encode* D *with a single code* $\mathbf{H^x}$*. If any parameter setting* H *in* $\mathbf{H^x}$ *leads to a small* $DL(D|H)$*, then* $DL(D|\mathbf{H^x})$ *is also small. Codes of this property are called universal codes. Among all possible such codes, the one which is minimax-optimal is chosen.* $DL(D|\mathbf{H^x})$ *is called Stochastic Complexity of* D*.*

To conclude, MDL is a principle that is different than both Bayesian and frequentist paradigms, in that it considers learning as finding the structural regularities in the data, and it employs a compression-based approach. It assumes no underlying true distribution, and implements a trade-off between over-fitting and expressional power. MDL's application to graph-based structural mining is not common, as many graph-miners tend to focus on pure frequencies [1, 12, 4]. To our knowledge, SUBDUE [2] is the only tool that uses this principle in evaluation of the substructures it encounters. In 2.3.3, we give a summary of SUBDUE's implementation of MDL.

2.3.3 Minimum Description Length in SUBDUE

SUBDUE is a greedy graph-based data mining system that works on labeled graphs [2]. It discovers frequent substructures in graphs, which are evaluated based on the MDL principle. SUBDUE has an inherent conceptual clustering mode such that it can work on the data in multiple iterations, with the graph at each iteration compressed by the best substructure found on previous iteration. With this approach, it can discover basic building blocks at first iteration, their combinations in the next, and so on. While SUBDUE discovers frequent patterns, it evaluates each pattern by estimating its value. The substructure that best compresses the input graph is the best one. Following MDL principle, value $V(S)$ of a substructure S in SUBDUE is calculated as follows:

$$V(S) = \frac{DL(G)}{DL(S) + DL(G|S)}, \tag{2.1}$$

where $DL(G)$ and $DL(S)$ are the description lengths of G and S, respectively, and $DL(G|S)$ is the description length of G after being compressed by S. In this context, compressing G with S is equal to finding all instances of S in G, and replacing each instance with a single vertex labeled as S. All egdes linking an instance's vertexes to other nodes in G originate from the replacement node in the compressed graph. This process is not recoverable, i.e. the initial state of the graph can not be obtained after the graph is processed. Thus, SUBDUE features a form of lossy compression.

In the original description, graphs and subgraphs in SUBDUE are represented with vertexes, edges and their labels. Therefore, the $DL(G)$ of a graph G equals to:

$$DL(G) = vbits + adjbits + ebits, \tag{2.2}$$

where *vbits*, *adjbits* and *ebits* are the number of bits required to represent the vertexes, adjacency information, and edges of G, respectively. SUBDUE uses an adjacency matrix representation to encode the edge information. The further details of this description length definition is given in [31].

SUBDUE's main algorithm is a computationally constrained beam search that looks for repetitive patterns in the input graph(s). Beam search is a restricted breadth-first search algorithm in which only a number of promising paths are kept at the frontier of the search [32]. As opposed to other graph miners which return all frequent subgraphs, SUBDUE's aim is to find the *interesting* ones. This seemingly subjective evaluation is accomplished through MDL. The *best* pattern becomes the one which compresses the input graph(s) most. A detailed overview of the search procedure is given in chapter 3.3.

2.3.4 Applications of SUBDUE

As an efficient graph mining tool with open-source code available, SUBDUE has long been used in many applications. Because the input of SUBDUE is a simple and convenient graph representation, it can be used with any structured data that can be represented with directed/undirected labeled graphs. As a result, SUBDUE has been successfully applied to many different domains including MRI brain scan classification [33], anomaly detection [34], telecommunications [35], earthquake activity analysis [36], insider threat discovery [37], molecular fragment mining [38] and web search engines [39]. One of the main advantages of SUBDUE over its alternatives is its meaningful output. While many regularity mining methods such as gSpan [1] or FSG [10] make no clear distinctions on the final substructure list, SUBDUE effectively ranks them and has much fewer output patterns.

In [40], Eberle and Holder experimented with SUBDUE and GASTON [4] in a task that requires each to output the best pattern found in the input graph database. Since GASTON does not evaluate the patterns, they modify the algorithm so that it estimates a score for each final substructure. Given a graph G and frequent subgraph S, $Score_S = Frequency_S * Size_S$. While this metric is different from MDL, it gives exact same results in their synthetic databases, which are generated intentionally to contain a single maximal substructure. SUBDUE's greedy yet efficient approach helps it outperform GASTON in terms of run-time for this specific configuration. Ketkar et al. compared SUBDUE with two of the leading graph miners, FSG [10] and gSpan[1] in [41]. The synthetic and real datasets they used in performance analysis are graph transaction databases, as FSG and gSpan expect such inputs, while SUBDUE can additionally work with a single large graph. In this work, a clearly non-linear behaviour of SUBDUE as the size of database increases is verified, in contrast to the other two. This degradation is mostly caused by SUBDUE's inefficient subgraph isomorphism testing, while both gSpan and FSG use efficient mechanisms such as canonical labels to reduce the number of exponential SI checks. However, SUBDUE manages to find the embedded pattern in the graphs with very high precision by reporting it as the best substructure %80 of the time. Increasing the support threshold for gSpan and FSG causes the number of output patterns to be reduced; however, at the cost of the embedded pattern being not discovered at all.

In addition to frequent subgraph discovery, SUBDUE can be used in conceptually different tasks. Jonyer et al. utilize SUBDUE's iterative nature en route to hierarchical conceptual clustering [31]. They modify SUBDUE to add it hierarchical clustering capabilities and fix a phenomenon called false compression that may occur in rare situations. This case arises due to the MDL metric for the exact reasons we discussed in 2.3.2. While default MDL in SUBDUE uses row-wise encoding of its adjacency matrices, the results may differ for substructures of similar value when column-wise encoding is used. This is a direct result of working with limited amount of data, in which case choosing one underlying representation over another affects MDL criterion. They fix this problem by making use of row-wise and column-wise encodings together to get a unbiased mechanism. In their experiments, they use their method to obtain so-called classification trees, which show the hierarchy of concepts existing in the dataset. In DNA analysis, this structure is equivalent to a hierarchy of compounds, with more complex structures emerging at upper layers. For a set of animal descriptions, a taxonomy tree which explains relations of animal species is obtained.

An interesting work is presented by Jonyer et al. in [42], in which SUBDUE is used to learn graph grammars. Graph grammars are in very similar to textual grammars, however, they can model more complex relationships than those existing in text-based ones. While the latter can only apply concatenation operation to model the variables' interaction with each other, graph grammars can use arbitrary graphical structures for the same task. For example, a context-free grammar of the form $S \rightarrow Abc, A \rightarrow a|\emptyset$ can be transformed into a graph grammar with the addition of random links between right-hand side elements. For illustration, while a textual version of the first rule $S \rightarrow Abc$ connects (A, b) and (b, c), its graph-grammar version may embed any subset of (A, b), (A, c) and (b, c). If the edges are directed, (b, A), (c, A) and (c, b) should be added to this list. While learning graph grammars in SUBDUE is a conceptually different task, the implementation details remain unchanged in [42]. This stems from the fact that they limit the grammars being learned to context-free ones, which are directly applicable on SUBDUE. Their method learns graph grammars by iteratively compressing the input graph with the best substructure at each level. Each compression becomes a rule of the grammar, i.e. a new substructure S consisting of three nodes x, Y, z will form the rule $S \rightarrow xYz$. In this example, Y is a previously discovered substructure which yielded an earlier rule. After rule generation, the graph is compressed with S and the next iteration is processed. Since compression consists of replacing a number of nodes with a newly generated single one, the process is inherently context-free, always resulting in a single non-terminal on left-hand side. Their method is able to learn recursive rules of the form $S \rightarrow aS|\emptyset$, since SUBDUE is capable of mining recursive subgraphs.

In many graph mining algorithms such as gSpan [1], FSG [10], SeuS [8] and GASTON [4], the optimizations toward reducing number of exponential isomorphism tests require the patterns embedded in the database to be exactly the same. Since they do not perform the test when there is a mis-match in terms of nodes, edges or their labels, many of the patterns that occur in the database in slightly different forms are missed. To be able to find these patterns, an inexact matching algorithm [22] is required for SI tests. SUBDUE is one of the rare graph mining algorithms that successfully implement this pattern. In addition to the exact matching of subgraphs, SUBDUE has a mode in which the user can set a threshold for similarity of graph patterns to be counted as the same. So, regularities that exist throughout the database with small variations are efficiently discovered by SUBDUE. Possible distortions include vertex/edge deletion and substitution of their labels. When two graphs are matched, the total cost is calculated based on the overlapping ratios that exactly match, penalizing each variation with a constant factor. If the total cost is lower than a user-specified threshold, a match is reported.

2.4 Mining Graphs with Numeric/Vectorial Attributes

In many graph mining algorithms [1, 10, 8, 4, 2, 15], labels of the vertices and edges are considered as concepts or alphanumeric strings. Counter-intuitively, numeric values embedded in these programs will be regarded as strings, and therefore their matching is only based on string comparison. To give an example, three nodes labeled as 1.1, 1.2 and 10.8 will be regarded as equally distant from each other. However, processing 1.1 as being much closer to 1.2 than 10.8 is a more intelligent way of matching. To cope with this lack of proper handling mechanisms, continuous values in graphs can be discretized into bins, and the bin information can be encoded in the graph as cluster labels. While such discretization steps are useful, we propose a more native methodology that can incorporate numeric and vectorial attributes in graph mining process. We skip the discretization step by letting the labels have multi-dimensional data attributes, which are utilized in the label matching procedure. While this considerably slows the discovery process, we show that the information gain is worth the increased complexity. Before we move on to chapter 3 for the details of our novel method, below we mention notable works aiming to solve this problem.

In the context of this thesis, we refer to single datum values as *numeric* attributes. Examples of numeric attributes can be temperature of a room (25 °C), age of a person (42), population of a city (1.350.000) etc. Data sequences, on the other hand, are identified as *vectorial* attributes. For instance, the binary representation of the number 19 (10011) can be considered as a data vector of length 5. The RGB (Red-Green-Blue) color space can be encoded by 3 single-byte numbers (each in range 0 .. 255) or 24 bits (each either 0 or 1). Alternatively, a sequence of n measurements by a sensor can be represented with a data vector of length n. As exemplified here, many meaningful data values in either discrete or continuous space can be encoded into vectors and numeric attributes. To use these type of data in graph-mining tools, one needs to put these values into clusters or bins, and replace the original attribute with its grouping information. To our knowledge, our method is the first to directly use the *data* in its discovery step.

SUBDUE uses a form of inexact matching proposed by Bunke in 1983 [43]. While this scheme can accommodate certain changes such as node/edge deletion and substitution, we think a major improvement can be achieved by handling of numeric/vectorial attributes. Real-life graphs often include continuous attributes, which are either discretized in pre-processing [44], or ignored completely [5] by graph mining algorithms.

Because of its inexact-matching approach that already offers error-tolerance, there have been multiple efforts to extend SUBDUE so that it can work with numeric attributes in a special manner [35, 45, 46, 47, 48]. An early example of this methodology is implemented by Baritchi et al. in [35] in 2000. They propose three different ways of matching numeric labels:

- Exact Match: Label l_i matches label l_j, if and only if $l_i = l_j$, which is the default behaviour of SUBDUE,
- Tolerance Match: Label l_i matches label l_j, if and only if $||l_i - l_j|| < t$, where t is an user-defined threshold,
- Probabilistic Match: $MatchCost(l_i, l_j)$ is defined as the probability of l_j being drawn from a probability distribution with mean l_i and user-defined standard deviation.

The three proposed label matching types enable SUBDUE to consider numeric labels as points in 1D space and match them based on their distance from each other. The parameters defined in them are controlled by the user. To generate final substructure definitions after the discovery process, they perform a post-processing operation to print out a detailed information of the best subgraphs.

A later variant of SUBDUE with such capabilities has been proposed by Romero et al. in 2010 [45, 46]. Their method, like [35], is limited to one-dimensional attributes such as a single number. They perform a data-range generating process based on the input data, effectively partitioning the given attribute values into clusters. These cluster numbers are then assigned to the attribute nodes as conceptual labels. The knowledge discovery phase runs over the quantized graph. Similarly, Davis et al. [48] present their own implementation of the SUBDUE system, while introducing an outlier-detection based quantization step. Their method basically detects the outliers in each numeric attribute and labels them as anomalies. Thus, for each attribute, they assign samples regular labels such as *Normal* or *Abnormal*. Interestingly, they focus on only the detection of subgraphs having vertexes with *Abnormal* attributes. Although they are lower on count, such information is valuable in security-based applications, where abnormal patterns are generally considered as suspicious. In 2012, Davis et al. developed a more generalized approach towards handling of numeric attributes in [47]. Following a similar pre-processing step to [48] to detect abnormal attributes, they prune the graph by removing vertexes having abnormal values. The edges connecting to these vertices from others are also removed. As a result, they perform %30 less isomorphism tests when compared to unconstrained search.

CHAPTER 3

A GRAPH MINING TOOL: EXTSUBDUE

In this chapter, first, the SUBDUE algorithm that forms the basis of this thesis is presented. We extend SUBDUE by adding it numeric and vector data label handling capabilities. We introduce the notation and SUBDUE's input format that will be used through this work in section 3.1 and 3.2, respectively, as well as the formal definition of our problem. In section 3.3, we give a detailed overview of SUBDUE, and discuss about its deficiencies as well as our contributions that lead to ExtSUBDUE in section 3.4.

3.1 Terminology and Notation

Graphs are versatile representations of structured data, in that they can model very complex relationships with simple elements such as vertexes and edges. A graph G can be represented with a triple $G = (V, E, \lambda)$, where V is the set of vertices, E is the set of edges, and λ is the labeling function that assigns a node or edge its label. A graph S is called a subgraph of G, if $\forall v_i \in S, v_i \in G$. S is *connected* if there is a *path* from each vertex v_i to each vertex v_j in S. Although it is possible and certainly useful to define unconnected subgraphs, we assume that each subgraph is connected. An edge $e = (v_i, v_j, type)$, s.t. $v_i, v_j \in G$ should have either *directed* or *undirected* in its *type* field. Directed edges have the property $(v_i, v_j, directed) \neq (v_j, v_i, directed)$, whereas $(v_i, v_j, undirected) = (v_j, v_i, undirected)$.

Conceptually, nodes (vertexes) in a graph usually map to objects or entities, while edges are used to define structural relations between these entities. A simple graph is given in figure 3.1a. This example features a unconnected graph which represents persons in a database. Here, the center nodes are labeled with the names of the people, which connect to attribute nodes using directed links. Even though unstructured datasets can be modeled with such star-shaped graphs, the knowledge to be learned from them is rather limited. More structured data can also be embedded in graphs, exemplified with a very simple social network in figure 3.1b. In this graph, the vertexes correspond to the people, and unlabeled edges link friends. The reader should be aware that the labeling function λ indeed allows unlabeled edges/nodes. Even in such graphs, structured patterns such as cliques or paths can be discovered by an unsupervised algorithm.

Many graph mining algorithms including gSpan [1], FSG [5] and GASTON [4] aim to find all subgraphs having a certain support in the input graphs. SUBDUE, on the other hand, returns much fewer substructures that compress the dataset most. Formally speaking, given a graph dataset $DS = G_1, G_2, ..., G_n$, we aim to find the pattern set $S = s_1, s_2, ..., s_n$, such that $\forall s_i \in S, V(s_j) > V(s_i) \Rightarrow s_j \in S$, with value function $V(s)$ in 2.1. Each substructure s_i is evaluated separately on DS, so compression value of one element does not affect another. A detailed explanation of $V(s)$ is given in section 2.3.3.

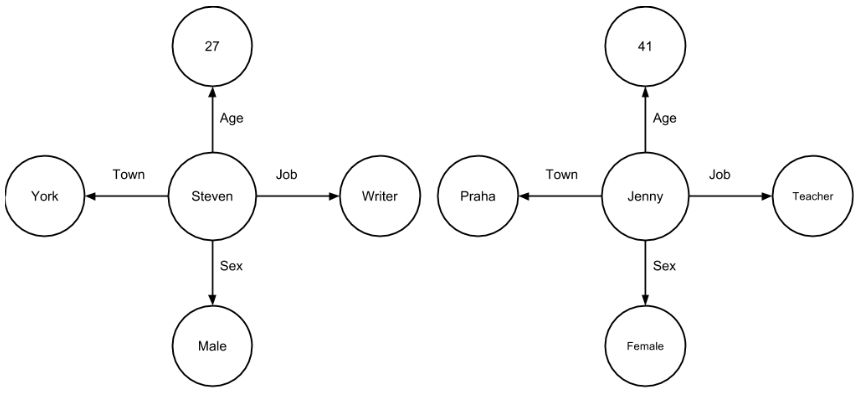
An inexact matching algorithm similar to [43] is center to SUBDUE's search procedure. In exact matching, two subgraphs s_i and s_j match exactly if and only if $GraphMatch(s_i, s_j) = true$. $GraphMatch()$ essentially checks whether $size(s_i) = size(s_j)$ and subgraph isomorphism test between s_i and s_j is true. In SUBDUE, the interpretation of $GraphMatch(s_i, s_j)$ slightly differs:

$$GraphMatch(s_i, s_j) = InexactGraphMatch(s_i, s_j, threshold), \tag{3.1}$$

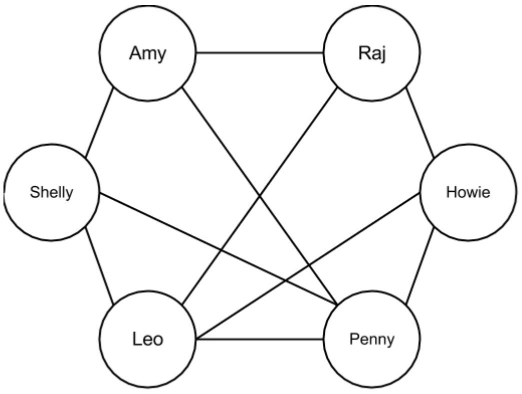
where $InexactGraphMatch(s_i, s_j, 0)$ implements exact matching. On the other hand, *threshold* parameter allows the user to match two graphs s_i and s_j if the cost of transforming s_i to s_j does not exceed a user-specified positive value. The operations allowed in this so-called transformation are vertex and edge deletion/addition, label changing on a vertex or edge, and reversing the direction of an edge. Each modification is penalized with a constant cost of 1.0. The purpose of $InexactGraphMatch()$ is to determine an optimal mapping between the two input graphs' vertices and edges. To avoid exponential exhaustion of the search space, the function switches to greedy search to determine an albeit sub-optimal mapping if the global optimum is not reached within a specified number of steps.

3.2 Input Format and Options in SUBDUE

Input Graph(s): SUBDUE can take a number of positive and negative graphs as input, each prefaced by either *XP* or *XN*, respectively. Positive and negative graphs only differ in their interpretations: SUBDUE, by default, discovers subgraphs that compress positive graphs most, and negative graphs least. This property makes it a feasible tool for supervised learning. In case a single graph is provided, the *XP* prefix can be omitted. Each graph consists of a set of vertices and edges, along with their labels. Below, we give definitions of vertexes and edges in input files. For example, a new vertex can be introduced with the following:



(a) Graph of an unstructured database



(b) A very simple social network

Figure 3.1: Graph examples.

```
v <#> <label>
```

Here, <#> is the identifier of this vertex, and <**label**> is its label. Vertex identifiers introduced within a graph should start at 1, and they must increment by 1 for each new vertex. Edges, on the other hand, embed both vertex IDs at each end, direction if any, and a label. There are three ways to define an edge in a graph:

```
e <vertex #1> <vertex #2> <label>
u <vertex #1> <vertex #2> <label>
d <vertex #1> <vertex #2> <label>
```

In this scheme, <**vertex#1**> is the ID of the source, and <**vertex#2**> is the ID of the destination vertex. **d** is used to define a directed edge, **u** symbolizes an undirected edge, and the interpretation of **e** depends on the *-undirected* option specified at the command line. Only if the *-undirected* flag is set, edges defined by **e** are assumed undirected, otherwise they are considered directed. The pre-defined substructures are given as separate input graphs to SUBDUE, each prefaced by a *PS* keyword.

Each <**label**> is given as a text sequence, regardless of its type. In SUBDUE, two types of labels are available: *numeric* and *string*. A *numeric* label consists of encoding of a single decimal number such as 15 or 23, whereas a string label is any sequence that does not correspond to a number as a whole(e.g. *label*, *cat*, 12*Racing*, ...). In addition to these, we add *vector* labels, that consists of a *string* label and the following <**data**> field. The <**data**> is the attribute of the node or the edge it belongs to, which turns the input graph into an attributed relational graph (ARG). This label type is explained in detail in section 3.4.

A simple example graph and pre-defined substructure is given below. Comments at each line start with a % symbol.

graph.g including input graph:

```
XP % Can be omitted since only one graph is provided
v 1 person % vertex with string label
v 2 18     % vertex with numeric label
v 3 person
v 4 22
v 5 person
v 6 18
v 7 money 1:1:3200 % vertex with vector label
v 8 money 1:1:2800
u 1 2 is   % edge with string label
u 3 4 is
u 5 6 is
d 5 7 has
d 1 8 has
```

ps-graph.g including a pre-defined substructure:

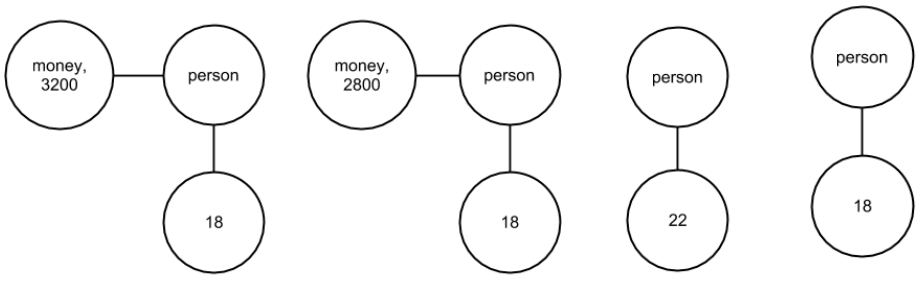
```
PS %Pre-defined sub prefix
v 1 person
v 2 18
u 1 2 is
```

Visualizations of *graph.g* and *ps-graph.g* are shown in figure 3.2. The pre-defined substructure has 2 instances in the input graph.

Command-line Options of SUBDUE: SUBDUE has a number of command line options that determine the input/output operations, complexity of the search process and more. Here, we give a summary of these options.

To start with, SUBDUE executable can be called with this line on a Linux machine:

```
subdue <options> <graph input file>
```



(a) Input graph (b) Pre-defined substructure

Figure 3.2: Visualization of sample graph and substructure.

where <**options**> consists of specified command line parameters, and <**graph input file**> is the file that includes input graph(s) (e.g. *example.g*). Some key options are given below, while the curious reader is advised to read through the SUBDUE Manual [49] for a full list:

- *-beam* <#>: The *beam* length of SUBDUE's search queue. Formal definition of this parameter is explained in section 3.3.
- *-compress*: If this option specified, the compressed input graph at the end of the iteration is written to a file.
- *-eval* <#>: Setting this parameter to 1 results in using MDL [25] in evaluation of candidate substructures. If *eval* is set to 2, a size-based heuristic is used, while a value of 3 different evaluation function called *Set Cover*.
- *-inc*: If *-inc* option is specified, data is handled incrementally, rather than processing a single input file. If the input graph is too big to fit into memory, it can be divided into multiple files, enabling SUBDUE to work within low-memory configurations.
- *-limit* <#>: Upper limit of number of patterns to be considered for extension at each iteration.
- *-iterations* <#>: If the argument is larger than 1, SUBDUE is run in multiple iterations. At the end of each iteration, the best substructure discovered is used to compress the input graph G, and the compressed graph becomes the input of the next iteration.
- *-maxsize* <#>: Maximum number of vertexes allowed in final substructures.
- *-minsize* <#>: Minimum number of vertexes allowed in final substructures.
- *-nsubs* <#>: The number of best substructures in the final list.
- *-overlap*: If *overlap* flag is specified, overlaps among patterns' instances are allowed.

- *-threshold <#>*: Tolerance level with respect to a substructure's size in graph matching step. It basically defines the maximum allowed ratio of structural discrepancy between a pattern and its instances.

3.3 SUBDUE: A Substructure Discovery Method

SUBDUE is a greedy graph mining tool that discovers structural regularities in graphs. It uses a variant of beam search [50] to limit the number of extended substructures. The frequent pattern search starts with single vertexes. In the second level, these single-vertex substructures are extended in all possible ways, based on the actual examples. After all extended patterns are generated, top *beam* subgraphs are selected to be parents in the next level. Search operation continues until no new substructures are generated, or a user-specified time/extension limit is exceeded. SUBDUE uses MDL [26] heuristic to evaluate and rank the patterns. MDL principle suggests that the subgraphs that compress the dataset most are indeed the best ones. SUBDUE's pattern discovery process is given in algorithm 1, adapted from [41].

Initial compression: Before SUBDUE begins its *unsupervised* discovery, i.e. finding frequent substructures in the input graph G, it can optionally compress G with a set of pre-defined substructures (Line 3, algorithm 1). Examples of such subgraphs can be a carbon ring in molecular graphs as given in figure 3.3, or any other substructure that is *known* to span the given data. Embedding domain information with pre-defined substructures helps SUBDUE to focus on more complex patterns. If such domain information is not available, SUBDUE can also be run in multiple iterations. In the first iteration, it can discover the basic units that cover the data. In the second iteration, interactions between these units and the rest of the nodes in G are modeled. This idea is implemented in MoFa [5], where an initial run over the data with a very high support threshold is used to find out the underlying simple patterns. These basic subgraphs, not surprisingly, correspond to frequent structures such as carbon rings.

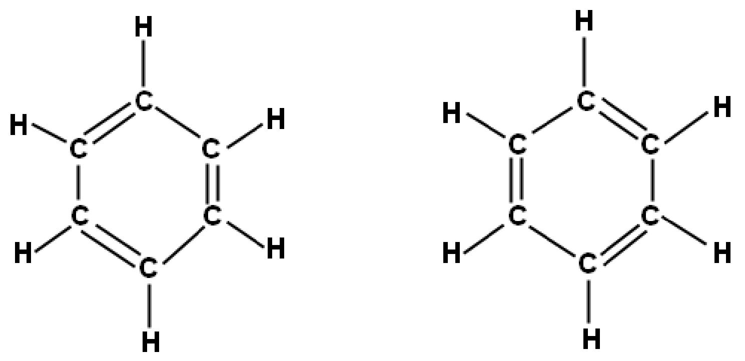


Figure 3.3: Carbon ring in molecules.

As opposed to SUBDUE's efficient beam search strategy,

Algorithm 1: Discovery Procedure of SUBDUE.

```
   1
   Input: Input graph $G$, $beam$, $maxBest$, $maxSubSize$, $limit$
   Output: $bestSubList$
1  $parentList$ = null; $childList$ = null; $bestSubList$ = null; $subCounter = 0$;
2  // If there are pre-defined subs, compress with them
3  $CompressWithPreDefinedSubs(G)$;
4  // Initialize the parent list with single vertex subs
5  parentList = $FindSingleVertexSubstructures(G)$;
6  while $subCounter \leq limit$ and $parentList$ not empty do
7      while $parentList$ not empty do
8          $parent = RemoveHead(parentList)$;
9          Extend $parent$ in all possible ways based on its instances;
10         Group extended subs into $currentChildList$;
11         foreach $child$ in $currentChildList$ do
12             if $size(child) \leq maxSubSize$ then
13                 $childValue = EvaluateSub(child, G)$;
14                 Insert $child$ to $childList$ ordered by $childValue$;
15                 if $length(childList) > beam$ then
16                     Trim $childList$ to $beam$ different valued elements;
17         $subCounter{+}{+}$;
18         Insert $parent$ to $bestSubList$ in order by value;
19         if $length(bestSubList) > maxBest$ then
20             Trim $bestSubList$ to $maxBest$ elements;
21     Switch $parentList$ and $childList$;
```

$CompressWithPreDefinedSubs()$ function performs an exact subgraph isomorphism test. This design choice effects the run-time performance of early compression with domain information. As the size of pre-defined substructures increases, the performance drops rapidly. Thus, this function should be used to a limited extent. Another notable feature with $CompressWithPreDefinedSubs()$ is that, given multiple basic patterns to build on, the graph is compressed with respect to the order of the initial patterns. However, it precisely follows the inexact matching scheme if needed, which is one of SUBDUE's strengths. Therefore, the domain knowledge injected into SUBDUE can be designed according to these tips:

- If the pre-defined substructures are subgraphs of each other, their order affects the compressed graph G'.
- If the matching tolerance is set too high, pre-defined patterns that are relatively similar, match each other's instances. In this case, their order also affects the final graph.

These tips are intended for guidance rather than restriction, as such behavior can as well be desired. In the final graph, each instance of pre-defined substructures is replaced by a single vertex. This vertex is labeled with the identifier of the pattern it refers to. The discovery process is performed after this initial compression, if any.

Subgraph extension: In algorithm 1, G is the input graph(s), *beam* is beam length of the search, *maxBest* is the maximum number of best substructures to be reported, *maxSubSize* is the maximum number of vertexes to be allowed for output patterns. The operation that initializes the current substructure list *parentList* is $FindSingleVertexSubstructures()$ function. *parentList* initially contains each vertex v, whose label $L(v)$ is encountered at least *twice* in G. The main loop starting at line 6 continues until the user-specified *limit* number of substructures are considered for extension, or the list of extensible subgraphs is empty.

Each substructure in *parentList* is expanded in all possible directions in algorithm 1 (lines 7-10). The resulting instances are grouped such that isomorphic embeddings are represented with a single pattern. Each augmented substructure is evaluated to determine its compression ratio, in lines 11-16. $EvaluateSub(child, G)$ function compresses G with *child* based on MDL principle, and sets value $V(child)$ as explained in 2.1. An overview of the Description Length (DL) used in MDL is given in 2.3.3. After evaluation, *child* is placed in *childList* which keeps track of all new substructures at the current iteration. As opposed to unconstrained search, the length of *childList* is restricted by the *beam* parameter.

Value-based queue: In early versions of SUBDUE, the actual length of *childList* is limited, rather than the number of different-valued substructures in it. This deficiency was later fixed on newer versions of SUBDUE, using a value-based restriction. The effect of limiting the length rather than number of values can be seen in example 3.3.1:

Example 3.3.1. *Limited queue example in SUBDUE.*
Consider a fully-connected pattern S having 8 vertexes embedded in graph G. Since SUBDUE iteratively discovers the substructures in graphs, each subgraph of S is also a frequent pattern. For example, if each vertex of S has a unique label, S has $\binom{8}{3} = 56$ *3-vertex subgraphs. If these subgraphs do not exist throughout G in anywhere other than instances of S, each of these subgraphs compress G with the same ratio. Therefore, the value of each sub-pattern is actually the same. If the beam parameter is set to 4, childList can only have 4 of these 56 subgraphs, although all of them have the same value. To cope with these cases, a value-based queue is implemented in SUBDUE, allowing childList to have no more than beam different-valued substructures. With this approach, any number of subgraphs having the same compression ratio claim only one spot in the queue. Current implementation of SUBDUE uses a value-based queue to implement childList, rather than a size-limited earlier version. As stated in [31], this method also has its shortcomings in that the number of substructures in the queue tends to grow exponentially. In [31], look-ahead functions are utilized to prune patterns whose extended children overlap, hinting they are part of the same large substructure.*
■

After all possible extensions of a parent subgraph are considered, they are put into the value-ordered *bestSubList* that includes the *best, highest-valued* substructures encountered so far. The subgraphs in *childList* become the parent substructures to be extended in the next turn. The process continues until a user-specified limit is reached, or no further extension is possible. After the process is finished, top *maxBest* patterns in *bestSubList* are reported.

3.4 ExtSUBDUE: Addressed Problems and Our Contributions

One of the main problems in SUBDUE is its incapability to work with numeric and continuous attributes. In its default version, SUBDUE considers each label as a unique identifier. For example, the two subgraphs in figure 3.4 are considered as non-isomorphic. In fact, SUBDUE estimates that the dissimilarity between these two graphs is indeed 0.5, considering 0 as isomorphic, and 1 as most dissimilar. Transforming the first graph to the second involves three label substitution operations, each of which brings a fixed penalty of 1.0. The calculation of the dissimilarity measure used in SUBDUE, $MatchCost(g_1, g_2)$, is given in equation 3.2. The operations in this transformation scheme are symmetric, since addition/deletion of a vertex or edge, substitution of a label and edge direction changing are all penalized with a cost of 1.0. Therefore, $MatchCost(g_1, g_2) = MatchCost(g_2, g_1)$. $MatchCost(g_1, g_2)$ of the two graphs g_1, g_2 in figure 3.4 is $\frac{3}{6} = 0.5$. g_1 can be transformed to g_2 in three steps, by changing the labels of the three vertexes to their counterparts in g_2. Both graphs have the same size of 6 with 3 vertices and 3 labels. The fact that these two graphs are intuitively *almost* isomorphic suggests improvement to SUBDUE's rigid matching function. Below, we give a list of our contributions to SUBDUE.

$$MatchCost(g_1, g_2) = \frac{\text{cost of transforming } g_1 \text{ to } g_2}{max(size(g_1), size(g_2))},$$
$$size(g) = \text{Number of vertices in } g + \text{Number of edges in } g. \tag{3.2}$$

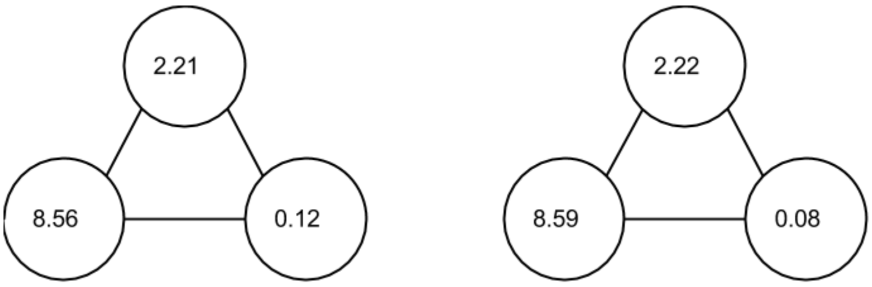


Figure 3.4: Non-isomorphic graphs.

1. **Elastic Matching of Labels:** Our first and most critical contribution to SUBDUE is in its label matching function, called *LabelMatchFactor*(l_1, l_2), given in 3.3.

$$LabelMatchFactor(l_1, l_2) = \begin{cases} 1, \text{if } l_1.labelType \neq l_2.labelType \\ 0, \ \text{if } l_1 = l_2 \\ \quad \text{for numeric/string labels } l_1, l_2 \end{cases} \tag{3.3}$$

where l_1, l_2 are the labels to be compared. Each label has a *labelType* which defines its label type (*numeric*, *string*, *vector*).

By default, SUBDUE considers two types of labels: *numeric* and *string*. However, SUBDUE's label matching for these two types are intrinsically the same: it checks for exact matching in both. This approach results in a certain decrease in terms of abstraction and expressive power: For example, consider the sample comparisons below:

Example 3.4.1. *The arguments in the examples below are denoted with their values for easy reading. You may assume that each number corresponds to a numeric label, while others are string labels.*

- $LabelMatchFactor(1.0, 1.0) = 0;$
- $LabelMatchFactor(1.001, 1.0) = 1;$
- $LabelMatchFactor(1.001, 989) = 1;$
- $LabelMatchFactor(2.7, AStringLabel) = 1;$

- $LabelMatchFactor(AStringLabel, YetAnotherStringLabel) = 1$;
- $LabelMatchFactor(AStringLabel, AStringLabel) = 0$.■

As it can be seen from these examples, $LabelMatchFactor(l_1, l_2)$ is used to match two labels in SUBDUE's discovery process. In other words, it defines the cost of label substitution operation. l_1 and l_2 can belong to either nodes or edges in the input graph G. By default, $LabelMatchFactor()$ is $LxL \rightarrow \{0, 1\}$, where L is the set of available labels, and does not allow any elasticity in terms of matching labels. We attempt to fix this deficiency by modifying $LabelMatchFactor()$ so that it is $LxL \rightarrow [0, 1]$. We add a third label type to L, named as *vector* label. This new label type consists of a string label for conceptual matching, as well as a data vector of any length as its attribute. Its format is "$name$ $distFunc : length : d_1 : d_2 : ...d_{length}$". Each *vector* label has a *name* field that defines what it is, such as *temperature* or *color*. If no such distinction is needed to represent the input data, it can be filled with a uniform name such as *label* throughout the dataset. Following SUBDUE's default label matching procedure, two *vector* labels whose string parts are not strictly same result in a matching cost of 1. The rest of a vector label is its data part. This string consists of a header including two parameters, and the subsequent data vector. The first parameter, *distFunc*, defines the distance function type to be used with these types of labels. Implemented distance functions are:

- City Block Distance in equation 3.6, with identifiers CB and 1,
- Euclidean Distance in equation 3.7, with identifiers EU and 2,
- Cosine Distance in equation 3.8, with identifiers CS and 3,
- Correlation Distance in equation 3.9, with identifiers CR and 4,
- Hamming Distance in equation 3.10, with identifiers HM and 5.

The *length* parameter, on the other hand, equals to the number of elements in the following data vector. The subsequent data entries d_1 .. d_{length} consist of *length* double-precision numbers. Some *vector* label examples are given below:

- $position\ 2 : 2 : 35.2928 : -87.108$
- $temperature\ 1 : 1 : 26$
- $color\ 2 : 3 : 144 : 56 : 215$
- $data\ 1 : 10 : 0.9 : 0.12 : 0.54 : 0.76 : -0.19 : 0 : 0 : 0.98 : -1.5 : 1$

With the addition of *vector* labels, $LabelMatchFactor()$ function is re-formulated into $ExtLabelMatchFactor()$ to as seen in equation 3.4. Single numeric data values can be written as a data vector of $length = 1$, $distFunc = 1$.

$$ExtLabelMatchFactor(l_1, l_2) = \begin{cases} 1, \text{if } l_1.labelType \neq l_2.labelType \\ 0, \text{ if } l_1 = l_2 \\ \quad \text{for } numeric/string \text{ labels } l_1, l_2 \\ VectorMatch(l_1, l_2) \\ \quad \text{for vector labels } l_1, l_2 \\ 1 \text{ otherwise.} \end{cases} \tag{3.4}$$

Where

$$VectorMatch(l_1, l_2) = \begin{cases} 1, \text{ if } l_1.name \neq l_2.name, \\ 1, \text{ if } l_1.distFunc \neq l_2.distFunc, \\ 1, \text{ if } l_1.length \neq l_2.length \\ VectorDistance(l_1.data, l_2.data, l_1.name, \\ \quad l_1.length, l_1.distFunc). \end{cases} \tag{3.5}$$

With the extended $ExtLabelMatchFactor()$ definition in equation 3.4, *vector* labels are treated in a different way than *numeric* or *string* labels. The elastic matching of *vector* labels is implemented in $VectorMatch(l_1, l_2)$. Similarly with $LabelMatchFactor()$, $ExtLabelMatchFactor()$ returns a maximum cost of 1 if the two input labels are of different types. For *numeric* and *string* types, it performs an exact comparison. For two *vector* labels l_1 and l_2, $VectorMatch()$ is called to get their distance in terms of pair-wise distance function specified by $distFunc$ parameter.

$VectorMatch(l_1, l_2)$ function in equation 3.5 first checks for header fields to see if l_1 and l_2 have the same *name*, use the same distance function ($distFunc$) and have the same *length*. If one of these checks fail, the maximum matching cost is returned. However, if they are fully compatible, the distance between their data sequences is returned.

The distance functions implemented in this study are:

(a) City Block Distance :

$$VectorDistance(\mathbf{p}, \mathbf{q}, name, l, CB) = \frac{\sum_{i=1}^{l} |p_i - q_i|}{maxDistance_{name,l,CB}}. \tag{3.6}$$

Where **p** and **q** are data sequences to be compared, *name* is the *string* label and l is the length of each sequence.

(b) Euclidean Distance:

$$VectorDistance(\mathbf{p}, \mathbf{q}, name, l, EU) = \frac{\sum_{i=1}^{l}(p_i - q_i)^2}{maxDistance_{name,l,EU}}. \tag{3.7}$$

(c) Cosine Distance, 1 - (cosine of the angle between **p** and **q** as vectors in l-dimensional space):

$$VectorDistance(\mathbf{p}, \mathbf{q}, name, l, CS) = 1 - \frac{\mathbf{pq}'}{\sqrt{(\mathbf{pp}')(\mathbf{qq}')}}. \tag{3.8}$$

(d) Correlation Distance, 1 - (sample correlation between data sequences **p** and **q**):

$$\begin{aligned} VectorDistance(\mathbf{p}, \mathbf{q}, name, l, CR) &= 1 - \frac{(\mathbf{p} - \bar{\mathbf{p}})(\mathbf{q} - \bar{\mathbf{q}})'}{||(\mathbf{p} - \bar{\mathbf{p}})||.||(\mathbf{q} - \bar{\mathbf{q}})||}. \\ \text{where } \bar{\mathbf{p}} = \frac{1}{l}\sum_{j=1}^{l} p_j, \bar{\mathbf{q}} &= \frac{1}{l}\sum_{j=1}^{l} q_j \end{aligned} \tag{3.9}$$

(e) Hamming Distance, percentage of positions having different values in **p** and **q**:

$$VectorDistance(\mathbf{p}, \mathbf{q}, name, l, HM) = \frac{\#(p_j \neq q_j)}{l}. \tag{3.10}$$

The newly added $VectorDistance(\mathbf{p}, \mathbf{q}, name, l, distFunc)$ considers **p** and **q** as points in multi-dimensional space for City Block and Euclidean distances, vectors in Cosine distance, and data sequences in Correlation and Hamming distance functions. While three of the functions (3.8, 3.9, 3.10) result in a distance in the range of $[1, 2]$, City Block Distance 3.6 (1,CS) and Euclidean Distance 3.7 (2,EU) functions need to be normalized. For this purpose, we normalize them by dividing them with a constant value called $maxDistance_{name,l,i}$ where $i \in \{1, 2\}$ that is calculated from the set of labels L in input graph G. Calculation of $maxDistance_{name,length,distFunc}$ involves going over the label list L twice to find the maximum unnormalized distance between each compatible label pair $l_1, l_2 \in L$.

$$maxDistance_{name,l,c} = \max_{l_1,l_2 \text{ in } L} LearnDistance(l_1, l_2, name, l, c).$$

where

$$LearnDistance(l_1, l_2, name, l, c) = \begin{cases} \sum_{i=1}^{l} |l_1.data[i] - l_2.data[i]|^c \text{ if} \\ \quad (l_1.name, l_2.name = name, \\ \quad l_1.distFunc, l_2.distFunc = c, \\ \quad l_1.length = l_2.length = l), \\ 0 \text{ otherwise.} \end{cases} \tag{3.11}$$

The value of $maxDistance_{name,l,c}$ is calculated for each possible triplet $(name, l, c)$ where $c \in \{1, 2\}$ existing in the dataset. Therefore, we ensure that each vector label type can be normalized. Using this approach, different features embedded in the same graph with different headers $(name, length, distFunc)$ will have their own distance functions for City Block and Euclidean options. This essentially increases the power of this representation: information from multiple domains can be incorporated within the same graph, using the same vector space and dimension. ExtSUBDUE can discover patterns which is a combination of various sources of information. Another workaround for this specific problem is to assume that all embedded *vector* labels (features) are normalized so that for $l_i = name\ distFunc : l : d_{i1} : d_{i2} : ... : d_{il}$, $\forall j \in \{1, 2, ..., l\}$, $d_{ij} \in [0, 1]$. However, it is often impossible to define such data vectors. For example, to normalize a data type $l_i = celcius\ 1 : 1 : degree_i$ such that $degree_i \in [0, 1]$, a minimum and maximum degree should be specified for temperature information. SUBDUE automatically determines the high and low values from these type of labels in input graph G, therefore easily generalizing over any set of attributes.

If the desired behavior of ExtSUBDUE is to work on a specific range for a certain vector data type across current and future graphs, these ranges can be specified by additional nodes in each input graph G. For example, if the user works within a range of $[0, 100]$ °C for temperature data, additional two vertexes which are not connected to any other vertex can be inserted into the input graph so that the distance between them is the maximum value allowed for that data type. An example of this situation is given in example 3.4.2.

Example 3.4.2. *Range Setting Example in ExtSUBDUE.*
Consider an input graph G that has node labels of the form $l_i = name\ 1 : 1 : d_i$ that contains a set of temperature measurements in °C. To allow ExtSUBDUE to work with these type of graphs consistently, the range can be set explicitly. If no such range information is provided, each graph G ExtSUBDUE works on can result in a different $maxDistance_{name,1,CB}$, and therefore an inconsistent distance function. Example Graph G is represented as follows:

```
v 1 temperature 1:1:28
v 2 temperature 1:1:35
v 3 temperature 1:1:-12
v 4 temperature 1:1:-5
v 5 temperature 1:1:17
... (edges)
```

For input graph G given above, SUBDUE will automatically set $maxDistance_{temperature,1,1}$ to 47. For example, for $label_1 = temperature\ 1 : 1 : 28$, $label_3 = temperature\ 1 : 1 : -12$ and $label_5 = temperature\ 1 : 1 : 17$, the distance function $VectorDistance()$ will return the following:

- *$label_1$ and $label_3$:*
  $VectorDistance(28, -12, temperature, 1, 1) = \frac{40}{47} = 0.85$
- *$label_1$ and $label_5$:*
  $VectorDistance(28, 17, temperature, 1, 1) = \frac{11}{47} = 0.23$
- *$label_3$ and $label_5$:*
  $VectorDistance(-12, 17, temperature, 1, 1) = \frac{29}{47} = 0.62$

If the two vertices defining the two extremes are added to G, a new distance function is generated. Extended Graph G' is given below:

```
v 1 temperature 1:1:28
v 2 temperature 1:1:35
v 3 temperature 1:1:-12
v 4 temperature 1:1:-5
v 5 temperature 1:1:17
... (edges)
v 6 temperature 1:1:-20
v 7 temperature 1:1:100
```

In G', $maxDistance_{temperature,1,1} = 120$. If, for any graph G_i, these two vertexes are added at the end of the graph, then $maxDistance_{temperature,1,1}$ will always be set to 120, regardless of the samples in it. The distance function can now work consistently across a set of graphs. The distance function given above changes accordingly:

- *$label_1$ and $label_3$:*
  $VectorDistance(1:1:28, 1:1:-12, temperature, 1, 1) = \frac{40}{120} = 0.33$
- *$label_1$ and $label_5$:*
  $VectorDistance(1:1:28, 1:1:17, temperature, 1, 1) = \frac{11}{120} = 0.09$
- *$label_3$ and $label_5$:*
  $VectorDistance(1:1:-12, 1:1:17, temperature, 1, 1) = \frac{29}{120} = 0.24$

While setting such extremes can be difficult in n-dimensional data where $n \gg 1$, ExtSUBDUE handles these vectors based on their own distribution. If there is enough data, this would be sufficient for many real-world applications. ■

2. **Modified Match Cost Calculation:** When matching two subgraphs in SUBDUE, the following transformations are allowed to accommodate inexact matching:

   - Vertex/edge addition
   - Vertex/edge deletion
   - Edge direction changing

- Label substitution

The cost of each transformation is fixed in classical approach of SUBDUE. We think that a better inexact matching algorithm would be the one that assigns a cost to label substitution based on their semantic meaning. For *numeric* and *string* labels, we do not modify this cost, which equals to the default cost of 1.0. On the other hand, *vector* labels have both *string* and *data* parts, therefore the dissimilarities between their *data* counterparts can determine the matching (transformation) cost of two labels.

Going back to the example in figure 3.4, we consider the vertexes in g_1 and g_2 as having vector labels by assigning them a uniform node label and a distance function. For simplicity, each vertex's *numeric* label l_i is transformed to its equivalent *vector* label l'_i in the form of $\textit{node}\ 1:1:l_i$. G, g_1 and g_2 then becomes G', g'_1 and g'_2, respectively. In this case, it is certain that $MatchCost(g'_1, g'_2) \leq 0.5$. In fact, if the modified input graph G' is considered to be limited to g'_1 and g'_2, $maxDistance_{node,1,1} = 8.59 - 0.08 = 8.51$. The matching cost of two subgraphs $MatchCost(g'_1, g_2)$ then becomes:

$$MatchCost(g'_1, g'_2) = \frac{totalCost}{6}. \tag{3.12}$$

where *totalCost* is calculated as:

$$\begin{aligned} \textit{totalCost} &= (\textit{LabelMatchFactor}(\textit{node}\ 1:1:2.21, \textit{node}\ 1:1:2.22)+ \\ &\quad \textit{LabelMatchFactor}(\textit{node}\ 1:1:8.56, \textit{node}\ 1:1:8.59)+ \\ &\quad \textit{LabelMatchFactor}(\textit{node}\ 1:1:0.12, \textit{node}\ 1:1:0.08)) \\ \textit{totalCost} &= 0.001 + 0.003 + 0.004 = 0.008 < 0.5. \end{aligned}$$

The suggested cost calculation is more intuitive and useful than not considering semantic meaning of labels. Normally, real-world graphs having continuous attributes need to be discretized so that graph mining tools can work on them. We lift this limitation by handling these type of labels within the mining algorithm.

3. **Controlled Discovery:** In some tasks, we felt the need to limit the discovery process to get a quick analysis of the pre-defined substructures we feed into ExtSUBDUE. For this purpose, we define three modes for the discovery process, specified via *-discovery* parameter:

   - No discovery (*-discovery 1*): In this mode, the unsupervised pattern mining process is not performed. The graph is compressed with only pre-defined subgraphs, if there are any.
   - Pre-defined discovery (*-discovery 2*): First, the graph is compressed with pre-defined subgraphs. In the frequent pattern mining algorithm, we limit the initial one-vertex substructures to vertexes representing compressed pre-defined subgraphs' instances. The user can then use ExtSUBDUE to analyze the interactions of the structures provided by the domain expert and

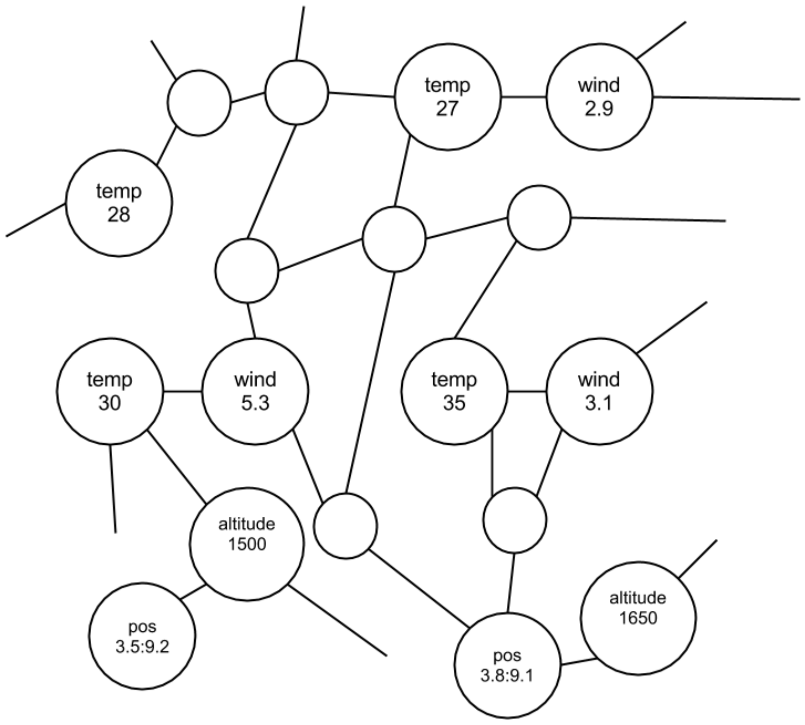


Figure 3.5: An example graph that requires forcing diversity.

their surrounding nodes. While the initial one-vertex patterns are limited, they are extended in every possible way in later steps. In effect, the only limitation enforced is that each final frequent subgraph will include at least one pre-defined substructure as one of its vertices.

- Regular discovery (*-discovery 3, default*): If no discovery mode is provided explicitly, ExtSUBDUE's default behavior is to have no limitations in its mining procedure.

4. **Forcing Diversity:** Since ExtSUBDUE is able to work in the space of real numbers in graphs, subgraphs with slightly different labels can be instances of each other with very low matching costs. On the other hand, this phonemenon is likely to result in reporting of very similar substructures in the output. Example 3.4.3 is essentially a showcase of this problem:

Example 3.4.3. *Final Output Similarity Example.*
Consider the partially shown input graph G in figure3.5. Each edge in G is labeled as 'link'. For the sake of simplicity, we leave out the details of the distance functions for given labels (distance function type, normalization constant etc.). We assume that the labels having the same name simply match each other within the current configuration. If the rest of the graph is not taken into account, an unsupervised discovery step will bring about these different substructures in order:

```
Subgraph 1:
```

```
v 1 temp 1:1:35
v 2 wind 1:1:3.1
u 1 2 link
(3 instances)

Subgraph 2:
v 1 temp 1:1:30
v 2 wind 1:1:5.3
u 1 2 link
(3 instances)

Subgraph 3:
v 1 temp 1:1:27
v 2 wind 1:1:2.9
u 1 2 link
(3 instances)

Subgraph 4:
v 1 pos 2:2:3.5:9.2
v 2 altitude 1:1:1500
u 1 2 link
(2 instances)

Subgraph 5:
v 1 pos 2:2:3.8:9.1
v 2 altitude 1650
u 1 2 link
(2 instances)
```

One property of ExtSUBDUE is that slightly different frequent substructures, although they match with a substructure with a higher score, will be reported if their values are high enough. In this specific example, the first three patterns represent the same structure, while the subsequent two are instances of a different concept. Given enough number of output substructures, all frequent and distinct subgraphs may be captured by ExtSUBDUE. However, a substructure having 1000 instances is bound to surpress another with 500 instances in the output list if the number of desired final patterns is less than 1000, since each embedding of the former may occupy a spot on the final list. This causes the algorithm to miss distinct patterns in such cases. To cope with this problem, we force ExtSUBDUE to include a frequent pattern in its output list, only if it does not match one with a higher score that is already on that list, within the elasticity limits allowed. If output diversity is forced with (-diverse) option, the result obtained as:

```
Subgraph 1:
```

```
v 1 temp 1:1:35
v 2 wind 1:1:3.1
u 1 2 edge
(3 instances)

Subgraph 2:
v 1 pos 2:2:3.5:9.2
v 2 altitude 1:1:1500
u 1 2 edge
(2 instances)
```

■

The effect of (*-diverse*) option is that it drastically reduces the number of redundant patterns in the output list, while retaining the most representative instance of each class. As a result, each output substructure is different from others. Substructures with better values do not overwhelm less-frequent but diverse patterns. One side effect is that if inexact matching is allowed, a pattern may also match its sub-patterns, even in graphs having only *numeric* or *string* labels. As a result, the best one among a pattern and its matching subgraphs will be reported in the output, excluding others with lower compression ratios.

5. **Machine-Readable Output of Instances:** In many real-world tasks, the actual instances along with frequent substructures can also be of great use. For example, in section 4, we develop a novel interactive object detection framework that aims to detect and localize desired objects in satellite images. We solve object detection task by reducing it to Subgraph Isomorphism (SI) problem, in which each instance corresponds to an object. Thus, the instances also need to be reported in a machine-readable format so that the result can be parsed to highlight the detections. For this purpose, we implemented options in ExtSUBDUE that allows printing instances of both pre-defined and unsupervised frequent substructures to file.

CHAPTER 4

AN OBJECT DETECTION FRAMEWORK USING EXTSUBDUE: GODFREY

In this section, we present a graph-based object detection method for satellite images, using ExtSUBDUE introduced in section 3.4. The proposed algorithm robustly detects repetitive objects, such as buildings, airplanes, trees etc. once the user specifies a template by drawing a bounding box around the target object. The algorithm starts with converting the given image into a relational graph, in which simple image elements such as image segments [51] are represented by nodes and spatial adjacency is embedded via edges. The next step involves user intervention by marking a target object by hand to find similar objects in the scene. The target object is translated into a subgraph having the same form as the main graph, yet it only includes a subset of the nodes belonging or adjacent to the object. After a representative subgraph of the object is extracted, we elaborate graph isomorphism to find similar subgraphs in the scene. The resulting instances are examples of our target object class. Figure 4.1 reflects the flow chart of our proposed algorithm. We optimize the system to help the user refine the output by receiving feedback in the discovery process. We utilize ExtSUBDUE to solve the graph matching problem. Experiments are conducted with buildings and airplanes, using GeoEye and Ikonos images. We name the method GODFREY, standing for Generalized Object Detection Framework for REmotelY Sensed Images.

4.0.1 Problems of Object Detection in Remotely Sensed Images

A quick look in the remote sensing literature shows that there is not a single route map in detecting man-made objects like buildings or roads, nor there exists a single appearance model capable of representing natural objects, such as, trees or rocks. Countries, even regions have different styles for building even the most basic structures. The intra-class variance emerging in many real-life problems constitutes the main reason why pattern recognition has recently evolved from traditional method of object-based training and testing, to using semantic analysis and interactions of objects [52]. For example, it is easier to recognize rooftops using the accompanying shadows [53].

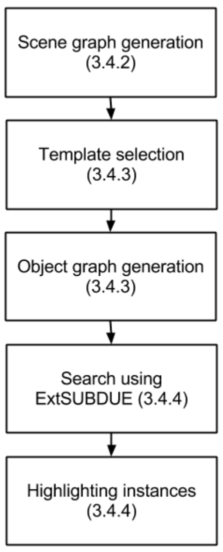

Scene graph generation (3.4.2)
Template selection (3.4.3)
Object graph generation (3.4.3)
Search using ExtSUBDUE (3.4.4)
Highlighting instances (3.4.4)

Figure 4.1: Flowchart of GODFREY.

Moreover, nearby objects tend to look like each other, such as buildings of a site sharing the same roof material. In this study, we propose an object detection framework, which aims to use such kinds of guiding information in object matching problems. We embed the contextual knowledge both implicitly and explicitly, in the form of adjacency relations with surroundings and in node labels as additional information, respectively.

With the increasing spatial resolution available in satellites, object detection methods in remote sensing have taken a leap from pixel-based measurements to region-based similarities. Early methods in Object Based Image Analysis(OBIA) classify pixels, because objects like buildings, trees and many others fit in a single pixel [54]. With commercial satellites at resolutions surpassing 0.5 meters per pixel (Worldview-2), many objects consist of numerous pixels, and can be seen as a combination of separately recognizable sub-parts.

In High-Resolution Remote Sensing imagery, due to the high variation among the samples of the target objects, the success of generic object detection approaches such as the work of Viola & Jones [55] or that of Jain et al. [56] are limited. Methods depending on independent detection of object components [57] are not practical for remote sensing objects, since object parts may not be easily identifiable. Pure learning based supervised methods [58] perform very well on well-defined objects like faces or airplanes. However, they require a high number of positive and negative samples to train an object. The main advantage of our method is that, to train a new class, only one representative example is sufficient. Our system allows the user to refine the results by marking more objects of the same type, correcting false positives, or lowering the similarity threshold. In this sense, we employ an interactive approach to improve the final results, similar to the live feedback procedure of Yao et al. [59].

4.1 Object Detection as a Graph Mining Problem

Our methodology exploits a graph-based data mining tool on the graph representation of input image to detect instances of a template object. Among many others, a successful detection algorithm with a data mining approach is presented by Inglada and Michel in [60]. Their method runs on a multi-level segmentation, representing the image segments and their relationships in an attributed relational graph form. They utilize explicit shape features of segments as node attributes and use this information in the graph matching step, similarly with our algorithm.

The marriage of image segmentation and SUBDUE is first reported in [61], while an application of such an approach on real data is presented in [62]. In [61], a hierarchical segmentation algorithm called RHSeg [63] is used to form a more abstract representation of the image. RHSeg, Recursive Hierarchical Segmentation, can optionally group the homogenous segments into clusters based on their spectral features. A layer of the hierarchic segmentation is selected manually, and it is transformed into a graph in which segments are represented with vertexes, and edges link adjacent segments. The cluster index of a segment is used as its conceptual vertex label. SUBDUE is run over this graph to discover repetitive patterns, just as in [62]. Zamalieva et al. [62], although their methodology is also based on RHSeg, generate graphs in a different manner. Their algorithm first discovers frequent relations on graphs by clustering edges based on the features of the two vertexes at each end, forming a vertex feature co-occurence space. The corresponding graph of the image is generated using unlabeled nodes and labeled edges, with each edge bearing a repetitive relation type. Both methods use unsupervised learning feature of SUBDUE, and therefore discover abundant patterns in the image. We utilize the pre-defined compression stage instead of unsupervised learning to work in a more controlled environment where we can detect desired objects.

4.1.1 Scene Graph Generation

The initial process in SUBDUE is graph generation, i.e. creating a graph representation of the input image. In order to work on a more compact level rather than the pixels, we apply an oversegmentation operation to get homogenous image segments, using a graph-based segmentation by Felzenszwalb et Huttenlocher [51]. The preferred segment size depends on the template object's size, and is set as a fraction of it. While optimal segmentation parameters can be learned using a metric such as Normalized Mutual Information (NMI), we *choose* not to. The fact that we do not need a perfect segmentation wipes away the burden of choosing the right parameters. Instead, we only perform an over-segmentation of the image to reduce the number of nodes in the final graph. Segments, the basic local structures obtained from the image, are represented with vertices in the scene graph. The labels of these nodes basically summarize the local features, and the spatial arrangement of local patches is preserved via putting edges between neighbor nodes.

Algorithm 2 summarizes our approach in the scene generation process. The reader may assume that the node list V consists of image segments. The edge list E is generated according to the rules mentioned above.

Algorithm 2: Graph Generation in GODFREY

```
   1
   Input: Node list $V$, edge list $E$, distance function $distFunc$
   Output: Graph $G$
1  foreach node $v_i$ in $V$ do
2      Extract features $\mathbf{f_i} = (f_1, ..., f_D)$;
3      Form node label $l_i$ = '$node\ distFunc : D : f_1 : f_2 : ... : f_D$';
4      Add node $n_i$ to $G$, with label $l_i$ representing $v_i$;
5      foreach neighbor node $v_j$ in $V$ do
6          Add edge $e_{i,j}$ to G;
```

At step 2 of algorithm 2, local description of the node is extracted and stored in a $1xD$ array, which constitutes the feature vector $\mathbf{f_i}$. We have implemented a number of features to be used single-handedly, or within combinations. All of the features store local geometry rather than global features, thus they are, by definition, descriptions of implicit object parts. Below, we give a list of implemented features for this step:

- Mean color of the segment on R-G-B bands ($D = 3$),
- Mean entropy of the segment on R-G-B bands ($D = 3$), with entropy defined as in [64],
- A set of shape features defining the segment including $\frac{MajorAxisLength}{MinorAxisLength}$, $\frac{MajorAxisLength}{Perimeter}$, $\frac{Perimeter}{Area}$, $Eccentricity$, $Solidity$ ($D = 5$),
- Hu moments of the segment [65] as a shape descriptor ($D = 7$),
- Angular Radial Transform moments of the segment [66] as a shape descriptor ($D = 35$).

Among these features, mean color and entropy represent color and texture information of a segment, respectively. The remaining three features implemented for segments encode contour-based shape context of the region. All of these features are normalized to $[0, 1]$ range. Combinations of them are generated by concatenating different features to obtain a longer vector.

Context Information: A vertex's label can additionally embed the features of its surrounding vertices, representing the context the object is in. If *context* is enabled and context level is set to 1, feature vectors of a vertex's immediate neighbors are averaged, and the result is concatenated with its own feature vector $(2 \times D)$. The data vector representing the context is weighted to reduce its effect, which would otherwise equal that of a segment's own data. If the context level is set to 2, averaged features of second-degree surroundings (neighbors of neighbors) is added to the end of its own feature vector $(3 \times D)$, and so on.

The major idea behind scene generation and object specification is given in figure 4.2. The original (small) image is shown in figure 4.2a, with the segmentation of the image in figure 4.2b. The user specifies the target object by drawing a bounding box around it as in figure 4.2c, and the segments representing the object in figure 4.2d are processed to form the object graph. Figure 4.2e shows the additional segments contributing to the object graph if context (level 1) is enabled.

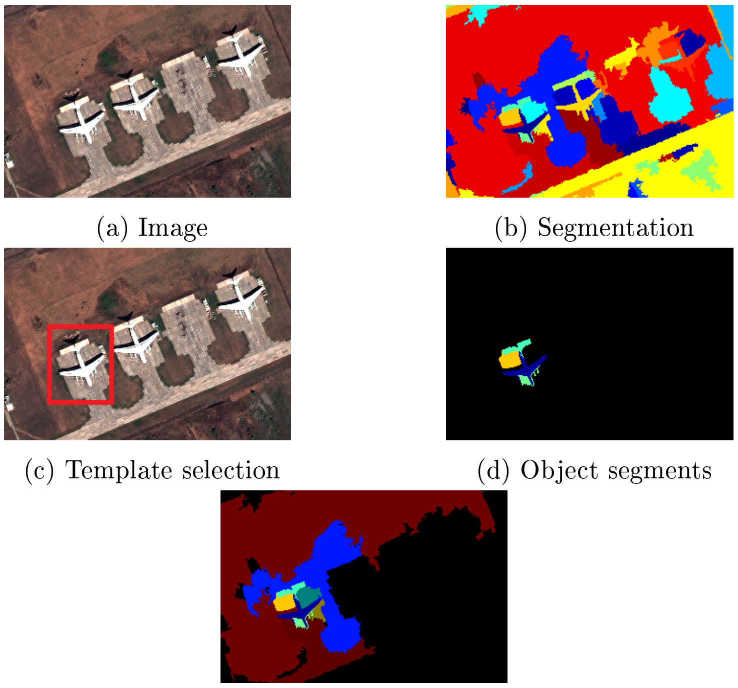

(a) Image
(b) Segmentation
(c) Template selection
(d) Object segments
(e) Context contribution

Figure 4.2: Object segment selection example.

4.1.2 Object Graph Generation

The initial scene graph generation is followed by a specification of the object via user input. To detect instances of a class in the input image, the user draws a bounding box around the target object. The nodes within the graph extracted from **inside** the bounding box, along with their edges, define the object. However, the full *object* subgraph under the bounding box does not model the final template for efficiency considerations. From the *object* graph, the connected 4-vertex subgraph whose corresponding segments cover most area of the bounding box is selected to represent the object. As a result, we select the object parts that cover the coarse appearance of the object, rather than focusing on the details. This pre-defined model is a *structured* part of the object that covers most of its area, i.e. which constitutes an improvement over bag-of-visual-words methods in Computer Vision(BOW) [67] where spatial information is mostly discarded. Such approaches ignore the relative spatial arrangement of words (segments in our case). The number of vertices to be included in the object graph can be more, and the representative power of the object graph increases with this count. However, 4 is found to be the sweet spot of the performance vs. accuracy trade-off, since the SI checks often take exponential time with the size of the patterns to be searched for.

4.1.3 Object Detection with Subgraph Discovery

After the *object* and the *scene* graphs are constructed, they are passed to ExtSUBDUE. In the search step, we simply find the embeddings of the pre-defined *object* graph in the *scene* graph, which yields similar objects in the scene. The reader should be aware of the fact that this supervised search is the $CompressWithPreDefinedSubs()$ function in algorithm 1. The rest of the search to find the frequent patterns is not performed. After the instances of the *object* graph is found, they are recorded. For each instance in the list, its segment with highest intensity is marked as a detection on the final graph. After all detections are marked, they are compared with the ground truth, and the performance of the algorithm is calculated.

Post-processing: An interesting feature of ExtSUBDUE is that for each instance, we have a match cost that defines the cost of matching this specific instance to the *object* graph. To be less reliant on selection of similarity degree, we apply Otsu's thresholding [68] to the match costs of the detections, and eliminate those instances that have a cost more than the result. To eliminate false detections, we remove very large results from the final mask. This size-based post-processing depends on the size of the template object, with maximum size allowed set empirically as $2 \times size(template)$.

The pre-defined search mechanism of SUBDUE is subject to the same parameters as the main body of the algorithm, though it is not as much optimized. The *threshold* parameter affects the detection performance. In the chapter 5, we show how robust GODFREY is to the selection of this parameter. The proposed approach is meant to solve the problem of detecting small objects in satellite images.

CHAPTER 5

EXPERIMENTS

In this chapter, we demonstrate ExtSUBDUE's advantages in working with numeric attributes over a number of methods. In section 5.1, using an artificially-generated dataset, we estimate ExtSUBDUE's performance in the problem of finding specific frequent patterns embedded in a database. In contrast to other methods, the node labels in this database consist of numeric values, rather than class labels. Then, we evaluate GODFREY, our object detection framework for remote sensing images, in section 5.2. GODFREY uses ExtSUBDUE to solve the object matching and localization problem.

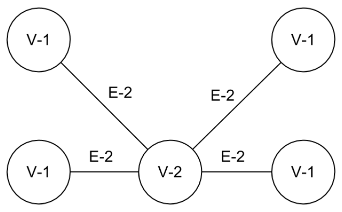
5.1 Artificial Dataset Results

We compare the performance of our proposed approach with other methods implemented on SUBDUE to work with graphs having numeric attributes. Although ExtSUBDUE can work with data vectors of any length, we stick to one-dimensional continuous data to be able to make a fair comparison with other works. As a result, each *vector* label that combines a conceptual name and a numeric attribute is of the form:

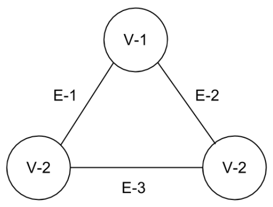
```
label 1:1:data
```

Below, we mention how the input graph transaction database is generated:

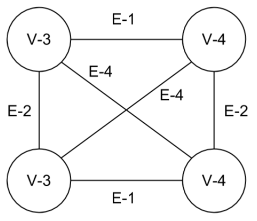
Database Generation: To evaluate ExtSUBDUE, we form a graph transaction database that embeds 6 different substructures, similar to [41]. The embedded subgraphs are given in figure 5.1. For each substructure, we create 5 separate databases including 40,80, 120, 160 and 200 graphs, respectively. In each database, %60 of the graphs include the relevant substructure, and others are randomly created. In addition, the defined substructure occupies roughly %60 of the graph it is included in, and the rest of the graph is generated randomly. In total, $6 \times 5 = 30$ database files are generated, with the overall graph count reaching $6 \times 400 = 2400$. In the setup of this experiment, we opt to work with databases of small graphs (10-15 vertices) instead of a large single graph with thousands of vertexes. The motivation behind this choice is to analyze several aspects of the problem by grouping transactions to obtain different configurations. In the labels of ExtSUBDUE's graphs, City Block (CB) distance is used.



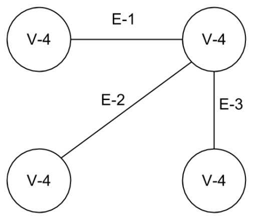
(a) Substructure 1



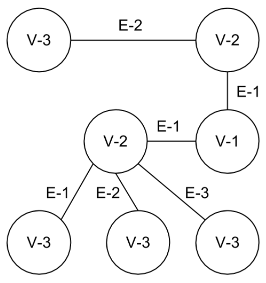
(b) Substructure 2



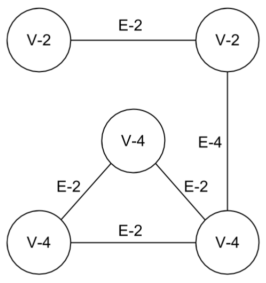
(c) Substructure 3



(d) Substructure 4



(e) Substructure 5



(f) Substructure 6

Figure 5.1: Patterns embedded in transactions.

As opposed to Ketkar et al. in [41], who use string labels such as "V-2" or "E-1", we distinguish labels in the graph with their data fields. While edge labels simply define one of the four connection types "E-1", "E-2", "E-3", "E-4" as *string* labels, the vertexes have *vector* labels whose name fields are uniform. There are four types of vertexes in the subgraphs given in figure 5.1. We distinguish such vector labels by their data fields, while giving them a uniform name such as "label". With the addition of a single-dimensional data field, the nodes in the graph are differentiable from each other.

The data attributes of these vertices are sampled from four different Gaussian distributions, corresponding to the four vertex label types. An histogram depicting the values sampled from each distribution is given in figure 5.2. The data fields of the four vertex classes are sampled from Gaussian distributions specified below:

- V-1 class: $\mu = 0; \sigma = 3$,
- V-2 class: $\mu = 10; \sigma = 3$,
- V-3 class: $\mu = 20; \sigma = 5$,
- V-4 class: $\mu = 30; \sigma = 5$,

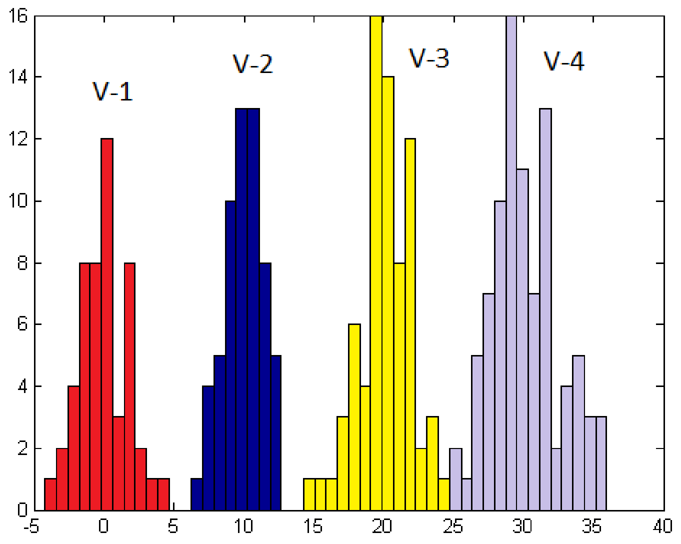


Figure 5.2: The distribution of data points for vertex classes.

We have implemented several methods based on SUBDUE that can work with numeric labels to evaluate our algorithm. Approaches using clustering as a pre-processing step are given in section 5.1.1. Additionally, 3 methods working on directly numeric labels are given in section 5.1.2.

5.1.1 SUBDUE with Clustering

First family of algorithms include a pre-processing phase to cluster the data attributes and assign corresponding vertices the group information as the string label. If a node's data field is labeled to be in cluster 4, it is assigned the label "V-4". After the graph databases are pre-processed by using one of the methods, default SUBDUE works on this compatible graph to discover the *best* substructure and its instances in each. The implemented approaches in this class are:

- GT: The index of the distribution for each data sample defines its label. The resulting graph is used as the ground truth, and its results basically signal the best achievable performance by SUBDUE.
- Kmeans-4: Clustering the data samples using K-Means clustering [69] into 4 groups. The number of clusters is given as auxiliary information to **improve** the results.
- Kmeans-5: Clustering the data samples using K-Means clustering into 5 groups.
- MEC: Clustering the data using Minimum Entropy Clustering [70] into a maximum of 10 groups. MEC algorithm aims to find the correct number of classes by estimating intra-class and inter-class variance, and then clusters the data using standard K-Means. In our case, it failed to find all four classes by determining the cluster count as 3, by putting V-3 and V-4 into the same class.
- NoLabels: The data fields of vertices' labels are discarded, meaning they are uniformly labeled.
- Ranges: Implementation of the work in Romero et al., presented in [45] and [46]. The data samples are separated into four data ranges using an iterative optimization function on pair-wise distances, and the index of the range for each sample is used as its conceptual label in the pre-processed graph.

5.1.2 SUBDUE with Numeric Labels

In addition to the aforementioned solutions that process the data and replace them with their grouping labels in the knowledge discovery process, there have been efforts to use the numeric labels directly in SUBDUE. The three approaches introduced by Baritchi et al. in [35] are:

- Exact: The numeric labels l_i and l_j match if and only if $l_i = l_j$. This is SUBDUE's default behavior if it encounters numeric labels in graphs.
- Inexact: Introducing tolerance to the label matching procedure, l_i matches l_j, if and only if $||l_i - l_j|| < threshold$.
- Prob (Probabilistic): $MatchCost(l_i, l_j)$ is the probability of l_j being drawn from a probability distribution with mean l_i and standard deviation σ. σ is set as 1, letting the program know its correct value.

The algorithms in section 5.1.2 require no pre-processing to discretize the data attributes. We have implemented them by modifying SUBDUE. Their alternatives in section 5.1.1, on the other hand, use the original SUBDUE implementation without any improvements. After the pre-process step, if any, the graph is fed into the corresponding SUBDUE implementation (original, modified, ExtSUBDUE) and the *best* substructure in each database is searched for. If the structure of the discovered subgraph matches the actual embedded substructure, we mark it as a match. Otherwise, it is considered as a miss. We also calculate the ratio of number of detected instances over the correct number in the database. The results are averaged for the 40, 80, 120, 160, 200-sized transaction databases for each pre-defined substructure and similarity *threshold* setting.

Accuracy vs. ***threshold*****:** *threshold* is provided to SUBDUE or ExtSUBDUE using the -*threshold* parameter. In figure 5.3, an accuracy-based comparison of the implemented algorithms is shown. Based on this chart, it can be inferred that SUBDUE reaches the peak of its performance with *threshold* set at a small value of 0.06 and outperforms is alternatives even with very small tolerance allowed. In this case, MEC fails to estimate the correct number of clusters by determining it as 3, instead of 4 (by assigning V-3 and V-4 to the same class). As a result, its final discovered patterns incorrectly include only three types of labels for vertexes, which is quite misleading. In fact, the errors in the pre-processing step are not recoverable in later stages, which applies to all clustering-based methods. On the other hand, ExtSUBDUE correctly distinguishes the four classes from each other in all of its detections.

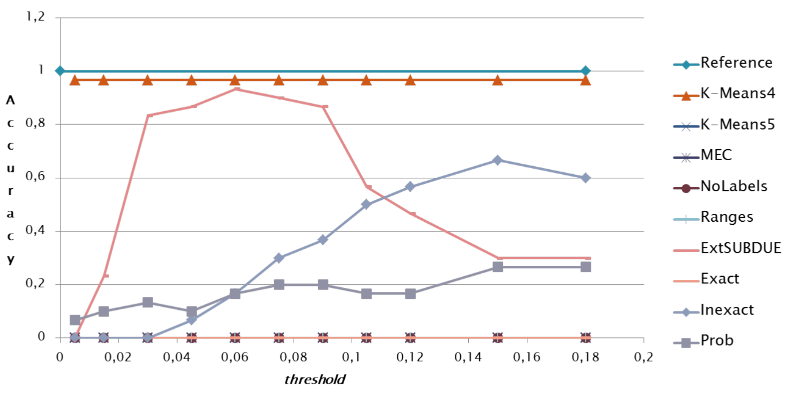


Figure 5.3: Accuracy vs. *threshold* of SUBDUE variants.

The difference between Inexact and ExtSUBDUE is that ExtSUBDUE is more error-tolerant even with a small *threshold*, although they use a very similar logic in matching of two *vector* labels. The difference is that in ExtSUBDUE, a collective matching cost is calculated, whereas in Inexact label matching [45], only pair-wise label matching costs are considered. Since graph isomorphism is an iterative process consisting many label matching calls, this results in the following advantage of ExtSUBDUE: Similarity threshold is enforced over the total cost, rather than applying it separately to each label matching function call. In effect, existence of strong partial mappings helps the algorithm to be even more tolerant in considering relatively non-similar parts as matches, which is not the case in Inexact [45] scheme. Probabilistic matching, on the other hand, fails to outperform ExtSUBDUE for any of the given *threshold* configurations.

The decline in ExtSUBDUE's performance as *threshold* increases can be attributed to the fact that it overextends the substructures after the peak, since its tolerance gets too high. ExtSUBDUE uses a single threshold for both structural (vertex/egde deletion, insertion) and label-wise (label matching) transformations. However, other methods compute such operations separately. We argue that if no overextension is allowed, *threshold* should be selected as low as possible.

The challenging performance of SUBDUE using K-means with 4 classes emerges from the fact that the correct number of classes is known. This method is highly reliant on the nature of the data, which may not be easily inferred if the data is multi-dimensional. K-means with 5 classes, for example, fails miserably since the correct vertex classes can not be inferred. Automatic determination of the number of groups, as in the case of MEC, can fail even in the given one-dimensional feature space. ExtSUBDUE is less reliant on the true distribution of the data as showcased in figure 5.2.

Instance recall vs ***threshold*****:** Figure 5.4 shows the ratio of correctly detected instances that the algorithms could identify over actual embeddings. For each algorithm, only correctly detected substructures' instances are taken into account for this metric. The results are averaged for all substructure types, and all transaction sizes. As *threshold* increases, ExtSUBDUE catches up with its alternatives. An interesting point is that although SUBDUE with probabilistic matching (Prob) misses most of the patterns, the instances of detected ones are mostly obtained. The sweet spot between accuracy and recall for ExtSUBDUE depends on the data. However, even with small threshold values, ExtSUBDUE hints enough information on the substructures present in the input graph(s), which often suffices for a quick initial analysis.

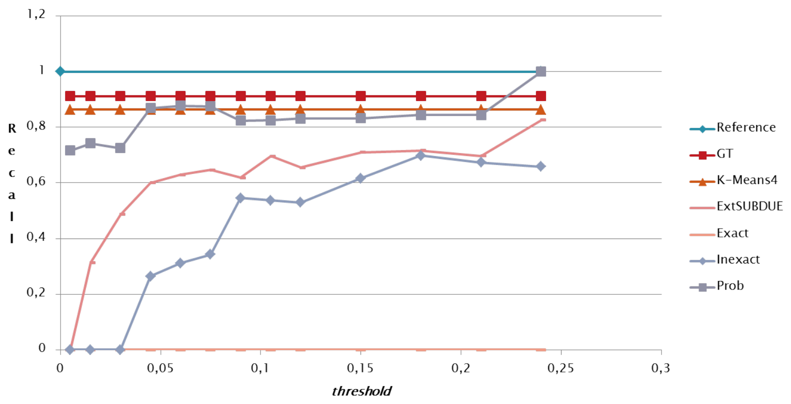


Figure 5.4: Recall vs. *threshold* curve of SUBDUE variants.

Time vs. *threshold***:** Since label matching for *vector* labels is much more costly than a strict equality check, ExtSUBDUE's performance is in direct proportion with the number of label matching checks. A time analysis of the algorithms is given in figure 5.5. We believe that the gap between ExtSUBDUE and others can be closed by optimizing its $LabelMatchFactor()$ function. In the future, we plan to implement a hash table that consists of values for latest comparisons, to avoid re-calculation of pairwise match costs as much as possible.

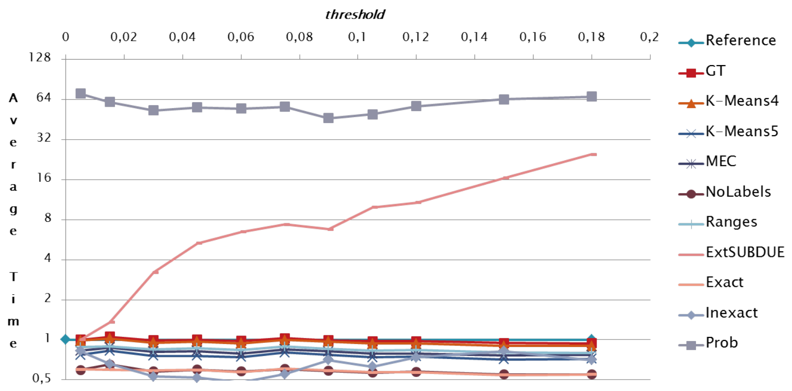


Figure 5.5: Time vs. *threshold* curve of SUBDUE variants.

5.2 Object Detection Evaluation

In order to test the object detection capabilities of GODFREY, we have prepared a dataset of 10 satellite images that include building and airplane objects. The dataset is divided into two parts, with 5 images used in building detection, and 5 images in airplane detection evaluation. Each image has a size around 1000x1000 or 1500x1000 pixels. For building detection, we used images having a resolution of 2 meters/pixel, while the airplane dataset consists of images at 1 meter/pixel. Each image contains 15-100 samples from the target object class. In figure 5.6, building and airplane examples existing in the dataset are demonstrated. Each image contains samples of either a specific building class or the airplane class. Although the buildings in different images vary greatly from each other, the airplanes are very similar in terms of appearance (commercial aircraft only). Therefore, we have 5 different types of buildings, yet only one aircraft class in 5 images.

Each class resides in a separate image (excluding Airplane class, which has samples in 5 images). The number of objects in each class is given below:

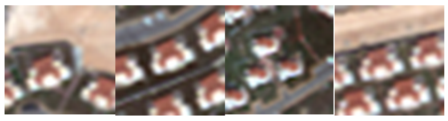
- Building class 1: 58 samples,
- Building class 2: 99 samples,
- Building class 3: 52 samples,

(a) Building class 1

(b) Building class 2

(c) Building class 3

(d) Building class 4


(e) Building class 5

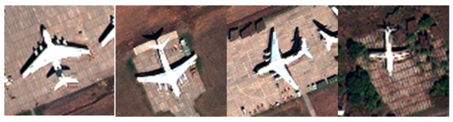

(f) Airplane class

Figure 5.6: Object classes in the dataset.

- Building class 4: 74 samples,
- Building class 5: 37 samples,
- Airplane class: 205 samples across 5 images.

The experiment setup is configured as follows:

- 8 samples are selected from each class as templates. Up to 2 *faulty* ones are discarded manually (will be explained below).
- GODFREY is run independently with each valid template.
- Performances of algorithms with each template of the class are averaged to get the final score.

The curse of working with a single template strikes at this point: The performance of the algorithm relies on the selection of the template, and more importantly, quality of the *object* graph extracted from the template. Automated selection of representative segments among all within the bounding box is prone to errors, and sometimes ambiguous background segments are incorrectly included in the object graph, rather than more descriptive object parts (such as tail of an airplane, or part of a roof for a building). When there is a single template for each run, which is our case, we think that manual selection of object segments will yield better results. However, for multiple images, the system can be fully automatic. We discuss a solution in the next chapter.

For these experiments, we discard *faulty* templates by checking their object graphs. The object graphs that include background segments are considered as *faulty* and are eliminated. In addition to GODFREY, we use two standard template matching functions implemented by D. Kroon from University of Twente, Italy to benchmark our method:

- Template_SSD: Template matching using sum-of-square differences, used in [71] and many other applications.
- Template_NCC: Template matching using normalized cross-correlation distance, used in [72, 73].

Template matching is a technique that is used to match two images based on their distances from each other. A smaller template mask is convolved with the large image to match the template to its instances. However, an important discrepancy with our case is that they work better when only a single object is present in the image. Their variations have been used in certain object detection tasks [56, 74]. Since template matching works with a single image patch representing the object, it is directly comparable with our algorithm. Learning-based approaches which are abundant in the literature need more training samples, and are therefore beyond the scope of this study.

Rotation Invariance: In order to achieve rotation invariance in the experiments, we rotate the template with 30-degree steps to obtain 12 versions. Then, we let the template matching functions work on these patches separately. GODFREY uses segments whose relative spacing does not change much, even though the object is rotated, and it is therefore already rotation-invariant to a great extent (though illumination changes are a factor here). As a result, GODFREY is called only once for each template, whereas Template_SSD and Template_NCC are called 12 times. Detections of all rotated versions are combined to form the final result image. Template_SSD and Template_NCC are given the exact same templates as GODFREY, and their results are averaged for each image, allowing a feasible comparison of the algorithms. Over the binary detection masks of all three algorithms, we apply a size-based check to discard very large detections. For each template, detections larger than 2**size*(*template*) are eliminated.

The key parameters for GODFREY are:

- *threshold* is set at six different values (0.1, 0.12, 0.14, 0.16, 0.18, 0.2). Since we apply Otsu's method to the match cost space of detections, a loose value towards the end of the spectrum (0.2) does give the algorithm stability which would not happen with a strict *threshold*. A large *threshold* will bring many false positives whose confidence scores are low, which are eliminated with adaptive thresholding.
- Features in the scene/object graphs are: Combination of color ($D = 3$) and basic shape features ($D = 5$), resulting in a purely feature vector with $D = 8$.
- First-level context is exploited. Conceptually, shadow segments near the rooftops and asphalt regions next to airplanes contribute to their graphs. As a result, we end up with a feature vector of $D = 16$.
- The *vector* labels in which node features are embedded use Euclidean distance.

In section 5.2.1, the quantitative and qualitative results are analyzed.

5.2.1 Comparison of GODFREY and Template Matching Methods

In this section, the performances of GODFREY, Template_SSD and Template_NCC are compared in "Building Detection" and "Airplane Detection" tasks. The two problems are solved using the same approach: Marking the template and finding similar objects in the scene. Below, we give the analysis for the two problems:

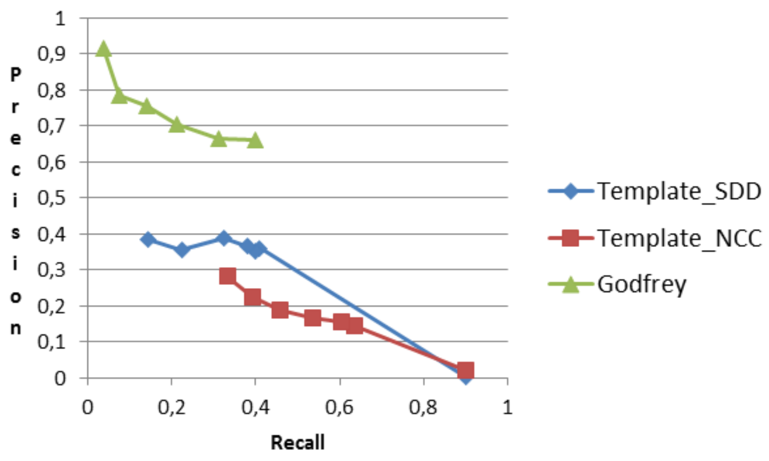


Figure 5.7: Precision vs. Recall among Building classes.

Building detection: In the building detection problem, the experiments show that ExtSUBDUE outperforms both template matching algorithms, as seen in the precision/recall chart figure 5.7. This figure is obtained by changing the *threshold* parameter in ExtSUBDUE (GODFREY) and binarizing the convolved image resulting from

Template_SSD and Template_NCC by varying threshold values. The results of the three algorithms are averaged over all images in the dataset. The asymptotic behavior of the three functions is clear enough to distinguish the template-based approaches from GODFREY. However, the relative competition of NCC-based and SSD-based template matching functions depends on the type of the template.

One of the main reasons why GODFREY performs better than template matching methods in this experiment is that buildings often emerge in radically varying shapes. For example, the buildings in figure 5.6a can not be matched using any pixel-based methodology. We consider image segments as a reliable abstraction level over low-level image. In addition to reducing the complexity, segmentation helps the algorithm to generalize over an object class with very simple features and their relations.

Table 5.1 features an image-based comparison of the performances in terms of their $fscore$s, where $fscore = \frac{2*precision*recall}{precision+recall}$. In each algorithm, the similarity thresholds are fixed across images, with their best overall performing settings taken into account. Table 5.1 hints that GODFREY is not only the most suitable, but also the most robust one for this task, since its performance does not vary as much as the others among images. Except Building Class 2, GODFREY performs better than template matching approaches. Building Class 2 is an exceptional case since it is well-suited for pixel-wise template matching. Its instances are relatively similar-shaped, and it has a high contrast with the background.

Table 5.1: F-score comparison of GODFREY, TM-1 and TM-2 in building detection.

Image ID	GODFREY	Template_SSD	Template_NCC
Image 1	**0.46**	0.16	0.21
Image 2	0.45	**0.52**	0.35
Image 3	**0.40**	0.34	0.28
Image 4	**0.5**	0.36	0.29
Image 5	**0.42**	0.35	0.3

Airplane detection: Airplanes are much more structured objects than buildings, so it is expected that pixel-based approaches perform better. However, while GODFREY and Template_NCC performs similarly with a slight decrease, the performance of Template_SSD takes a steep decline. This situation can be attributed to the fact that the aircraft, generally, are parked over asphalt areas, which decreases the contrast around the target object. GODFREY can compensate for this loss of precision by utilizing context information, since it represents the information that airplanes should be near or over grey regions. It should, also, be noted that template matching algorithms are generally used to find single objects, not multiple instances. The parameter settings for aircraft tests are the same in all algorithms as those in building detection experiments.

The precision vs. recall curve in figure 5.8 shows the dramatic performance decrease in Template_SSD, with GODFREY and Template_NCC experiencing slight differences.

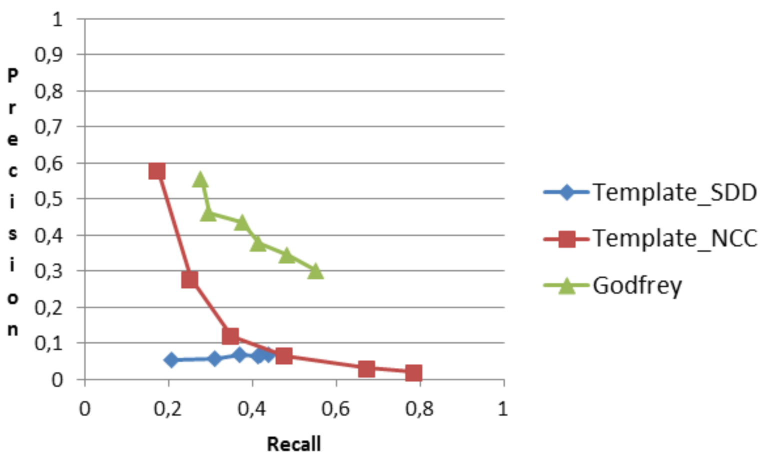


Figure 5.8: Precision vs. Recall among Building classes.

Similarly with table 5.1, table 5.2 compares f-scores of the algorithms for all of the input images. This check is useful to avoid perfect runs on one image biasing the overall performance.

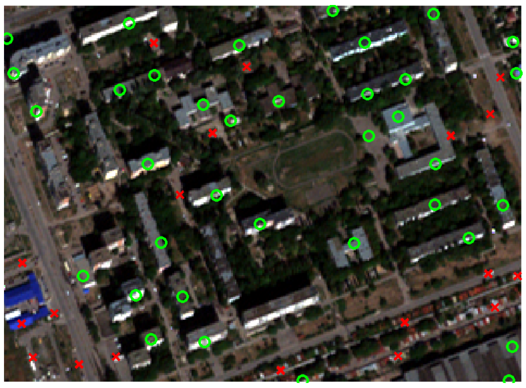
Table 5.2: F-score comparison of algorithms in aircraft detection.

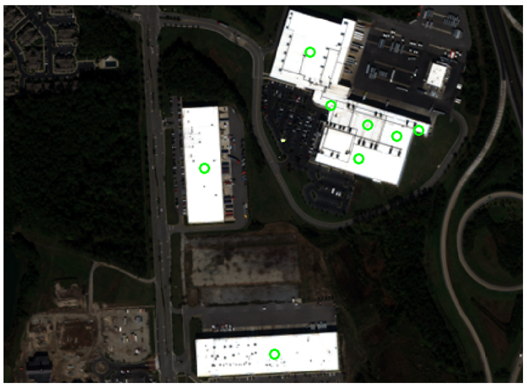
Image ID	GODFREY	Template_SSD	Template_NCC
Image 6	**0.62**	0.30	0.17
Image 7	**0.32**	0.06	0.17
Image 3	**0.34**	0.06	0.17
Image 4	**0.24**	0.10	0.07
Image 5	0.31	0.09	**0.65**

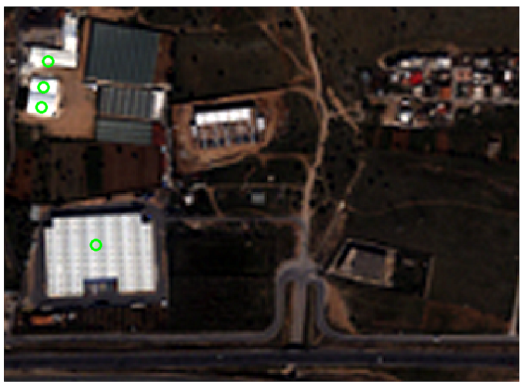
Visual results: Samples from each class can be seen in figure 5.9. Please keep in mind that these images are intended to be useful for illustration purposes. Actual images are much larger. If an object is detected multiple times, as in figure 5.9f, it is counted as one true positive, with second detection being ignored.

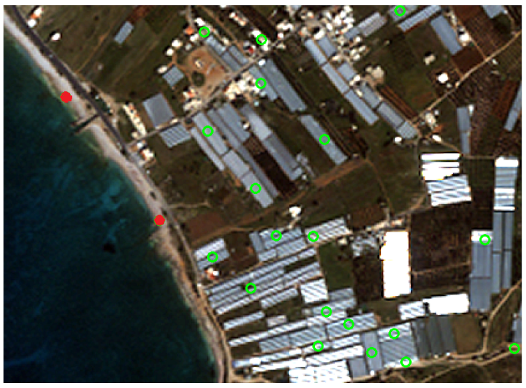
To sum up, GODFREY is an algorithm that boosts the following key features:

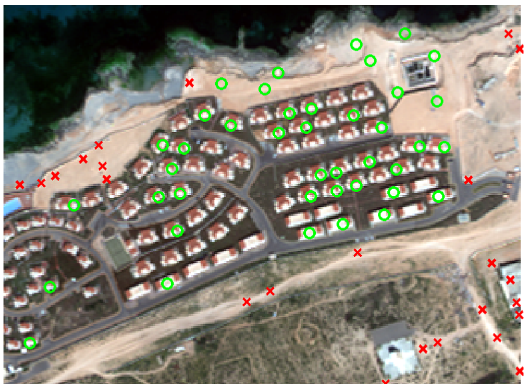
- It is quite robust object modeling using graphs, which introduces rotation and scale invariance,
- It is relatively less dependent on the specified threshold when compared to template matching methods since it applies Otsu's thresholding on the costs of detections,
- It localizes objects rather than purely detecting them in the scene.


(a) Building class 1 results


(b) Building class 2 results


(c) Building class 3 results


(d) Building class 4 results


(e) Building class 5 results

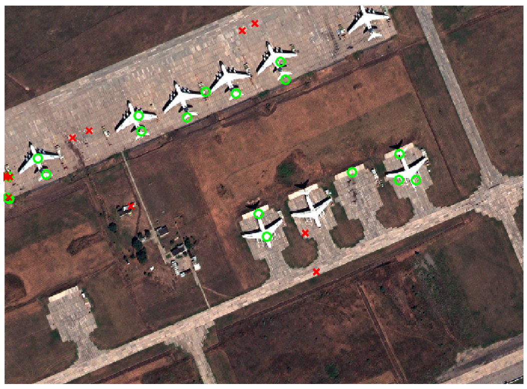

(f) Airplane class results

Figure 5.9: Sample detections (Best viewed in color).

CHAPTER 6

CONCLUSION AND FUTURE WORK

This thesis proposes a new approach to the graph mining problem by extending the classical SUBDUE algorithm. The proposed method, called Extended SUBDUE (ExtSUBDUE), is capable of working with labels having data vectors of any length as their attributes. This relieves the burden of pre-processing the data such that the numeric attributes are clustered into bins, and the cluster information replace the original data samples. The advantage of using the attributes in the discovery process helps ExtSUBDUE to discover the structural regularities in graphs with no assumptions on the distribution of the data samples. Information from various channels can be combined within the same graph with different *vector* label types, allowing formation of patterns across many data types and domains.

A powerful aspect of ExtSUBDUE is that it does not assume any true distribution of the data, as opposed to clustering algorithms that require the correct number of clusters. Clustering-based approaches perfom a one-time grouping procedure to quantize the feature space into a few clusters. This operation is prone to errors especially if the data is not linearly separable, which is mostly the case. Moreover, clustering is an operation that highly depends on the number of clusters, which is usually specified by the user. ExtSUBDUE already performs a clustering-like approach in that the final substructures discovered correspond to the most representative ones, i.e. those having most instances within allowed limits. Since cluster centers are more dense than outliers, points equal to or nearby these centers will exist in the final patterns.

Based on ExtSUBDUE, we develop an object detection framework for remote sensing images. The motivation behind the development of this system is to bring a unified solution to detection of various objects in satellite images, such as buildings, airplanes, trees, etc. These objects differ from regional targets such as residential areas, airports, and harbours in that they can be modeled with a few homogenous image segments. We believe that the proposed framework, GODFREY, is a flexible tool that can be extended to include more image features such as shape, color and texture features. While its current implementation only includes image segments as vertices in the *scene* graph, drastically different approaches can be exploited in the graph generation step. For example, SIFT keypoints [75] can be represented with vertices, with the edges embedding spatial arrangement of keypoints. Many more features can be extracted from image segments/keypoints since ExtSUBDUE works on data vectors of any length.

A potential drawback of GODFREY is that the results are heavily affected by the choice of the template, and formation of the *object* graph. We automatically form the *object* graph after the bounding box is drawn. However, existence of background segments in the object graphs becomes inevitable in some cases. There are two possible solutions in mind for this problem:

- Manual selection of object segments by letting the user click on object segments,
- Multiple training samples. If multiple object examples are provided for training, unsupervised ExtSUBDUE can be used to discover the representative subgraphs across the graphs of these examples (not the whole scene). The discovered best substructure becomes the *best* pattern representing that class. This subgraph can then be fed to GODFREY as the *object* graph.

Determination of similarity *threshold* to be used in GODFREY, on the other hand, is a different problem. It depends on the nature of the image and the object, and can not be easily inferred using unsupervised approaches. To overcome this deficit, we have decided to speed-up the algorithm on its second or latter runs. We save all of the intermediate data structures such as the image segmentation, *scene* graph, and image features. In effect, if the same image is processed again, only the *object* graph generation, knowledge discovery and subsequent segment highlighting phases are executed. Therefore, the run time of the algorithm improves greatly, allowing the user experiment with several values of *threshold*.

In summary, both ExtSUBDUE and and its application to remote sensing domain, GODFREY, are flexible approaches that can generalize over various problems. Our future plans include working with standard object detection databases such as Caltech-101 [76] to estimate the performance of GODFREY in slightly different problems. Moreover, ExtSUBDUE itself has room for further development, via optimization of its label matching function and adding canonical labeling for efficient subgraph isomorphism tests. While we are content with the results so far, the experiments and analyses hint that there is still much work that can be done.

REFERENCES

[1] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, (Washington, DC, USA), pp. 721–, IEEE Computer Society, 2002.

[2] D. J. Cook and L. B. Holder, "Substructure discovery using minimum description length and background knowledge," *J. Artif. Int. Res.*, vol. 1, pp. 231–255, Feb. 1994.

[3] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, (Washington, DC, USA), pp. 549–, IEEE Computer Society, 2003.

[4] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, (New York, NY, USA), pp. 647–652, ACM, 2004.

[5] C. Borgelt, "Mining molecular fragments: Finding relevant substructures of molecules," in *In Proc. of 2002 IEEE International Conference on Data Mining (ICDM*, pp. 51–58, IEEE Press, 2002.

[6] B. Berendt, A. Hotho, and G. Stumme, "Towards semantic web mining," in *IN INTERNATIONAL SEMANTIC WEB CONFERENCE (ISWC*, pp. 264–278, Springer, 2002.

[7] L. Holder, D. Cook, J. Coble, and M. Mukherjee, "Graph-based relational learning with application to security," *Fundam. Inf.*, vol. 66, pp. 83–101, Nov. 2004.

[8] S. Ghazizadeh and S. S. Chawathe, "Seus: Structure extraction using summaries," in *In Proc. of the 5th International Conference on Discovery Science*, pp. 71–85, Springer, 2002.

[9] J. Huan, W. Wang, J. Prins, and J. Yang, "Spin: mining maximal frequent subgraphs from graph databases," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, (New York, NY, USA), pp. 581–586, ACM, 2004.

[10] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, (Washington, DC, USA), pp. 313–320, IEEE Computer Society, 2001.

[11] M. Kuramochi and G. Karypis, "Finding frequent patterns in a large sparse graph*," *Data Min. Knowl. Discov.*, vol. 11, pp. 243–271, Nov. 2005.

[12] M. Wörlein, E. Dreweke, T. Meinl, and I. Fischer, "Edgar: the embedding-based graph miner," in *Proceedings of the International Workshop on Mining and Learning with Graphs (MLG 2006*, pp. 221–228, 2006.

[13] Motoda and N. Indurkhya, "Graph-based induction as a unified learning framework," in *Journal of Applied Intelligence*, vol. 4, pp. 297–328, 1994.

[14] M. A. Abdulrahim and M. Misra, "A graph isomorphism algorithm for object recognition," *Pattern Anal. Appl.*, vol. 1, no. 3, pp. 189–201, 1998.

[15] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, (London, UK, UK), pp. 13–23, Springer-Verlag, 2000.

[16] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, pp. 207–216, June 1993.

[17] X. Yan and J. Han, "Closegraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, (New York, NY, USA), pp. 286–295, ACM, 2003.

[18] M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen, "A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston," in *Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'05, (Berlin, Heidelberg), pp. 392–403, Springer-Verlag, 2005.

[19] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii," *CoRR*, vol. abs/1301.1493, 2013.

[20] A. Hellal and L. B. Romdhane, "Nodar: mining globally distributed substructures from a single labeled graph.," *J. Intell. Inf. Syst.*, vol. 40, no. 1, pp. 1–15, 2013.

[21] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 10, pp. 1367–1372, 2004.

[22] L. G. Shapiro and R. M. Haralick, "Structural descriptions and inexact matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 3, pp. 504–519, May 1981.

[23] W. H. Tsai and K. S. Fu, "Subgraph Error-Correcting Isomorphism for Syntactic Pattern Recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 13, pp. 48–62, 1983.

[24] I. Olmos, J. A. Gonzalez, and M. Osorio, "Subgraph isomorphism detection using a code based representation.," in *FLAIRS Conference* (I. Russell and Z. Markov, eds.), pp. 474–479, AAAI Press, 2005.

[25] J. Rissanen, *Stochastic Complexity in Statistical Inquiry Theory.* River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1989.

[26] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, Sept. 1978.

[27] A. N. Kolmogorov, “Three approaches to the quantitative definition of information.,” *Problems in Information Transmission*, vol. 1, pp. 1–7, 1965.

[28] P. Grünwald, *A Tutorial Introduction to the Minimum Description Length Principle*, ch. 1-2. MIT Press, Apr. 2005.

[29] M. Li and P. M. Vitnyi, *An Introduction to Kolmogorov Complexity and Its Applications.* Springer Publishing Company, Incorporated, 3 ed., 2008.

[30] J. J. Rissanen, “Fisher information and stochastic complexity,” *Information Theory, IEEE Transactions on*, vol. 42, no. 1, pp. 40–47, 1996.

[31] I. Jonyer, L. B. Holder, and D. J. Cook, “Graph-based hierarchical conceptual clustering,” *International Journal on Artificial Intelligence Tools*, vol. 2, pp. 107–135, 2001.

[32] W. Zhang, *State-space search - algorithms, complexity, extensions, and applications.* Springer, 1999.

[33] S. S. Long and L. B. Holder, “Graph based mri brain scan classification and correlation discovery.,” in *CIBCB*, pp. 335–342, IEEE, 2012.

[34] W. Eberle, L. B. Holder, and B. Massengill, “Graph-based anomaly detection applied to homeland security cargo screening.,” in *FLAIRS Conference* (G. M. Youngblood and P. M. McCarthy, eds.), AAAI Press, 2012.

[35] A. Baritchi, D. J. Cook, and L. B. Holder, “Discovering structural patterns in telecommunications data,” in *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*, pp. 82–85, AAAI Press, 2000.

[36] J. A. Gonzalez, L. B. Holder, and D. J. Cook, “Structural knowledge discovery used to analyze earthquake activity,” in *Proceedings of the Thirteenth Annual Florida AI Research Symposium*, pp. 86–90, 2000.

[37] W. Eberle and L. Holder, “Insider threat detection using graph-based approaches,” *Conference For Homeland Security, Cybersecurity Applications and Technology*, vol. 0, pp. 237–241, 2009.

[38] D. J. Cook, L. B. Holder, S. Su, R. Maglothin, and I. Jonyer, “Structural mining of molecular biology data,” in *IEEE Engineering in Medicine and Biology, special issue on Advances in Genomics*, p. 2001, 2001.

[39] N. Manocha, D. J. Cook, and L. B. Holder, “Structural web search using a graph-based discovery system,” in *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pp. 133–137, AAAI Press, 2001.

[40] W. Eberle and L. B. Holder, “Graph-based knowledge discovery: Compression versus frequency,” in Murray and McCarthy [77].

[41] N. S. Ketkar, “Subdue: compression-based frequent pattern discovery in graph data,” in *in OSDM ’05: Proceedings of the 1st international workshop on open source data mining*, pp. 71–76, ACM Press, 2005.

[42] I. Jonyer, L. B. Holder, and D. J. Cook, "Concept formation using graph grammars," in *Proceedings of the KDD Workshop on Multi-Relational Data Mining*, 2002.

[43] H. Bunke, "Inexact graph matching for structural pattern recognition," *Pattern Recognition Letters*, vol. 1, pp. 245–253, May 1983.

[44] G. Perez, I. Olmos, and J. A. Gonzalez, "Subgraph isomorphism detection with support for continuous labels.," in *FLAIRS Conference* (H. W. Guesgen and R. C. Murray, eds.), AAAI Press, 2010.

[45] O. E. Romero, J. A. Gonzalez, and L. B. Holder, "Handling of numeric ranges with the subdue system," in Murray and McCarthy [77].

[46] O. E. Romero, J. A. Gonzalez, and L. B. Holder, "Handling of numeric ranges for graph-based knowledge discovery," in *FLAIRS Conference*, 2010.

[47] M. Davis, W. Liu, and P. Miller, "Finding the most descriptive substructures in graphs with discrete and numeric labels," in *Proceedings of the First international conference on New Frontiers in Mining Complex Patterns*, NFMCP'12, (Berlin, Heidelberg), pp. 138–154, Springer-Verlag, 2013.

[48] M. Davis, W. Liu, P. Miller, and G. Redpath, "Detecting anomalies in graphs with numeric labels," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, (New York, NY, USA), pp. 1197–1202, ACM, 2011.

[49] T. S. Project, "Subdue manual," 2011.

[50] W. Zhang, *State-space search - algorithms, complexity, extensions, and applications.* Springer, 1999.

[51] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, pp. 167–181, Sept. 2004.

[52] J. Porway, Q. Wang, and S. C. Zhu, "A hierarchical and contextual model for aerial image parsing," *Int. J. Comput. Vision*, vol. 88, pp. 254–283, June 2010.

[53] H. Akcay and S. , "Building detection using directional spatial constraints," in *Geoscience and Remote Sensing Symposium (IGARSS), 2010 IEEE International*, pp. 1932 –1935, july 2010.

[54] T. Blaschke, "Object based image analysis for remote sensing," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, pp. 2–16, Jan. 2010.

[55] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features.," in *CVPR (1)*, pp. 511–518, IEEE Computer Society, 2001.

[56] A. K. Jain, Y. Zhong, and S. Lakshmanan, "Object matching using deformable templates," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, pp. 267–278, Mar. 1996.

[57] A. Mohan, C. Papageorgiou, and T. Poggio, "Example-based object detection in images by components," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, pp. 349 –361, apr 2001.

[58] C. Papageorgiou and T. Poggio, “A trainable system for object detection,” *Int. J. Comput. Vision*, vol. 38, pp. 15–33, June 2000.

[59] A. Yao, J. Gall, C. Leistner, and L. J. V. Gool, “Interactive object detection,” in *CVPR*, pp. 3242–3249, 2012.

[60] J. Inglada and J. Michel, “Qualitative spatial reasoning for high-resolution remote sensing image analysis.,” *IEEE T. Geoscience and Remote Sensing*, vol. 47, no. 2, pp. 599–612, 2009.

[61] J. C. Tilton, D. J. Cook, and N. S. Ketkar, “The integraton of graph-based knowledge discovery with image segmentation hierarchies for data analysis, data mining and knowledge discovery,” in *IGARSS (3)*, pp. 491–494, 2008.

[62] D. Zamalieva, S. Aksoy, and J. C. Tilton, “Finding compound structures in images using image segmentation and graph-based knowledge discovery.,” in *IGARSS (5)*, pp. 252–255, IEEE, 2009.

[63] J. C. Tilton, “Method for recursive hierarchical segmentation by region growing and spectral clustering with a natural convergence criterion,” *Disclosures of Invention and New Technology(Including Software) : NASA*, vol. GSC 14.305-1, February 2000.

[64] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, *Digital Image Processing Using MATLAB*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003.

[65] M.-K. Hu, “Visual pattern recognition by moment invariants,” *Information Theory, IRE Transactions on*, vol. 8, pp. 179 –187, february 1962.

[66] J. Ricard, D. Coeurjolly, and A. Baskurt, “Generalizations of angular radial transform for 2d and 3d shape retrieval,” *Pattern Recognition Letters*, vol. 26, no. 14, pp. 2174 – 2186, 2005.

[67] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, “Evaluating bag-of-visual-words representations in scene classification,” in *Proceedings of the international workshop on Workshop on multimedia information retrieval*, MIR ’07, (New York, NY, USA), pp. 197–206, ACM, 2007.

[68] N. Otsu, “A Threshold Selection Method from Gray-level Histograms,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[69] J. B. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[70] H. Li, “Minimum entropy clustering and applications to gene expression analysis,” in *In Proceedings of IEEE Computational Systems Bioinformatics Conference*, pp. 142–151, 2004.

[71] N. N. Dawoud, B. B. Samir, and J. Janier, “Article: Fast template matching method based optimized sum of absolute difference algorithm for face localization,” *International Journal of Computer Applications*, vol. 18, pp. 30–34, March 2011. Published by Foundation of Computer Science.

[72] L. Cole, D. Austin, and L. Cole, "Visual object recognition using template matching," in *Proceedings of Australian Conference on Robotics and Automation*, 2004.

[73] K. Briechle and U. D. Hanebeck, "Template matching using fast normalized cross correlation," in *Proceedings of SPIE: Optical Pattern Recognition XII*, vol. 4387, pp. 95–102, Mar. 2001.

[74] R. M. Dufour, E. L. Miller, and N. P. Galatsanos, "Template matching based object recognition with unknown geometric parameters.," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1385–1396, 2002.

[75] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, (Los Alamitos, CA, USA), pp. 1150–1157 vol.2, IEEE Computer Society, Aug. 1999.

[76] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Comput. Vis. Image Underst.*, vol. 106, pp. 59–70, Apr. 2007.

[77] R. C. Murray and P. M. McCarthy, eds., *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, May 18-20, 2011, Palm Beach, Florida, USA*, AAAI Press, 2011.