

A Spectral-Graph-Pattern-Matching Approach to the SAT Problem

Research Proposal

Rave Harpaz
Pattern Recognition Laboratory
Department of Computer Science
The Graduate Center
The City University of New York
365 Fifth Avenue
New York, N.Y. 10016

February 16, 2004

1 Introduction

Pattern-matching on graphs is the problem of finding a homomorphic or isomorphic image of a given graph, called the *pattern*, in another graph, called the *target*. This problem is also known as *graph/subgraph homomorphism* or *subgraph isomorphism*, and in the pattern recognition literature is usually referred to as *exact* and *inexact* graph matching.

Graph matching is a fundamental task in a variety of pattern recognition applications such as character recognition, natural language processing, and computer vision. Many techniques, both general and application specific have been proposed for solving this problem. One family of graph matching techniques based on *spectral graph theory*, and generally known as *spectral methods*, seeks to represent and distinguish structural properties of graphs using eigenvalues and eigenvectors of graph adjacency matrices. The main characteristic of this method is that the eigenvalues of graph adjacency matrices are invariant with respect to node permutations. A direct consequence of this is that if two graphs are isomorphic then their adjacency matrices will have the same eigenvalues.

In this paper we propose a novel approach for solving SAT. We first show how SAT can be formulated as a graph homomorphism problem, i.e. reduce SAT to graph homomorphism, and then develop a search strategy for SAT which is based on spectral methods. Needless to say that graph homomorphism NP-complete, merely for the fact that it is a generalization of subgraph isomorphism which is known to be NP-complete [GJ79]. Our aim is to show that this approach is more versatile and outperforms existing strategies on certain class instances.

2 A Graph Homomorphism formulation of SAT

We first discuss homomorphisms in the context of relations, and then show how it applies to graphs, boolean matrices, and operations on boolean matrices.

2.1 Relational Homomorphism [Har78, HK78]

Definition 1 (*Composition**). Let A and B be two sets, R a binary relation such that $R \subseteq A \times A$, and H a binary relation such that $H \subseteq A \times B$. Define the composition of R with H by

$$R \circ^* H = \{(b, b') \in B \times B \mid \text{for some } (a, a') \in R, (a, b) \in H \text{ and } (a', b') \in H\}$$

Note that this definition is different from the usual definition of binary relation composition. It is used in order to facilitate the idea is that a pair (a, a') in relation R can be mapped to a pair (b, b') only under the condition that the relation H associates b with a and b' with a' . $R \circ^* H$ is equivalent to $H^T \circ R \circ H$ using the regular definition of composition, where $H^T = \{(a, b) | (b, a) \in H\}$.

Definition 2 (Homomorphism). *Let $R \subseteq A \times A$, $S \subseteq B \times B$, and $H \subseteq A \times B$. H is a homomorphism of R into S iff:*

1. *H is defined everywhere on A (total), that is, for every $a \in A$, there exists a $b \in B$ such that $(a, b) \in H$.*
2. *H is single-valued on B , that is, if $(a, b) \in H$ and $(a, b') \in H$ then $b = b'$. (the first two conditions ensure that H is a mapping)*
3. *$R \circ^* H \subseteq S$.*

The idea is that for a relation $H \subseteq A \times B$ to be a homomorphism of R to S we insist that H be capable of mapping each pair in R to some pair in S . We also allow some pairs in S to be the image of no pair in R . This idea can be depicted by the graphs in figure 1, where the nodes of R resp., S are the elements of set A , resp., set B , and the edges of R resp., S correspond to the pairs in relation R , resp., S .

2.2 Graph Homomorphism[VM97, SS93]

Definition 3. *A (directed) graph $G = (V, E)$ consists of a set V and a relation $E \subseteq V \times V$. The elements of V are called vertices, and E is called the relation associated to G . It is said that there is an arc from vertex v to vertex v' if $(v, v') \in E$. A graph $G = (V, E)$ is finite if $|V|$ is finite, and is undirected if E is symmetric, in which case the arcs are called edges.*

The classic [GJ79] definition of the graph homomorphism problem is as follows. Later we show how this problem can be applied to boolean matrices and operations on boolean matrices.

Definition 4 (Graph Homomorphism). :

Input: *Two graphs $G = (V, E)$ and $G' = (V', E')$.*

Question: *Does There exist a mapping $H : V \mapsto V'$ such that $\forall (x, y) \in E \rightarrow (H(x), H(y)) \in E'$?*

The relation E associated to the graph G can be represented by a boolean matrix A , where $a_{ij} = 1$ if arc $(v_i, v_j) \in E$, and $a_{ij} = 0$ otherwise. Likewise

a homomorphism $H \subseteq V \times V'$ which is also a relation can be represented by a boolean matrix A , where $a_{ij} = 1$ if $(v_i, v_j) \in H$. Usually the relation and its corresponding boolean matrix are denoted by the same name. Using these matrices a homomorphism from one graph to another can be defined as follows, and later used to illustrate the process of tree search for solving the graph homomorphism problem.

Definition 5. *A relation $H \subseteq V \times V'$ is a homomorphism from graph $G = (V, E)$ to graph $G' = (V', E')$ if:*

1. $I \subseteq HH^T$, that is H is total (I is the identity matrix).
2. $H^T H \subseteq I$, that is H is single valued (injective).
3. $E \subseteq HE'H^T$, that is H is structure preserving.

The idea of these conditions is: for H to be total every $v \in V$ must be associated with at least one $v' \in V'$. Starting with an element of v and applying H , which in terms of matrix operations is the operation $H^T v$, where v is a boolean vector having 1 in v 's position and 0 elsewhere, and then applying H^T , which is the operation Hv , and together is $H(H^T v)$, should at least yield the original element. Thus $I \subseteq HH^T$. For H to be single valued every $v \in V$ must be associated with at most one $v' \in V'$. So starting with v' (an image), and applying H^T (getting the predecessors of v') and then applying H (getting the successors of Hv'), i.e. $H^T(Hv')$, we should get at most v' . Thus $H^T H \subseteq I$. For H to be structure preserving, if x, y are two vertices from V with images $H^T x$ and $H^T y$, then $xy^T \subseteq E$ shall imply that $H^T x H^T y \subseteq E'$, which by the first condition ($I \subseteq HH^T$) and monotonicity ($R \subseteq S \rightarrow QR \subseteq QS$), is equivalent to $xy^T \subseteq HE'H^T$, thus $E \subseteq HE'H^T$. An illustration of a homomorphism from one graph to another is presented in figure 1.

2.3 Relational/Graph Homomorphism Formulation of SAT

We first review the original formulation of the SAT problem.

Definition 6. *Let $X = x_1, x_2, \dots, x_n$ be a finite set of **Boolean variables**, and let $\bar{X} = \bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$ stand for the negations of x_1, x_2, \dots, x_n . we call the elements of $X \cup \bar{X}$ **literals**. We call a set of literals a **clause** C , and a set of clauses a **formula** ϕ . We say that a formula is in **conjunctive normal form** (CNF), if $\phi = \bigwedge_{i=1}^n C_i$, where $n \geq 1$, and each C_i is the disjunction of one or more literals.*

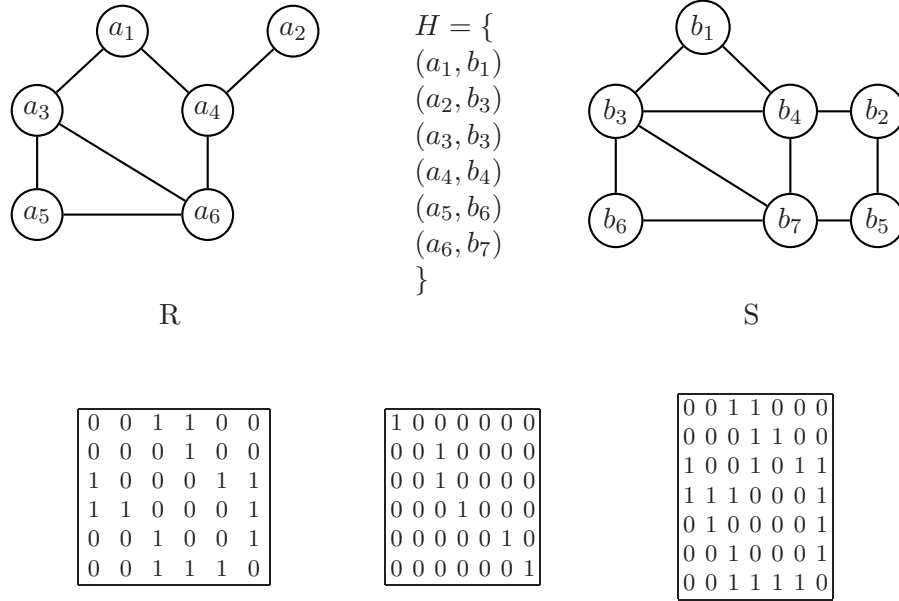


Figure 1: Relational/Subgraph homomorphism

Definition 7. A truth assignment T for ϕ is a mapping $T : X \rightarrow \{1, 0\}$. A literal u is true under T ($T \models u$) iff $T(u) = 1$. A truth assignment T for X satisfies a clause $C \in \phi$, where ϕ is a Boolean formula in CNF, iff at least one literal $u \in C$ is true under T . We say that T satisfies ϕ iff it satisfies every clause in ϕ .

Definition 8 (SAT problem).

Input: A Boolean formula ϕ in conjunctive normal form.

Question: Does there exist a satisfying truth assignment for ϕ ?

To simplify the manipulation of CNF formulas, it is helpful to represent them as sets. In doing so we eliminate the order of clauses and order of literals in each clause. A clause $C = \bigvee_{j \leq m} l_j$ is represented as the set $C = \{l_j | j \leq m\}$, e.g. $(p \vee q \vee r)$ will be denoted as $\{pqr\}$. A formula $\phi = \bigwedge_{i \leq n} C_i$ is represented as the set $\phi = \{C_i | i \leq n\}$, e.g. $(p \vee q) \wedge (\bar{p} \vee r)$ will be denoted as $\{\{pq\}, \{\bar{p}r\}\}$, or simply $\{pq, \bar{p}r\}$.

Formulated this way, the problem of finding a satisfying assignment can be viewed as the problem of finding a mapping H from variables to truth values, i.e. $H : X \mapsto \{0, 1\}$ that satisfies the formula. We know that a boolean formula in CNF is satisfiable if and only if all its clauses are mutually satisfiable, and

a clause is satisfiable if and only if one of its literals evaluates to true under some truth assignment. Therefore rather than searching for a mapping from variables to truth values, we propose that the SAT problem be viewed as the problem of searching for a mapping from clauses to literals, that is, for each clause select a literal that satisfies it, maintaining consistency in the selection.

Definition 9. let $A = \{C \mid C \in \phi\}$, that is A is the set of clauses appearing in boolean formula ϕ . Let $B = \{l \mid \exists C_i \in \phi \text{ such that } l \in C_i\}$, that is B is the set of literals appearing in formula ϕ .

Definition 10. Let $T \subseteq A \times P(B)$, $T = \{(a, l) \mid a \in A \text{ and } l \in a\}$. T is the set of all possible pairs of clauses and literals appearing in the formula, with the restriction that a clause can only be paired with a literal appearing in it.

Definition 11 (constraint graph, pattern graph). let $G_R = (A, R)$ be an undirected graph where $R \subseteq A \times A$, $R = \{(a_i, a_j) \mid \exists l \in B \text{ such that } l \in a_i \text{ and } \bar{l} \in a_j\}$. That is, R contains pairs of clauses that have a literal appearing in one element of the pair and its negation in the other element of the pair. The idea is that R represents the relationship between clauses that constraint each other.

Definition 12 (consistency graph, target graph). let $G_S = (B, S)$ be an undirected graph with loops where $S \subseteq B \times B$, $S = \{(b_i, b_j) \mid b_i \in B, b_j \in B \text{ and } b_i \neq \bar{b}_j\}$. The idea is for S to maintain the consistency of literal selection for each clause. Once a literal is selected to satisfy one clause, its negation cannot be selected to satisfy another.

To show that SAT can be formulated as a graph pattern-matching or more precisely as a graph homomorphism problem we first need to show that S , the consistency graph has the potential of being a homomorphic image of R , the constraint graph. That is, if E and E' are the set of edges of graphs R and S respectively then $|E| \leq |E'|$. Note that the same observation for the sets of vertices of R and S , V and V' respectively, $|V| \leq |V'|$ does not need to hold in the case of graph homomorphism as opposed to the problem of subgraph isomorphism. An example is the formula $\phi = \{x_1, x_2, x_1x_2\}$, there are 3 vertices in constraint graph R , but only 2 in the consistency graph S , yet still there exists a homomorphic image of R in S . Also note that the observation will not hold for formulas containing a clause that has a literal and its complement appearing in it, e.g. $\phi = \{x_1, \bar{x}_1, x_1\bar{x}_1\}$, in which case the clause is satisfiable and can be omitted from the formula. An other observation is that $|E| = |E'|$ when the formula is empty, in which case it is trivially satisfiable, or when it contains an empty clause, in which case it is trivially unsatisfiable.

Proposition 1. if $R = (V, E)$ and $S = (V', E')$ are the constraint and consistency graphs respectively then $|E| \leq |E'|$.

Proof: The proof is by induction on the number of edges in the graphs, i.e by constructing graphs G_R and G_S gradually, adding one edge at a time to G_R and observing its effect on the construction of G_S . The procedure for constructing the two graphs is as follows: for each clause in the formula we add a vertex in G_R , then we scan the clause and for each of its literals we add a vertex to G_S if it does not already appear in G_S along with its loop and edges connecting it to other vertices already in G_S . We then check if the complement of this literal appears in another clause, if so we add an edge to G_R connecting these two clauses.

The observation is true for the trivial case when G_R contains no edges. In that case G_S will have at least one edge which is a loop placed on one of the vertices representing a literal appearing in some clause in G_R . Assume the observation is true for n edges appearing in G_R , i.e. $|E| = n \leq |E'|$. Now we add an edge to G_R which will then contain $n + 1$ edges. There are three possible cases to consider. This edge may connect two vertices in G_R which are not connected to any other vertices in G_R , or the edge may be incident to one vertex in G_R which is already connected to another vertex and to another that is not connected to any other of the vertices in G_R , or the edge connects two vertices in G_R which are already connected to other vertices. In the first case, the two vertices in G_R that are now connected by an edge correspond to clauses that are different than any of the clauses already in G_R or else they would have already been in G_R , therefore they must differ from all other clauses by at least one literal. These literals must therefore be added to G_S along with their loops. These added loops (at least 2) to G_S make up for the edge just added to G_R . Moreover, if each clause contains more than one literal not appearing in other clauses already in G_R , these also will be connected by an edge in G_S as long as they do not complement each other, and to all other vertices already in G_S . In the second case one vertex exists in G_R but the other does not. This vertex corresponds to a clause that is different than all others already in G_R and therefore must contain at least one literal that is not in G_S . By adding this literal along with its loop we make up for the edge just added to G_R . Moreover, as in the previous case this new vertex must be connect to all other vertices already appearing in G_S . In the last case an edge is added to two vertices already connected to other vertices in G_R , this means based on the procedure for constructing the graphs, that there is literal appearing in one of the two clauses represented by these vertices that was not already in G_S . So add this literal as a vertex to G_S along with its loop and connections to other vertices in G_S , and it makes up for the edge added to G_R .

Theorem 1. *A formula ϕ is satisfiable if and only if:*

1. *There exists a homomorphism H from graph G_R to graph G_S .*

2. $H \subseteq T$.

Proof: (\rightarrow) If there exists a graph homomorphism H from graph G_R into graph G_S such that $H \subseteq T$ then each clause in ϕ is mapped to a literal appearing in it. By assigning true to that literal each clause can then be satisfied. To show that the clauses are mutually satisfiable we need to show that no two clauses are mapped to a literal and its complement. But if two clauses are mapped to two complementing literals then these clauses would have been connected in G_R by an edge, however if that was the case then the structure preserving mapping H must express this edge in G_S by connecting the two complementing literals. By the definition of G_S such an edge cannot be present. Therefore no two clauses are mapped to a literal and its complement.

(\leftarrow) If formula ϕ is satisfiable then each clause can be mapped to at least one literal (assigned true) appearing in it, and no two clauses are assigned to a literal and its complement. Let H express this mapping. We need to show that H is structure preserving, i.e. that every pair of clauses that are connected by an edge in G_R are mapped by H to a pair of literals that are also connected by an edge in G_S . For the formula to be satisfiable, an edge in G_R connecting two clauses implies that these clauses are mapped to two non-complementing literals. But by the definition of G_S every non-complementing pair of literals are connected by an edge. If the clauses were mapped to the same literal, this would have according to the definition of G_S been expressed by a loop placed on that literal. Therefore G_H is structure preserving.

Definition 13 (SAT revisited).

Input: $\langle R, S, T \rangle$.

Question: Does there exist a graph homomorphism H from G_R to G_S , such that $H \subseteq T$?

For examples see figures 2 and 3.

2.4 Tree-search for Subgraph Homomorphism

Solving graph homomorphism can be viewed as a tree search. If G is the pattern graph and G' is the target graph, then a graph homomorphism H can be viewed as an $n \times m$ boolean matrix, satisfying the three conditions given in definition 5, and where n and m are the number of vertices in G and G' respectively. Such a matrix will have exactly one 1 in each row, although it may have more than one 1 in each column. Candidates for the homomorphism are then the leaves of the following n - depth m - ary tree. In our case we are interested in finding one homomorphism, which can best be solved by performing a depth-first search on this tree, returning the first matrix satisfying the three

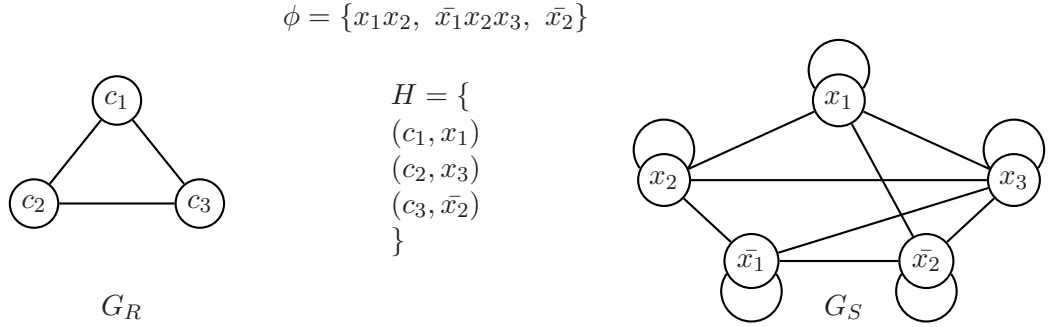


Figure 2: The satisfying assignment can be read from H , $x_1 = 1$, $x_2 = 0$, $x_3 = 1$. Note that $H = \{(c_1, \bar{x}_1), (c_2, \bar{x}_2), (c_3, x_3)\}$ is also a homomorphism satisfying condition 1 of theorem 1, but not 2 ($H \subseteq T$).

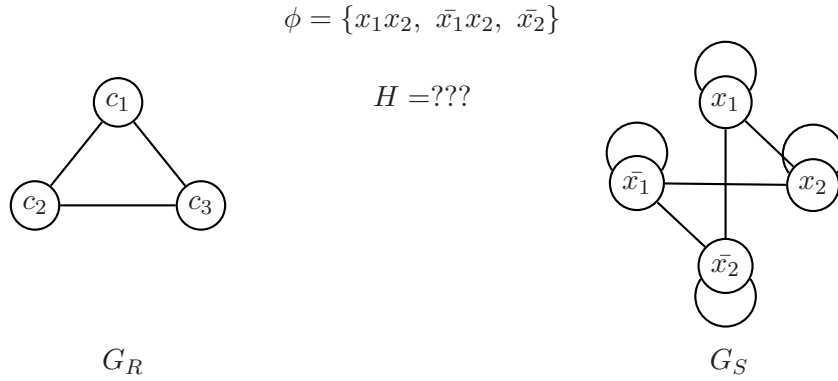


Figure 3: here condition 1 of theorem 1 is violated (no homomorphism exists), indicating that ϕ is unsatisfiable.

conditions. The search for a homomorphism can be improved in various ways. Some entries in the start matrix can be set to 0 if the corresponding mapping of the pattern vertex to a target one is forbidden. In our case the start matrix will be the matrix representing relation T from definition 10, which encompasses the restriction that a clause can only be paired with a literal appearing in it. Another improvement is based on the fact that the image of a vertex which is adjacent to some other vertex has to be adjacent to the image of the other vertex, thus a branch containing a mapping that violates this condition can be pruned. Another improvement is the pruning of nodes that contain an inconsistency, for example if clause c_i is mapped to literal l_j then no other clause cannot be mapped to \bar{l}_j . In [VM97] the authors present significant performance improvements for certain graphs, by reducing the problem of finding a homo-

morphic image of one graph in another to the problem of finding homomorphic images of every connected component of one graph in the other. An illustration of tree-search for the SAT instance and related graphs of figure 3, is sketched in figure 4.

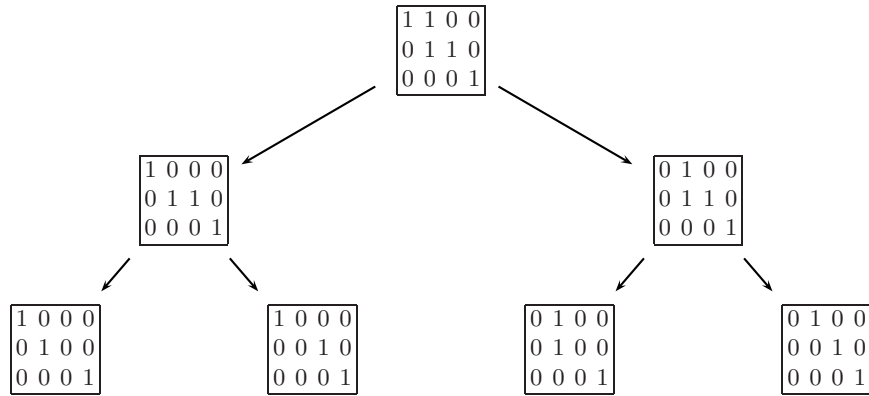


Figure 4: Tree-search for the SAT instance and related graphs of figure 3. Because the instance is unsatisfiable none of the leaves are homomorphisms, they all violate the conditions of definition 5.

3 Graph Matching Using Spectral Graph Theory

Of all the deep connections between combinatorics and linear algebra, those in the field known as spectral graph theory are among the most mysterious. Upon first encounter, it may seem quite bizarre that eigenvalues and eigenvectors of matrices should have anything at all to say about graph properties. But when looked at in the right way, this connection turns out to be quite natural and powerful.

As mentioned earlier the main characteristic of spectral methods is that the eigenvalues of graph adjacency matrices are invariant with respect to node permutations. A direct consequence of this is that if two graphs are isomorphic then their adjacency matrices will have the same eigenvalues. Unfortunately the converse is not true, namely we cannot deduce from the equality of eigenvalues that two graphs are isomorphic. Nonetheless by pairing the eigenvalues with their corresponding eigenvectors useful information about a graph's structural properties may be revealed. Another alluring characteristic is the ability to map this structural information into lower-dimensional subspaces. These properties together with the fact that computing eigenvalues/vectors (which takes polynomial time) is far less expensive than other combinatorial methods has attracted researchers for the purpose of solving graph matching problems.

In this paper we propose a novel approach based on spectral methods for solving SAT, which we have shown can be reduced to the graph matching problem. We adopt some of the techniques developed by Kosinov and Caelli in [KC02] and Luo et al. in [BRH03], for solving inexact subgraph matching by projections of vertices into the eigenspaces of graphs. The main theme of these techniques is to pose graph matching not as a combinatorial matching problem but, rather as one of clustering common local relational structures between different graphs, by grouping vertices of different graphs which share similar relational properties.

3.1 Mathematical Foundations

This section lays the mathematical foundations that lead the main theorem that the eigenvalues of graph adjacency matrices are invariant with respect to node permutations, and to the eigenspace projection methods used later on.

Definition 14 (Eigenvalues, Eigenvectors). *Let A be an $n \times n$ matrix. The scalar λ is called an **eigenvalue** (proper value) of A if there is a nonzero vector \mathbf{x} such that*

$$A\mathbf{x} = \lambda\mathbf{x}$$

*the vector \mathbf{x} is called an **eigenvector** of A corresponding to λ .*

To find the eigenvalues and eigenvectors for an $n \times n$ matrix A , we write the equation $A\mathbf{x} = \lambda\mathbf{x}$ in the form $\lambda I\mathbf{x} = A\mathbf{x}$, which then produces

$$(\lambda I - A)\mathbf{x} = \mathbf{0}$$

This homogeneous system of equations has nonzero solutions if and only if $(\lambda I - A)$ is not invertible, that is, if and only if $\det(\lambda I - A) = 0$.

Theorem 2. *Let A be an $n \times n$ matrix.*

1. *An eigenvalue of A is a scalar λ such that $\det(\lambda I - A) = 0$.*
2. *The eigenvectors of A corresponding to λ are the nonzero solutions of $(\lambda I - A)\mathbf{x} = \mathbf{0}$.*

The equation $\det(\lambda I - A) = 0$ is called the **characteristic equation** of A , and when expanded to polynomial form, the polynomial

$$|\lambda I - A| = \lambda^n + c_{n-1}\lambda^{n-1} + \dots + c_1\lambda + c_0$$

is called the **characteristic polynomial** of A . Because the characteristic polynomial of A is of degree n , A can have at most n distinct eigenvalues.

Theorem 3. *Let A be an $n \times n$ matrix. If A has a zero eigenvalue then A has linearly dependent rows and columns, its determinant is zero, and therefore A is singular.*

Theorem 4. *Let A be an $n \times n$ matrix. The sum of the n eigenvalues of A equals the sum of the n diagonal entries, which is known as the **trace** of A . Furthermore, the product of the n eigenvalues equals the determinant of A .*

Definition 15 (Coordinate Representation relative to a Basis). *Let $B = \{v_1, v_2, \dots, v_n\}$ be an ordered basis for a vector space V and let \mathbf{x} be a vector in V such that*

$$\mathbf{x} = c_1v_1 + c_2v_2 \dots + c_nv_n$$

*The scalars c_1, c_2, \dots, c_n are called the **coordinates of \mathbf{x} relative to the basis B** . The **coordinate vector** of \mathbf{x} relative to B is*

$$[\mathbf{x}]_B = \begin{bmatrix} c_1 \\ c_2 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{bmatrix}$$

Definition 16 (Transition Matrix). The matrix P is called the transition matrix from basis B to basis B' if multiplication of P by a coordinate vector of \mathbf{x} relative to basis B gives the coordinate vector of \mathbf{x} relative to B' , that is

$$P[\mathbf{x}]_B = [\mathbf{x}]_{B'}$$

Theorem 5. If P is the transition matrix from basis B to basis B' , then P is invertible and the transition matrix from B' to B is given by P^{-1} .

Definition 17 (Similar Matrices). For square matrices A and B of order n , B is said to be *similar* to A if there exists an invertible matrix P such that

$$B = P^{-1}AP$$

The next definition relates similar matrices to adjacency matrices of graphs and to the concept of graph isomorphism.

Definition 18. If A and B are adjacency matrices which correspond to two different labelings of the same graph G , then for some permutation matrix P

$$B = P^{-1}AP$$

Thus, isomorphic graphs have similar adjacency matrices.

Theorem 6 (Similar Matrices Have the Same Eigenvalues). If A and B are similar $n \times n$ matrices, then they have the same eigenvalues. Moreover an eigenvector \mathbf{x} of A corresponds to an eigenvector $P^{-1}\mathbf{x}$ of B .

Proof: since $A = PBP^{-1}$

$$A\mathbf{x} = \lambda\mathbf{x} \Rightarrow PBP^{-1}\mathbf{x} = \lambda\mathbf{x} \Rightarrow B(P^{-1}\mathbf{x}) = \lambda(P^{-1}\mathbf{x})$$

So the eigenvalue of B is still λ , and the eigenvector has been multiplied by P^{-1} .

Definition 19 (Diagonalizable Matrix). An $n \times n$ matrix A is *diagonalizable* if A is similar to a diagonal matrix D . That is, A is diagonalizable if there exists an invertible matrix P such that

$$D = P^{-1}AP$$

Theorem 7 (condition for diagonalization). An $n \times n$ matrix A is diagonalizable if and only if it has n linearly independent eigenvectors.

Theorem 8. If an $n \times n$ matrix A has n distinct eigenvalues, then the corresponding eigenvectors are linearly independent and therefore A is diagonalizable.

Theorem 9. *Suppose an $n \times n$ matrix A has n linearly independent eigenvectors. Then if these vectors are chosen to be the columns of a matrix V , it follows that $V^{-1}AV$ is a diagonal matrix D , with the eigenvalues of A along its diagonal.*

Proof: put the eigenvectors of A as the columns of V and compute the product AV one column at a time:

$$AV = A[v_1 : v_2 : \dots : v_n] = [\lambda_1 v_1 : \lambda_2 v_2 : \dots : \lambda_n v_n] = VD$$

Rearranging the equation $AV = VD$ we get

$$A = VDV^{-1}$$

which is commonly referred to as the **eigendecomposition** of a matrix.

Since SAT is formulated in terms of undirected graphs, i.e. symmetric adjacency matrices the following theorem is the means by which spectral methods can be applied to our graph matching formulation of SAT.

Theorem 10 (Real Spectral Theorem). *if A is an $n \times n$ symmetric matrix, then*

1. *A is diagonalizable.*
2. *All eigenvalues of A called the **spectra** of A are real.*

Definition 20 (Orthogonal Matrix). *A square matrix P is called **orthogonal** if it is invertible and*

$$P^{-1} = P^T$$

Definition 21. *An $n \times n$ matrix P is orthogonal if its column vectors form an orthonormal set.*

Definition 22 (Orthogonally Diagonalizable). *A matrix A is orthogonally diagonalizable if there exists an orthogonal matrix P such that $P^{-1}AP = D$.*

Theorem 11 (Fundamental Theorem of Symmetric Matrices). *Let A be an $n \times n$ matrix. Then A is orthogonally diagonalizable and has real eigenvalues if and only if A is symmetric.*

By the last theorem we know that our adjacency matrices can be eigendecomposable into $A = VDV^{-1} = VDV^T$, where V is a set of orthonormal eigenvectors and D is a diagonal real matrix containing A 's eigenvalues.

Up to now we only dealt with adjacency matrices, other matrices which are also consistent with the idea of graph eigenvalue invariance, and which have been proven to be more versatile, i.e. able to deal more types of graphs, are the Laplacian and Normal matrices [Chu97, CDS80].

Definition 23 (Laplacian Matrix). Let d_v denote the degree of vertex v . The Laplacian matrix of graph G denoted $L(G)$ is defined as follows

$$L(u, v) = \begin{cases} d_v & \text{if } u = v, \\ -1 & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 24 (Normal Matrix). Let d_v denote the degree of vertex v . The Normal matrix of graph G denoted $N(G)$ is defined as follows

$$N(u, v) = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0, \\ -\frac{1}{\sqrt{d_v d_u}} & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

Equivalently definitions of the Laplacian and Normal are

$$L = D - A$$

$$N = D^{-1/2} A D^{-1/2}$$

where D is the diagonal matrix of vertex degrees and A the regular adjacency matrix.

Similar to vector norms which are used to measure the size of a vector, matrix norms are used to measure the size matrices. Although there are several different matrix norms, the one we will use is called the **Frobenius** norm, and also referred to as the **Euclidean** norm.

Definition 25 (Euclidean Norm). The Euclidean norm of an $n \times n$ matrix A denoted $\|A\|_F$ is defined as the square root of the sum of the absolute squares of its elements

$$\|A\|_F = \left(\sum_{i=1}^n \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2}$$

3.2 Eigenspectra and Eigenvectors of Graphs

Spectral methods for graph matching heavily rely on eigendecomposition of adjacency matrices, which as a result of the *fundamental theorem of symmetric matrices* (theorem 11) is expressed as follows:

$$A = V D V^T$$

where A is a square symmetric adjacency matrix of a graph, with $a_{ij} = 1$ if there exists an edge that connects vertex i with j , and $a_{ij} = 0$ otherwise; V is an orthogonal matrix whose columns are normalized eigenvectors of A , and D is a diagonal matrix containing the eigenvalues λ_i of matrix A . These values, as mentioned before are called the spectra/spectrum of A , hence the name spectral methods. It is also customary to replace A the adjacency matrix, by the closely related Laplacian L or the Normal matrices for the reasons outlined in the previous section.

As stated earlier the main property of spectral methods is that the spectra of a matrix is invariant under vertex permutations/transformations. That is, two isomorphic graphs represented by matrices A and B , and related by $B = P^{-1}AB$, where P is perceived as the permutation matrix, will have the same eigenvalues (theorem 6). Nonetheless it was proved that the spectra of graphs is not unique, i.e. two graphs cannot be distinguished by relying exclusively on their spectra. Furthermore it was shown that as the number of vertices gets large, the probability of occurrence of a non-isomorphic co-spectral sub-graph pair in any two graphs being compared, asymptotically reaches unity. These results suggest that pure spectral methods based solely on eigenvalues are generally not rich enough to fully represent graph structure variability.

In [KC02] the authors propose to overcome this lack of uniqueness by using graph spectra coupled with the associated eigenvectors, or even by relying solely on the eigenvectors. Another drawback usually attributed to spectral methods is that they are not extendible to matching graphs of different sizes, such the method proposed in [Ume88] which was among the pioneering works in this field. This problem, according to the same authors can be eliminated by applying normalization and projection operations, presented in the next section. In [BRH03] the authors propose the use of other spectral features such as eigenmode perimeter, eigenmode volume, Cheeger number, inter-mode adjacency matrices, and inter-mode edge distance, that provide a compact summary of graph structure and used to overcome the same problems. These methods will be outlined later as an alternative to the solutions provided by Kosinov and Caelli.

3.3 Normalization and Projections

Subspace projection methods, in the principle component analysis literature, are generally used to reduce the dimensionality of data, while minimizing the information loss due to the decreased number of dimensions. Formally if A is the dataset covariance matrix, then using eigendecomposition A is first decomposed into:

$$A = VDVT^T$$

where V is a matrix of eigenvectors called the *principle components*, and D a diagonal matrix of eigenvalues. The original data is then projected onto a smaller number of most important principle components, that are associated with the largest eigenvalues as follows:

$$\hat{x} = V_k^T x$$

where \hat{x} is the projection, V_k^T is the transpose matrix of k principle components, and x is an item of the original data.

By taking the same approach, we can project vertex connectivity data from a graph adjacency matrix onto a smaller set of its most important eigenvectors. The projection obtained would then represent the relational properties of individual vertices relative to others in the lower dimensional eigenspace of a given graph. In this eigenvector subspace, structurally similar vertices or vertex groups will be located close to each other, which can then be used for approximate comparison and matching of graphs.

The number of dimensions to choose for vertex projections of a given pair of graphs, that is, the k is most important eigenvectors, is the smaller value of the ranks of the adjacency matrices of the two graphs being compared, i.e.

$$k = \min(\text{rank}(A_1), \text{rank}(A_2))$$

To ensure the comparability of the projections for graphs of different size, i.e. to be able to deal with graphs of different size, the authors state that empirical evidence suggests that an extra step of renormalization of the projections is necessary. The idea is that for the purpose of comparing two graphs we should not consider the values of the projections as is, but should look at how they are positioned and oriented relative to each other in their own eigenvector subspace. To do that we disregard the magnitudes of the projections by normalizing them and pay attention only to the orientation. I.e. the renormalized eigenvectors and eigenvalues become:

$$V'_k = \frac{V_k}{\|V_k\|}$$

$$D'_k = \frac{D_k}{\|D_k\|}$$

where $\|V_k\|$ and $\|D_k\|$ are the Euclidean norms of matrices V_k and D_k respectively. The renormalized subspace projection of each vertex in A , i.e the projections of the vertices onto the renormalized eigensubspace can then be obtained by computing:

$$D'_k (V'_k)^T$$

If A was an $n \times n$ adjacency matrix and $(V'_k)^T$ a $k \times n$ matrix, then the result will be a $k \times n$ matrix whose columns are the projections of the vertices into the renormalized eigensubspace.

In addition we should also carry a dominant sign correction of the projection coordinates of either of the two graphs so as to align one set of vertex projections against the other. That is, for each eigenvector, the number of its positive and negative components is counted. The eigenvector is then multiplied by -1 if the number of negative components is greater than the number of positive components, or left as is otherwise. This corresponds to setting the direction of the axes in such a way to result in the most compatible alignment between vertex data using the dominant sign test.

An illustration of these propositions is provided in figure 5. Two graphs X and Y Although different in size, are nevertheless quite similar to each other. In fact, one may see graph Y as an enlarged version of graph X . The result of projecting the two graphs into the normalized 2D eigenvector subspace shown on the right demonstrates the following two important features of the proposed method: firstly, the projections of vertices of both graphs follow a similar pattern, which means that it is possible to determine overall structural similarity of graphs with different number of vertices, and secondly, one may also see (by examining the juxtaposition of the projected vertices of both graphs) that graph vertices with similar relational properties tend to get projected into the areas that are close to each other. These properties are quite valuable, and, as such, have the potential to prove useful in solving the graph matching problem.

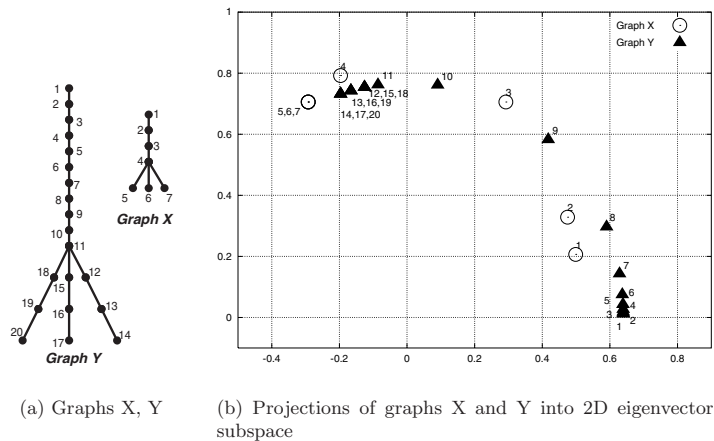


Figure 5: Two graphs and their projections, which follow a similar pattern.

3.4 Clustering of Projections

The only remaining step for solving the graph matching problem is to find the correspondence among the vertices that have similar relational properties, and the overall similarity between the graphs. The main advantage of using clustering to solve this problem is that it can equally well discover correspondence relationships of various types, i.e. it is not limited to finding the best one-to-one matches of vertices from one graph to the other, but it can also identify the whole sub-graphs and vertex groups that possess similar structural properties, which is very important if the two graphs have substantially different sizes, or when we are seeking to find homomorphisms rather than monomorphisms such as with our reduction of SAT to the subgraph homomorphism problem.

In order to realize vertex correspondence the authors suggest to deploy a standard agglomerative hierarchical clustering routine. Agglomerative hierarchical clustering is a bottom-up clustering method where clusters have sub-clusters, which in turn have sub-clusters, etc. The clustering starts with every single object in a single cluster. Then, in each successive iteration, it agglomerates (merges) the closest pair of clusters by satisfying some similarity criteria, until all of the data is in one cluster, or no clusters can be merged any more because of large dissimilarity measures. A generic algorithm describing the process of this clustering method is outlined in figure 6. To make the algorithm more sensitive to the two graphs involved in the clustering an extra binary dimension $\{0, d\}$ is introduced into the projection data, whereon a given vertex pair has a difference of 0 if they belong to the same graph and d if they belong to different graphs. When all of the vertex projections are placed in their appropriate hyperplanes (defined by the values of 0 and d on an additional axis), a small distance (close to zero) between two projections will indicate structural similarity of a pair of vertices that come from the same graph, while a distance that is only slightly different from d will signal structural similarity of a pair of vertices that belong to different graphs. d should always be larger than the maximum vertex projection distance for both graphs. If the renormalization step is carried, then $d = 2$ will be sufficient since all of the projection coordinates will lie in the $[-1, 1]$ range.

Figure 7 demonstrates the result of vertex projection clustering of two sample graphs. It recovers a natural correspondence among the groups of vertices in these two graphs. The linkages that were established between vertices during the clustering phase are shown as straight lines connecting the projected graph vertices. Figures 8, 9, and 10 demonstrate the result of vertex projections of SAT instances formulated as pairs of graphs. Note that for the satisfiable in-

procedure *AgglomerativeHierarchicalClustering*(*Dataset : D*)
 assign each $x \in D$ to a separate cluster.
 evaluate all pair-wise distances between clusters.
 construct a distance matrix using the distance values.
 look for the pair of clusters with the shortest distance.
 remove the pair from the matrix and merge them.
 evaluate all distances from this new cluster to all other clusters, and update the matrix.
 repeat until the distance matrix is reduced to a single element, or until no clusters can be merged.
end.

Figure 6: A generic outline of an Agglomerative Hierarchical Clustering algorithm .

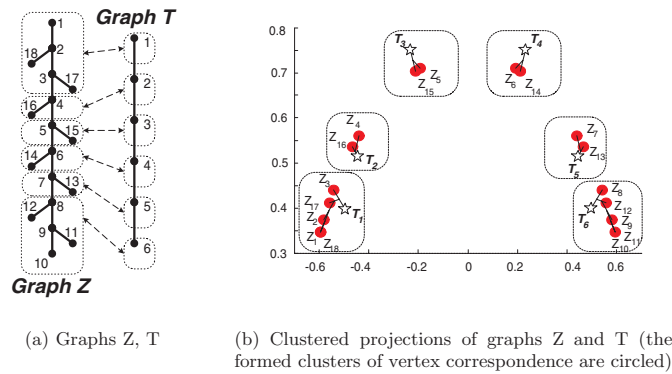


Figure 7: Clustering of vertex projections of two similar graphs.

stances the clauses and their homomorphic literal images are projected into the same clusters, whereas for the unsatisfiable instances no correspondence can be found within the clusters between clauses and literals. That is, the clauses and their potential matches are projected far from each other. This leads to the hypothesis that the overall distance between the projections of clauses and the literals appearing in them is likely to be larger for unsatisfiable instances than for satisfiable instances. Moreover it seems that the correspondence between clauses and literals for satisfiable instances can be recovered by searching for each cluster the projection of the closest literal appearing in it.

Once the correspondence clustering decomposition has been obtained, the only remaining task is to provide a measure of how similar the two graphs being compared are to each other. The authors of [KC02] propose a similarity measure taken from the data mining literature, but other similarity measures,

$$\phi = \{x_1x_2, \bar{x}_1x_2x_3, \bar{x}_2\}$$

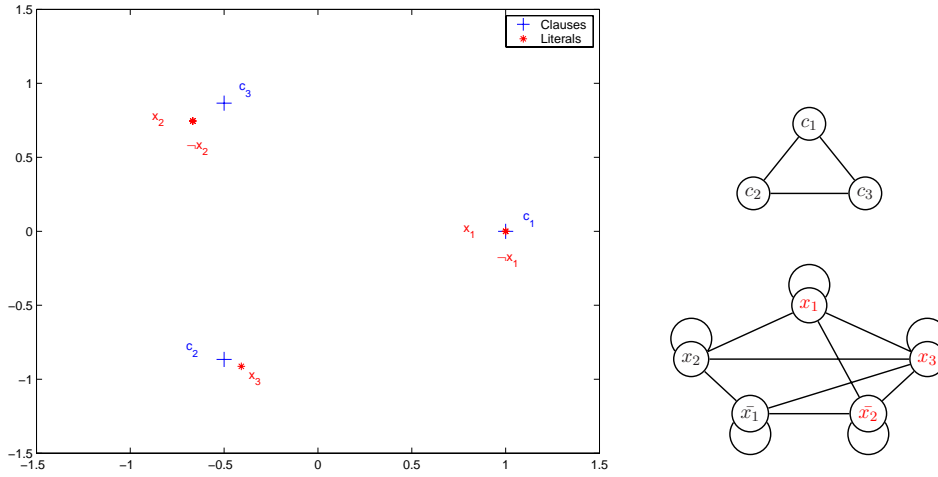


Figure 8: Projection of a satisfiable instance, where the two graphs on the right correspond to the clause and literals graphs of the SAT instance.

$$\phi = \{x_1x_2, \bar{x}_1x_2, \bar{x}_2\}$$

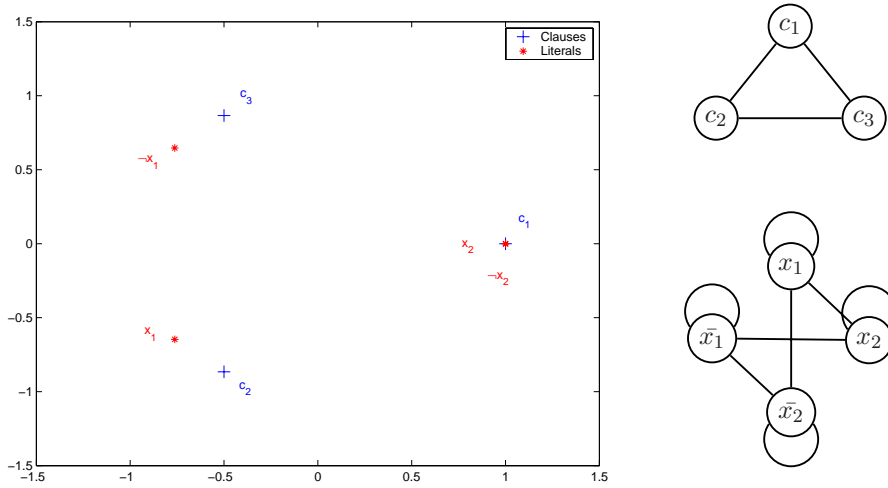


Figure 9: Projection of an unsatisfiable instance.

$$\phi = \{x_1x_5, \bar{x}_1x_2x_4, \bar{x}_1x_2x_3, \bar{x}_1x_4, \bar{x}_1, \bar{x}_2\bar{x}_4, \bar{x}_3\}$$

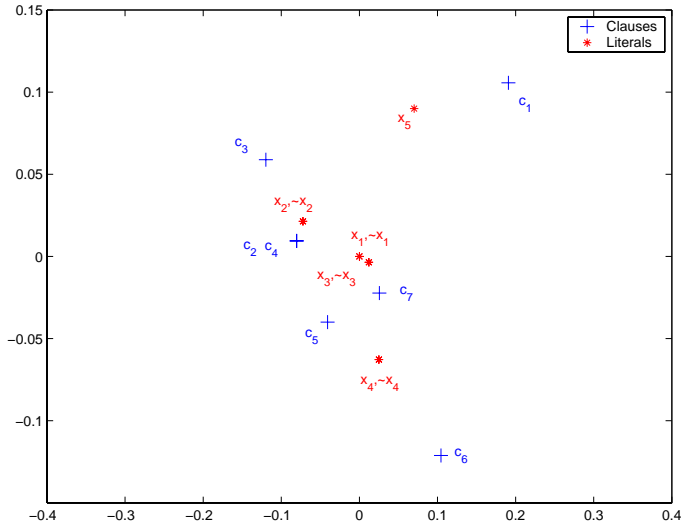


Figure 10: Projection of another satisfiable instance.

some of which are more suitable for our case are available.

4 Similarity Measures

The problem of similarity measures between structured objects has been studied in the domain of structural pattern recognition and pattern analysis. In general, similarity measures can be grouped into three main classes. *Feature-based distances*, where a set of features is extracted from the structural representation of the objects, and used as n -dimensional vectors that can be compared by the Euclidian distance. *Cost-bases distances*, where the distance between two objects is measured by the number of modifications required in order to transform one object into the other. *Distribution based distances*, where the distribution of elements of the two objects are compared.

In general the concept of similarity or dissimilarity is expressed by means of a distance function $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{R}$, where \mathcal{D} is a domain. If $x \in \mathcal{D}$ and $y \in \mathcal{D}$ are two objects, then the similarity between them is usually expressed as $d(x, y)$. Very often the function d is a metric, i.e., it satisfies fundamental metric space properties; non-negativity, symmetry, uniqueness, and triangular inequality.

4.1 L_p Distances, Minkowski Distances

Many similarity measures are based on the L_p distance between two points. For two points $x, y \in \mathcal{R}^k$, the L_p distance is defined as

$$L_p(x, y) = \left(\sum_{i=0}^k |x_i - y_i|^p \right)^{1/p}$$

This is often called the Minkowski distance. For $p = 2$, this yields the Euclidian distance L_2 , and for $p = 1$, we get the Manhattan (city block) distance L_1 [VH00].

We propose an extension of the L_2 similarity measure which is tailored to our problem. Recall from definition 10 that relation T where $T = \{(a, l) \mid a \in A \text{ and } l \in a\}$ and A is the set of clauses, is the set of all possible pairs of clauses and literals appearing in the formula, with the restriction that a clause can only be paired with a literal appearing in it. Let a' be the projection of the vertex corresponding to clause a into the eigenspace of the constraint graph as described in the previous section, and similarly l' the projection of the vertex corresponding to literal l into the eigenspace of the consistency graph. For all $(a, l) \in T$ we first compute $L_2(a', l')$. Let A and B be the constraint and consistency graphs respectively, then our first measure of similarity $d_1(A, B)$ between graphs A and B is computed as follows

$$d_1(A, B) = \left(\frac{1}{|\phi|} \sum_{a \in \phi} \sum_{l \in a} \frac{L_2(a', l')}{|a|} \right)$$

where $|a|$ is the number of literal appearing in clause a , and $|\phi|$ is the number of clauses appearing in formula ϕ . Thus we are first finding the mean of the distances between each clause and its potential mapping, and then the mean of the sum of these distances. Another measure we will be interested in recording is the distance of a mapping (assuming the formula is satisfiable and the mapping is known) expressed by relation H , where H is a set of pairs (a, l) , a being a clause and l its matched literal. Then d_2 is computed as follows

$$d_2(A, B) = \left(\frac{1}{|H|} \sum_{(a, l) \in H} L_2(a', l') \right)$$

where $|H|$ is the number of pairs in H .

Both distance measures d_1 and d_2 take linear time in the size of the formula if the projections are already computed.

4.2 A Data Mining Similarity Measure

If an agglomerative hierarchical clustering routine such as the one discussed in section 3.4 is executed, then the authors of [KC02] propose to use the following similarity measure:

$$d(G_1, G_2) = \alpha_1 \cdot d_{DB} + \alpha_2 \cdot d_{set}$$

$$d_{DB} = \frac{1}{M} \sum_{i=1}^M \max_{j=1 \dots M, j \neq i} d_{ij}, \quad \text{where } d_{ij} = \frac{\mu_i + \mu_j}{L_2(c_i, c_j)}$$

$$d_{set} = \frac{\sum_{i=1}^M (1 - p_i)}{\text{rank}(A_{G_1}) + \text{rank}(A_{G_2})}, \quad \text{where } p_i = \frac{|V_i(G_1) - V_i(G_2)|}{V_i(G_1) + V_i(G_2)}$$

The first component d_{DB} in the graph similarity formula is the Davis-Bouldin *cluster validity index*. Where M is the number of vertex clusters, μ_i is the average distance of all vertex projections in cluster i to the center c_i , and $L_2(c_i, c_j)$ is the distance between cluster centers c_i and c_j . The index ranges over the interval $[0, \infty]$ with smaller values corresponding to coherent clustering. The second component d_{set} is an estimate derived from the set similarity formula, that encourages a greater number of correspondence clusters, while, at the same time, introducing a penalty factor p_i , dependent upon the degree of how unbalanced cluster i is. In this formula $V_i(G_1)$ and $V_i(G_2)$ are the number of vertices from graph one and two, respectively, found in the i -th cluster, and A_{G_1} , A_{G_2} are the adjacency matrices of the two graphs being matched. The authors found that the best values for α_1 and α_2 to be 2.5 and 1 respectively.

4.3 Distribution Based Similarity Measures

Distribution based similarity measures can be considered as a kind of distance measure between probability densities, although some of these measures cannot be considered real distance measures because they are not symmetric.

4.3.1 χ^2 Similarity Measure

The χ^2 test is generally used to test if a sample of data came from a population with a specific distribution. We adopt the χ^2 to test if the projections of vertices from two graphs onto an eigenspace follow a similar distribution. That is, test whether there is no significant difference between the proportions of points in each cluster for each graph. The calculated χ^2 statistic will then be

our measure of similarity between the two graphs. The greater its value is, the more dissimilar the graphs are. The calculation of the χ^2 statistic proceeds as follows: let x_{ij} be number of points belonging to cluster i from graph j , where $i \in \{1 \dots n\}$ (n is the number of clusters), $j \in \{1, 2\}$ (for the two graphs).

1. calculate the total number of points in each cluster,

$$total_i = \sum_{j=1}^2 x_{i,j}$$

2. calculate the total number of points in each graph,

$$total_j = \sum_{i=1}^n x_{i,j}$$

3. calculate the total number of points,

$$Total = \sum_{j=1}^2 total_j$$

4. calculate the proportion of points in each cluster,

$$P_i = \frac{total_i}{Total}$$

5. calculate the predicted proportion of points of each graph in each cluster,

$$f_{i,j} = P_i \cdot total_j$$

6. calculate the value of the χ^2 statistic S , which should follow a χ^2 distribution with $2n$ degrees of freedom.

$$S = \sum_{i=1}^n \sum_{j=1}^2 \frac{(x_{ij} - f_{ij})^2}{f_{ij}}$$

4.3.2 Relative Entropy/ Kullback-Leibler Distance

Suppose we have a discrete set of symbols $\{a_1, a_2, \dots, a_n\}$ with associated probabilities $p(a_i)$. The *information* of selecting a certain element a_i from the set of symbols, which is elected with probability $p(a_i)$ is defined as

$$I(a_i) = \log \frac{1}{p(a_i)} = -\log p(a_i)$$

and used to express the level of uncertainty or measure of surprise in selecting a_i . The *entropy* of a given discrete distribution, which expresses the average amount of information per symbol or the level of uncertainty in the selection of a symbol from the set is defined as

$$H = \sum_{i=1}^n p(a_i) \cdot I(a_i) = - \sum_{i=1}^n p(a_i) \log p(a_i)$$

Suppose we now have two distributions $p(a_i), q(a_i)$ over the same set of symbols, the *relative entropy* or the *Kullback-Leibler distance* is a measure of discrepancy between the two distributions. It and is defined as

$$D_{KL}(p, q) = \sum_{i=1}^n q(a_i) \log \frac{q(a_i)}{p(a_i)}$$

Note that this measure is not symmetric, i.e. $D_{KL}(p, q) \neq D_{KL}(q, p)$. We adopt the Kullback-Leibler distance as measure of similarity between our graphs as follows. Let the set of symbols be the clusters $\{c_1, c_2, \dots, c_n\}$, the probabilities associated with each cluster $p(c_i), q(c_i)$ are the proportion of points from each graph in cluster i . For example if cluster i has 10 points from graph 1, and 5 points from graph 2, where the total number of points in each graph is 100 and 40 respectively, then $p(c_i) = 0.1$ and $q(c_i) = 0.125$. To overcome the absence of symmetry with this measure we propose that the sum of the Kullback-Leibler distances in each direction be our measure of similarity between the graphs, i.e.

$$\begin{aligned} d(G_1, G_2) &= D_{KL}(p, q) + D_{KL}(q, p) = \sum_{i=1}^n q(c_i) \log \frac{q(c_i)}{p(c_i)} + \sum_{i=1}^n p(c_i) \log \frac{p(c_i)}{q(c_i)} \\ &= \sum_{i=1}^n q(c_i) \log \frac{q(c_i)}{p(c_i)} - p(c_i) \log \frac{q(c_i)}{p(c_i)} \\ &= \sum_{i=1}^n \log \frac{q(c_i)}{p(c_i)} (q(c_i) - p(c_i)) \end{aligned}$$

5 Other Spectral Features and Embeddings of Graphs

In the section we propose an alternative to the approach discussed thus far, which is based on the work of Luo et al. in [BRH03]. Their method is similar in nature to the one already discussed, where the main difference is the way the structure of graphs is mapped into a low dimensional eigenspace. More specifically, they use different features to characterize the structure of graphs

and embed them in a pattern space. Based on the eigendecomposition of the adjacency or related Laplacian $A = VDV^T$, and the extraction of the k most important principle components, the following features vectors are suggested.

5.1 Unary Features

5.1.1 Leading Eigenvalues

For graph G , its feature vector is

$$F_G = (\lambda_1, \lambda_2, \dots, \lambda_k)^T$$

That is the k leading eigenvalues, or the spectrum of G .

5.1.2 Volume

The volume of a graph is defined to be the sum of the degrees of the vertices belonging to the graph. If G is our graph, let $d(i)$ be the degree of vertex i , and

$$vol(j) = \sum_{i \in V(G)} V_k(i, j)d(i)$$

where $V_k(i, j)$ is the i -th component of eigenvector j , be the volume associated with eigenvector j . Then the volume feature vector for graph G is

$$F_G = (vol(1), vol(2), \dots, vol(k))^T$$

5.1.3 Perimeter

For a subgraph S of G the set of perimeter edges is

$$\Delta(S) = \{(u, v) | (u, v) \in E(G), u \in S, v \notin S\}$$

the perimeter length of S is defined to be the number of edges in the perimeter set, i.e. $\Gamma(S) = |\Delta(S)|$. By analogy the perimeter length of the adjacency matrix for eigenvector indexed x is

$$\Gamma(x) = \sum_{x \neq y} \sum_{i \in V(G)} \sum_{j \in V(G)} V_k(i, x)V_k(j, y)A(i, j)$$

The perimeter feature vector is then

$$F_G = (\Gamma(1), \Gamma(2), \dots, \Gamma(k))^T$$

5.1.4 Cheeger Constant

The Cheeger constant is used to solve isoperimetric problems for graphs. Isoperimetric problems examine optimal relations between the size of a cut the sizes of the separated parts. A typical isoperimetric problem is to remove as few edges of the graph as possible to separate out a subset of vertices of some desired “size”, usually disconnect the graph into two almost equal parts. The size of the separated parts is not measured by the number of vertices in each part, but takes into account the weight each vertex has, i.e. measure the volume of the parts. Determining the Cheeger constant of a graph is equivalent to solving the following problem: for a fixed number m , find a subset S with $m \leq \text{vol } S \leq \text{vol } \bar{S}$ such that $\Delta(S)$ contains as few edges as possible.

The Cheeger constant for a subgraph S of G is defined as

$$H(S) = \frac{|\Delta(S)|}{\min(\text{vol}(S), \text{vol}(\bar{S}))}$$

where \bar{S} is the complement of S . The Cheeger constant of a graph G is defined as

$$H(G) = \min_S H(S)$$

The analogue of the Cheeger constant for the i -th eigenvector of the adjacency matrix is

$$H(i) = \frac{\Gamma(i)}{\min(\text{vol}(i), \text{vol}(\bar{i}))}$$

where

$$\text{vol}(\bar{i}) = \sum_{i=1}^k \sum_{j \in V(G)} V_k(j, i) d(j) - \text{vol}(i)$$

The Cheeger constant feature vector is

$$F_G = (H(1), H(2), \dots, H(k))^T$$

5.2 Binary Features

5.2.1 Inter-mode Adjacency Matrix

The inter-mode adjacency matrix is found by projecting the adjacency matrix onto the basis spanned by the eigenvectors, which is computed by

$$U = V_k^T A V_k$$

The element of the matrix with row i and column j is given by

$$U(i, j) = \sum_{x \in V(G)} \sum_{y \in V(G)} V_k(x, i) V_k(y, j) A(x, y)$$

This matrix is then converted into a long vector by stacking the columns of U in eigenvalue order. The resulting feature vector is

$$F_G = (U(1, 1), U(1, 2), \dots, U(1, k), U(2, 1), \dots, U(2, k) \dots U(k, k))^T$$

5.2.2 Inter-mode Distances

The between mode distance is defined as the minimum number of edges, between the most significant vertices associated with each eigenvector of the adjacency matrix, which is the one having the largest co-efficient in the associated eigenvector. For eigenvector indexed i the most significant vertex is x_i

$$x_i = \arg \max_x V_k(x, i)$$

We know that $A^l(i, j)$ is the number of paths of length l edges between vertices i and j . Hence the minimum number of edges between the most significant vertices of eigenvectors u and v is

$$d_{u,v} = \arg \min_l A^l(x_u, x_v)$$

To describe a graph, the inter-mode distances between pairs of eigenvectors can be put into a $k \times k$ matrix which can be converted into a long feature vector

$$F_G = (d_{1,1}, d_{1,2}, \dots, d_{1,k}, d_{2,1}, \dots, d_{k,k})$$

5.3 Pattern Space Embedding

To embed our two graphs that are to be matched in an eigenspace, we perform principle component analysis on the covariance matrix for the spectral pattern vectors. In particular each of our graphs is vectorized in the way outlined above. Let F_1, F_2 be the two features vectors corresponding to graphs G_1 and G_2 respectively. We arrange them as the columns of matrix S , i.e.

$$S = [F_1 | F_2]$$

Next we compute the covariance matrix C for the the elements of the rows of S by

$$C = SS^T$$

We then extract the principle components for the data by performing in eigen-decomposition on the covariance matrix C ,

$$C = \Phi \Lambda \Phi^T$$

where Φ is the matrix of eigenvectors/principle components, and Λ is a diagonal matrix of eigenvalues. We use the first k leading eigenvectors to represent the graphs. The coordinate system of the eigenspace is spanned by the k orthogonal vectors

$$\Phi_k = [\phi_1 | \phi_2 | \dots | \phi_k]$$

The graphs represented by the feature vectors F_1, F_2 are then projected onto this eigenspace using

$$x_i = \Phi_k^T F_i$$

Thus graph G_i is represented by a k -dimensional vector x_i in the eigenspace. To measure the similarity of the two graphs we compute the Euclidian distance between x_1 and x_2 , i.e.

$$d(G_1, G_2) = \left(\sum_{i=1}^k |x_{1i} - x_{2i}|^2 \right)^{1/2}$$

6 Data That Needs to be Recorded

Before developing a SAT search strategy we propose to record and organize similarity measures of pairs of graphs representing a SAT instance in the following table:

	100%	75%	50%	25%	0%
$d_1(G, G')$					
$d_2(G, G')$					
\vdots					
$d_k(G, G')$					

where $d_i(G, G')$ is the i -th type of similarity measure, and the percentages in the header represent classes of SAT instances that out of the 2^n possible truth assignments, $x\%$ are satisfying assignments. Thus, the 0% class represents the class of unsatisfiable formulas, and the 100% class represents the valid formulas. Each cell in this table will contain a distribution and range of a certain type of similarity measure for a certain class of SAT instances. The goal is to be able

to model these distributions, and later use them as part of the search strategy, which is guided by the likelihood of a formula being satisfiable or unsatisfiable.

We will also record the similarity measure of mappings. That is, a similarity measure between the two sets of points each coming from a different graph that constitute the homomorphic mapping of vertices from one graph to the other. This similarity measure is different from the others in that it does not consider all the points but only those pairs of points that belong to the homomorphic mapping H . Its formulation was given in section 4.1, and the reason it is recorded is to have some indication of how far is a regular L_2 based similarity measure for two graphs from a possible mapping. Also it can be used as a stopping criteria for a clustering routine, i.e. by knowing the mean distance between a clause and its mapped literal in the projected space, we have an estimate of the diameter of the cluster, and thus know when to stop merging clusters.

7 A SAT search strategy

At each node of the search space DPLL based SAT algorithms select a literal according to some heuristic/search strategy, simplify the formula based on that selection, and deduce necessary assignments or conflicts. If a conflict has been encountered the algorithm must backtrack to a higher level to repair the assignments causing the inconsistency, otherwise the algorithm repeats the same process, i.e. selects another variable, and continues to search deeper levels of the search space. However, if all variables appearing in the formula have been assigned and no inconsistencies were encountered then the formula is satisfiable. If the whole search space was traversed without finding any satisfying assignments (the algorithm has nowhere to backtrack to) then the formula is unsatisfiable.

We propose to use the same approach, i.e. base our algorithm on the DPLL procedure, but offer a novel strategy for selecting variables at each node of the search space. Our hope is that this new strategy will lead as soon as possible to satisfying assignments in the case of a satisfiable instance, and will lead as soon as possible to conflicts in the case of unsatisfiable instances. The main advantage of this approach is that it is able to deal with both satisfiable and unsatisfiable instances with equal or better accuracy (in the selection of variables) than existing SAT strategies. Other advantages inherent to our approach will be revealed later.

7.1 The Algorithm

At the root node of the search space the formula is converted into a graph matching problem as discussed in section 2. Based on the measure of similarity between the *clause* and *consistency* graphs, computed by the methods discussed in the previous section, we can determine, hopefully with high accuracy whether the formula is likely to be satisfiable or not. In case it was determined that the formula is more likely to be satisfiable the algorithm enters into an *S* (satisfiable) mode, otherwise into a *U* (unsatisfiable) mode.

In the *S* mode the aim is to find as soon as possible a satisfying assignment by an accurate selection of satisfied literals. At each node of the search space a simple (computationally) heuristic such as randomly selecting literals or selecting the most frequent literal, is employed to select a candidate list of literals. For each of these literals the formula is simplified and converted into the corresponding clause and consistency graphs. By the methods discussed above a measure of similarity is computed for each pair of graphs. The literal corresponding to the pair with the highest measure of similarity is then chosen. The underlying assumption is that a formula is more likely to be satisfiable if its corresponding graphs are more likely to be matched. Alternatively a more accurate but computationally more expensive approach is to generate the candidate list by selecting literals of node pairs (from the two graphs) with the largest proximity in the clustered subspace, and proceed as before. The assumption here is that the pairs with the largest proximity are more likely to be part of the mapping/matching of the nodes of the two different graphs, and thus a safer choice. One of the most valuable properties of this approach is at any given time, i.e. at any node an approximation of the truth assignment can be generated by pairing the nodes of the two graphs in the clustered subspace. This approximation can be tested and if it turns out to be a satisfying assignment then we are done.

U mode ... to be continued

References

- [BRH03] B.Luo, R.C.Wilson, and E.R. Hancock, *Spectral embedding of graphs*, Pattern Recognition (2003), no. 36, 2213–2223.
- [CDS80] D.M. Cvetkovisc, M. Doob, and H. Sachs, *Spectra of graphs*, Academic Press, New York, 1980.
- [Chu97] F. R. K. Chung, *Spectral graph theory*, CBMS Regional Conf. Ser. Math, AMS, Providence RI, 1997.
- [CV00] Jordi Cortadella and Gabriel Valiente, *A relational view of subgraph isomorphism*, Proc. Fifth Int. Seminar on Relational Methods in Computer Science (Québec, Canada), 2000, pp. 45–54.
- [GJ79] M.R. Garey and D.S. Johnson, *Computers and intractability, a guide to the theory of np-completeness*, W.H.Freeman and Co., San Francisco, 1979.
- [Har78] R.M. Haralick, *The characterization of binary relation homomorphisms*, Int. J. general systems **4** (1978), 113–121.
- [HK78] R. Haralick and J. Kartus, *Arrangements, homomorphisms and discrete relaxation*, IEEE Trans. on Systems, Man, and Cybernetics. **8(8)** (1978), 600–612.
- [KC02] S. Kosinov and T. Caelli, *Inexact multisubgraph matching using graph eigenspace and clustering models*, Structural, Syntactic and Statistical Pattern Recognition, Lecture Notes in Computer Science, LNCS 2396 (M. Kamel D. de Ritter T. Caelli A. Amin, R. Duin, ed.), Springer, 2002, pp. 133–142.
- [LV02] Javier Larrosa and Gabriel Valiente, *Constraint satisfaction algorithms for graph pattern matching*, Mathematical Structures in Computer Science **12** (2002), no. 4, 403–422.
- [SS93] G. Schmidt and T. Strohleln, *Relations and graphs – discrete mathematics for computer science*, Springer-Verlag, 1993.
- [Ume88] S. Umeyama, *An eigendecomposition approach to weighted graph matching problems*, IEEE Trans. Pattern Anal. Mach. Intell. (1988), no. 10(5), 695–703.
- [VH00] Remco C. Veltkamp and Michiel Hagedoorn, *Shape similarity measures, properties, and constructions*, 2000.

- [VM97] Gabriel Valiente and Conrado Mart'inez, *An algorithm for graph pattern-matching*, In Proc. 4th South American Workshop on String Processing, volume 8 of International Informatics Series, Carleton University Press, 1997, pp. 180–197.
- [Vos92] G. Vosselman, *Relational matching, lecture notes in computer science, number 628*, Springer Verlag, Berlin, 1992.