Abstract Euler Diagram Isomorphism

Gem Stapleton
Visual Modelling Group
University of Brighton, UK
g.e.stapleton@brighton.ac.uk

Andrew Fish
Visual Modelling Group
University of Brighton, UK
andrew.fish@brighton.ac.uk

Peter Rodgers
Computing Laboratory
University of Kent, UK
p.j.rodgers@kent.ac.uk

Abstract

Euler diagrams are widely used for information visualization and form the basis of a variety of formal languages that are used to express constraints in computing. Tools to automatically generate and layout these diagrams from an abstract description have to overcome the fact that this problem is computationally difficult. We develop a theory of isomorphism of diagram descriptions and identify invariants of these descriptions under isomorphism. We can apply this theory to improve the efficiency of the generation of all abstract descriptions (up to isomorphism). We can also consider the production and use of libraries of diagrams with nice visual properties: by providing a normal form for the abstract descriptions we can improve efficiency of searches for isomorphic diagrams within such libraries and, moreover, utilize invariants for further efficiency savings. We produce an implementation of the theory and give an indication of the efficiency improvements.

1 Introduction

In this paper we develop the theory of isomorphism for Euler diagrams. Isomorphism of Euler diagrams is relevant in many applications where they are used. Examples of their application areas include any areas where representing relationships between collections of objects is helpful, such as [4, 9, 11, 13, 16, 22, 23, 24].

In computing, many modelling notations use closed curves as part of their syntax and can, therefore, be viewed as extending Euler diagrams. As an example, constraint diagrams [12] were designed for formal software specification as a potential replacement for the Object Constraint Language; see [10, 14] for examples of software modelling using constraint diagrams. A constraint diagram can be seen in figure 1.

The study of Euler diagram isomorphism is related to the study of other topics, such as graph isomorphism, where the study of optimizations for isomorphism checking are well

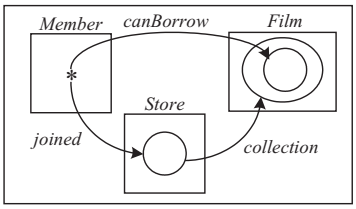

Figure 1. A constraint diagram.

advanced [15, 18, 19]. The abstract descriptions of Euler diagrams are equivalent to the abstract descriptions of hypergraphs [17], although the visualization method is different. This equivalence means that the techniques presented in this paper are applicable to hypergraphs.

After providing motivation (section 2) and background material (section 3) we develop the theory of diagram isomorphism, in section 4, and identify invariants of Euler diagrams in section 5. The problem of Euler diagram isomorphism checking is equivalent to hypergraph isomorphism checking. Applications of this theory include improving efficiency when generating abstract diagrams to populate a library, or searching for a diagram within such a library. We produce an implementation which performs isomorphism checking, making use of the invariants as optimisations. The amount of efficiency gain by using our methods is given by running the task of generating all abstract diagrams and comparing the effects of applying different invariants, detailed in section 6.

2 Motivation

In general, the generation problem is to find an Euler diagram that represents some specified collection of sets and possess certain properties, sometimes called wellformedness conditions. The input to an Euler diagram generation algorithm is an abstract description of the diagram to be generated. Various methods for generating Euler diagrams have been developed, each concentrating on a particular class of Euler diagrams, for example [2, 3, 6, 13, 25]. The

existing generation algorithms use some method for embedding curves in the plane. Diagram generation using these algorithms is a time consuming process and, moreover, can result in layouts which are not aesthetically pleasing and, therefore, potentially hard to read.

A previously unadopted generation approach is to have a library of nicely drawn diagrams from which an appropriate diagram is selected. In this context, it is helpful to have a notion of isomorphism amongst abstract descriptions in order to facilitate the extraction of an appropriate Euler diagram from the library. When using a library of example Euler diagrams, each token in the library would be marked with is abstract description. For example, suppose we wish to generate a diagram that expresses $A \cap B = \emptyset$ and $C \subseteq B$, like d_1 in figure 2. The diagram d_1 is 'isomorphic' to d_2, which expresses $C \cap A = \emptyset$ and $B \subseteq A$. If the library contained d_3 then we could select d_3 and insert labels in the appropriate way to yield either d_1 or d_2. To extract d_3 from the library would require us to establish that its abstract description is isomorphic to that for d_1.

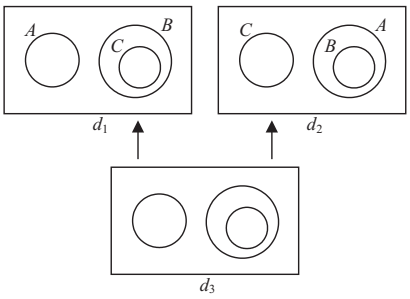

Figure 2. Using a library for generation.

We will need to be able to extract appropriate diagrams from the library in an efficient manner. Merely checking for isomorphism is time consuming and, if the library is large, will be infeasible in some cases. Thus, we require methods to partition the library of examples to reduce the number of checks that must be performed. When searching through a library, we can use invariants to subdivide the search space, reducing the number of brute-force checks to determine whether any given abstract description is isomorphic to another.

We can first partition the set of library diagrams by the number of labels they contain. Thus, we can break down the set of library examples into sets each of which contains all diagrams with a given number of labels. It is likely that this will be sufficient to provide an efficient search through the space when there are few labels. However, as the number of labels increases, the number of diagrams in the associated set can be very large. Thus, we could further subdivide these sets using more sophisticated isomorphism invariants. Obviously the order in which the subdivisions are performed can have an impact on the efficiency of the search through the library. In this paper, we identify a collection of invariants and in the future we will establish how best to use them for set-subdivision.

A further motivation for the work in this paper relates to listing all abstract descriptions, up to isomorphism, for various purposes. These include classifying types of diagrams that might be drawn nicely and being able to count various types of diagrams such as those which are 'atomic' (defined later). Once such a collection has been built up, we can use it to generate a library of drawn diagrams using previously developed generation techniques. Just as when searching through the library, will need to be able to efficiently check for isomorphism when producing a collection of all abstract descriptions.

3 Euler Diagrams: Syntax and Semantics

We briefly overview the syntax and semantics of Euler diagrams; the formalizations are adapted from [20, 21]. The diagram d_1 in figure 2 contains three *contours*; these are the closed curves labelled A, B and C. Contours represent sets and their spatial relationship is used to make statements about containment and disjointness of sets. So, in d_1 the contours assert that the sets A and B are disjoint because the contours do not overlap in any way; similarly $A \cap C = \emptyset$. The placement of C inside B expresses $C \subseteq B$. A *zone* is a maximal sets of points in the plane that can be described as being inside certain contours and outside the rest of the contours. The diagram d_1 contains four zones, one of which can be described as inside B but outside both A and C. A **drawn Euler diagram** is a finite collection of labelled closed curves, in which each label occurs at most once.

To formalize diagram descriptions, the input to a generator, all that is necessary is knowledge about the labels and the zones that are to be present. A zone description is taken to be a set of labels, in, following the style of [6]; the set in contains the labels of the curves that contain a particular zone. For example, the zone inside B but outside A and C in d_1 is described by $\{B\}$; given the labels A, B and C, we can deduce that the zone described by $\{B\}$ is outside A and C.

We can also talk about zone descriptions that are subsets of the label set in a diagram, d, but for which no zone occurs in d. For example $\{A, B\}$ describes a zone which is not present in d_1 (no zone is inside both A and B). Such zones are said to be *missing* from the diagram; d_1, therefore, has four missing zones. Further, we assume the existence of some fixed label set, $\mathcal{L}$, from which we choose the labels used in our Euler diagrams.

An **abstract Euler diagram** is an ordered pair, (L, Z) where $L \subseteq \mathcal{L}$ and $Z \subseteq \mathbb{P}L$. It is these abstract Euler diagrams that are the input to generation algorithms and,

thus, from which a drawn Euler diagram is generated. We frequently blur the distinction between abstract diagrams and their drawn counterparts and simply refer to Euler diagrams; the context will make it clear whether we mean 'drawn' or 'abstract'. Similarly, a set of labels will be called a zone.

At the semantic level, an *interpretation* is a universal set, U, together with an assignment of a subset of U to each contour (strictly, to contour labels) which is extended to interpret zones. Formally, an **interpretation** is a pair, (U, Ψ) where U is any set and $\Psi\colon \mathcal{L} \to \mathbb{P}U$ is a function. In a diagram $d = (L, Z)$, a zone $in \subseteq L$ represents the set $\bigcap_{l \in in} \Psi(l) \cap \bigcap_{l \in L - in} (U - \Psi(l))$. An interpretation is a **model** for d if all of the zones which are missing from d represent the empty set.

4 Isomorphic Diagrams

Given a fixed label set, $L \subseteq \mathcal{L}$, the table below identifies how many abstract diagrams there are with that label set; the set of diagrams with label set L is denoted $D(L)$.

$\lvert L\rvert$	0	1	2	3	4	5
$\lvert D(L)\rvert$	2	4	16	256	65536	4294967296

Note that given a label set L and a subset, M of L, such that $|M| = |L| - 1$, we have $|D(L)|$=$|D(M)|^2$. Of course, many of these abstract Euler diagrams are isomorphic. For example, d_1 and d_2 in figure 2 have abstract descriptions $L = \{A, B, C\}$ and zone sets $\{\{A\}, \{B\}, \{B, C\}\}$ and $\{\{C\}, \{A\}, \{A, B\}\}$ respectively which are the same up to renaming the labels. Recall that a permutation of a set S is a bijection from S to itself.

Definition 4.1 *Let $d_1 = (L_1, Z_1)$ and $d_2 = (L_2, Z_2)$ be Euler diagrams. Then d_1 and d_2 are **isomorphic** if there exists a permutation, $\sigma\colon \mathcal{L} \to \mathcal{L}$, such that the image of σ when the domain is restricted to L_1 equals L_2 and which induces a bijection, $\Sigma\colon Z_1 \to Z_2$ defined by $\Sigma(z_1) = \{\sigma(l) : l \in z_1\}$; such a permutation σ is called an **isomorphism** from d_1 to d_2.*

There is potentially a large number of non-isomorphic diagrams that have the same number of labels; the table below gives the numbers for up to five labels, where $NID(L)$ denotes the set of non-isomorphic diagrams.

$\lvert L\rvert$	0	1	2	3	4	5
$\lvert NID(L)\rvert$	2	4	12	80	3984	37333248

The table below shows how many diagrams there are with $|L|$ labels and $|Z|$ zones up to isomorphism and indicates how many brute-force isomorphism tests we might need to perform if we first compare number of labels and number of zones.

$\lvert L\rvert$	$\lvert Z\rvert$								
	0	1	2	3	4	5	6	7	8
0	1	1							
1	1	2	1						
2	1	3	4	3	1				
3	1	4	9	16	20	16	9	4	1
4	1	5	17	52	136	284	477	655	730
5	1	6	28	134	625	2674	10195	34230	100577

The numbers in the above two tables are obtained from [8], after noting the analogy with switching theory. Even with only five labels, $|L| = 5$, the number of non-isomorphic diagrams with, say, 8 zones, is rather large at 100577. When $|Z|$ increases to 16, there are 5182326 non-isomorphic diagrams with five labels. Thus it is important to have further methods of partitioning the set of non-isomorphic diagrams.

4.1 Linking Isomorphism and Semantics

The notion of isomorphism is well-defined with respect to semantics: although isomorphic diagrams are not, in general, semantically equivalent but they are *expressively equivalent*, defined below.

Definition 4.2 *Let (U, Ψ_1) and (U, Ψ_2) be two interpretations. We say (U, Ψ_1) and (U, Ψ_2) are **permutation-equivalent** if there exists a permutation of $\mathcal{L}$, say $\sigma\colon \mathcal{L} \to \mathcal{L}$, such that for all $l \in \mathcal{L}$, $\Psi_1(l) = \Psi_2(\sigma(l))$.*

To illustrate, the interpretation with $U = \{1, 2, 3\}$, $\Psi(A) = \{1\}$, $\Psi(B) = \{2, 3\}$ and $\Psi(C) = \{3\}$ is a model for d_1 but not d_2 in figure 2. However, the interpretation $U = \{1, 2, 3\}$, $\Psi(C) = \{1\}$, $\Psi(A) = \{2, 3\}$ and $\Psi(B) = \{3\}$ is a model for d_2 but not d_1. These two models are permutation equivalent and d_1 and d_2 are isomorphic.

Definition 4.3 *Let $d_1 = (L_1, Z_1)$ and $d_2 = (L_2, Z_2)$ be Euler diagrams. Then d_1 and d_2 are **expressively equivalent** if there exists a fixed permutation $\sigma\colon \mathcal{L} \to \mathcal{L}$ such that the models for d_2 are precisely those which are permutation equivalent to the models for d_1 under σ.*

Theorem 4.1 *Two diagrams are isomorphic if and only if they are expressively equivalent.*

5 Isomorphism Invariants

In the worst case, determining whether two abstract Euler diagrams are isomorphic takes exponential time in proportion to the number of labels in the diagram. Of course, there are some obvious checks that one can perform to reduce the computations involved by using invariants. The invariants we define below each capture a certain type of structure present in Euler diagrams that is preserved under

isomorphism. The two most obvious invariants are the number of labels present and the number of zones present. However, these invariants do not capture the complexity of the relationships between the closed curves (i.e. how the zone set relates to the label set). Thus, we define a range of invariants that go some what towards capturing features of the zone set and its relationship with the label set. For example, a diagram that contains exactly two zones inside one contour will not be isomorphic to any diagram that has no contour which contains exactly two zones. Our refined invariants take this kind of difference between non-isomorphic diagrams into account.

5.1 Zone Invariants

Trivially, we can use the number of zones as an isomorphism invariant.

Lemma 5.1 (Invariants) *Let $d_1 = (L_1, Z_1)$ and $d_2 = (L_2, Z_2)$ be isomorphic Euler diagrams. Then $|Z_1| = |Z_2|$.*

Rather than simply counting the zones, we can count the number of zones of each size, where the size of a zone is the number of labels it contains. To illustrate, isomorphic diagrams in figure 2 each have 1 zone inside 0 contours, 2 zones inside 2 contours, 1 zone inside 2 contours, and no zones inside all three contours.

Definition 5.1 *Let $d = (L, Z)$ be an Euler diagram. Then the **zone-partition sequence** associated with d, denoted $ZPS(d)$, is defined to be a sequence of natural numbers, $ZPS(d) = (s_0, s_1, ..., s_{|L|})$, where s_i is the number of zones in d with cardinality i (informally, s_i is the number of zones in d that are 'inside' i contours).*

Theorem 5.1 (Invariant) *Given two isomorphic diagrams d_1 and d_2, $ZPS(d_1) = ZPS(d_2)$.*

Zone-partition sequences provide a method of subdividing the space of abstract diagrams into smaller classes, beyond the subdivision provided by simply counting the labels and the zones. Indeed, when we use zone-partition sequences to subdivide our library of examples, we no longer need to consider the number of labels and number of zones, since these are derivable from $ZPS(d)$; the number of labels in d is the length of $ZPS(d)$ and the number of zones is the sum of the entries in $ZPS(d)$.

Hardly surprisingly, there are diagrams which are not isomorphic but that have the same zone-partition sequence. To illustrate, the non-isomorphic diagrams in figure 3 both have zone-partition sequence $(1, 2, 2, 0, 0)$.

5.2 Label Invariants

As with the number of zones, we can use the number of labels as an isomorphism invariant.

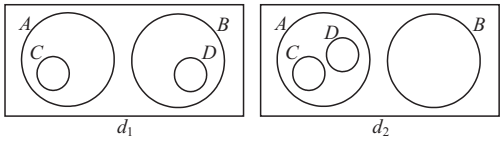

Figure 3. Zone partition sequences.

Lemma 5.2 (Invariants) *Let $d_1 = (L_1, Z_1)$ and $d_2 = (L_2, Z_2)$ be isomorphic Euler diagrams. Then $|L_1| = |L_2|$.*

We can further refine our subdivision of the set of atomic diagrams by considering how many zones are contained by each label. First we have the following lemma, whose proof proceeds by a simple induction on $|L|$.

Lemma 5.3 *For any Euler diagram $d = (L, Z)$ and label $l \in L$, there are at most $2^{|L|-1}$ zones, z, in d that contain the label l, that is $l \in z$.*

Definition 5.2 *Let $d = (L, Z)$ be an Euler diagram. Then the **label-partition sequence** associated with d, denoted $LPS(d)$, is defined to be a sequence of natural numbers, $LPS(d) = (t_0, t_1, ..., t_{2^{|L|-1}})$, where t_i is the number of labels in d which contain i zones (informally, t_i is the number of contours in d that contain i zones).*

The diagrams in figure 3, which have equal zone-partition sequence, have different label-partition sequences: $LPS(d_1) = (0, 2, 2, 0, 0, 0, 0, 0, 0)$ and $LPS(d_2) = (0, 3, 0, 1, 0, 0, 0, 0, 0)$.

Theorem 5.2 (Invariant) *Given two isomorphic diagrams d_1 and d_2, $LPS(d_1) = LPS(d_2)$.*

5.3 Using Labels and Zones

A further subdivision of the set of abstract diagrams can be achieved by considering in more detail the relationship between the labels used and the zones in which they occur. First, recall that a lexicographical ordering of sequences can be achieved by ordering $(l_1, l_2, ..., l_n)$ before $(m_1, m_2, ..., m_p)$ whenever there is a j such that $l_i = m_i$ for all $i < j$ and $l_j < m_j$.

Definition 5.3 *Let $d = (L, Z)$ be an Euler diagram. For each label, $l \in L$, we define a **label-zone sequence**, denoted $LZS(l, d)$ to be $(l_1, l_2, ..., l_{2^{(|L|-1)}})$ where l_i is the number of zones in d that contain i labels and which include l, that is $l_i = |\{z \in Z : l \in z \wedge |z| = i\}|$. Further, we define the **label-zone sequence** for d, denoted $LZS(d)$, to be the lexicographically ordered sequence of label-zone sequences for the labels in d.*

For example, in figure 3, the contour A in d_1 contains three zones; of these, one is inside just A and two are inside A and another contour. This gives $LZS(A, d_1) = (1, 1, 0, 0)$. Similarly, B has label-zone sequence $LZS(B, d_1) = (1, 1, 0, 0)$. For C and D we have

$$LZS(C, d_1) = LZS(D, d_1) = (0, 1, 0, 0).$$

Thus,

$$LZS(d_1) = ((0, 1, 0, 0), (0, 1, 0, 0), (1, 1, 0, 0), (1, 1, 0, 0)).$$

The label-zone sequence for d_2 is

$$LZS(d_2) = ((0, 1, 0, 0), (0, 1, 0, 0), (1, 0, 0, 0), (1, 2, 0, 0)).$$

This more refined invariant captures more accurately the relationship between labels and the zones they contain.

Lemma 5.4 *Given diagrams d_1 and d_2 such that $LZS(d_1) = LZS(d_2)$, we have the following*

1. $LPS(d_1) = LPS(d_2)$ *and*
2. $ZPS(d_1) = ZPS(d_2)$.

Theorem 5.3 (Invariant) *Given isomorphic diagrams d_1 and d_2, $LZS(d_1) = LZS(d_2)$.*

5.4 Refinements to Blocks

Refinements to the above invariants can be made based on the observation that one only requires so-called *atomic* diagrams from which one can generate all diagrams in a relatively simple way. An atomic diagram is one in which the curves form a connected component of the plane; non-atomic diagrams are called **nested** [7].

This notion of nesting at the drawn diagram level has an analogy at the abstract level, defined by considering the *dual graph* (in [5, 6] this graph is called the *superdual*). Given an abstract diagram, $d = (L, Z)$, the **dual graph** of d, denoted $dual(d)$, has Z as its vertex set and there is any edge between vertex v_1 and vertex v_2 if the symmetric difference of v_1 and v_2 contains exactly one label. Figure 4 shows a diagram with its dual graph.

Under certain well-formedness conditions, the absence cut-vertices in the dual graph corresponds to atomic diagrams [7]. We can use the dual graph to further partition the space of non-isomorphic diagrams. Recall, a block of a graph is a maximal connected subgraph that contains no cut vertex.

Definition 5.4 *A **block** of an abstract Euler diagram, $d_1 = (L, Z_1)$ is an abstract Euler diagram, $d_2 = (L, Z_2)$ such that the dual of d_2 is a block of the dual graph of d_1.*

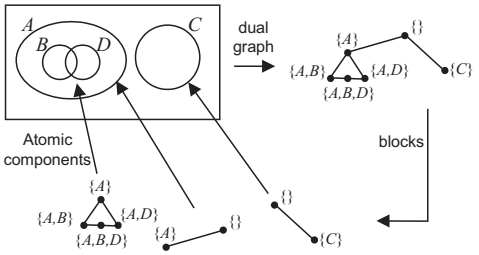

Figure 4. Finding diagram blocks.

An illustration can be seen in figure 4. The blocks of d can be found by computing the blocks of $dual(d)$ for which there are known algorithms, see [1] for example.

Definition 5.5 *Let d be an Euler diagram. The **blockwise zone-partition sequence** for d, denoted $BZPS(d)$, is the lexicographically ordered sequence of zone-partition sequences for the blocks of d.*

To illustrate, the nested diagram d_1 in figure 4 has blockwise zone-partition sequence

$$BZPS(d_1) = ((0, 1, 2, 1, 0), (1, 1, 0, 0, 0), (1, 1, 0, 0, 0)).$$

Theorem 5.4 *Given two isomorphic diagrams, d_1 and d_2, $BZPS(d_1) = BZPS(d_2)$.*

The definitions of the other sequences given above, such as the label-partition sequence, extend to blockwise sequences in a manner similar to that exemplified by the zone-partition sequence. Moreover, these further blockwise sequences are also invariant under isomorphism. We conjecture that these refined blockwise invariants will be particularly helpful when abstract descriptions have large numbers of labels.

5.5 Refinements to Complements

We note that the more zones there are in a diagram, the more checks we need to perform when seeking to establish whether two diagrams are isomorphic. When more than half of the zones are present, we can reduce these number of checks by comparing diagram *complements*.

Definition 5.6 *The **complement** of an Euler diagram, $d_1 = (L, Z_1)$, is an Euler diagram, $d = (L, Z_2)$, where $Z_2 = \mathbb{P}L - Z_1$.*

In figure 5, d_1 has complement $(L = \{A, B, C\}, Z = \{\{B\}, \{B, C\}\})$. Clearly, given a diagram, d, the complement of its complement is d.

Theorem 5.5 *Given isomorphic diagrams d_1 and d_2, their complements are isomorphic.*

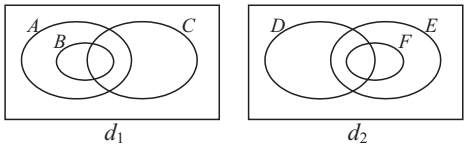

Figure 5. Diagram complements.

To illustrate the computational savings argument, suppose we wish to determine whether the diagrams d_1 and d_2 in figure 5 are isomorphic and that we have already established that the various sequences associated with them, which are invariant under isomorphism, are equal. Then we would need to construct bijections, $\sigma\colon \{A, B, C\} \rightarrow \{D, E, F\}$, and determine whether they are isomorphisms by checking that each zone in d_1 maps to a zone in d_2 (that is, check whether σ extends bijectively to the zones). Alternatively, we can check whether σ extends bijectively to the complements, reducing the number of checks (the complements of both diagrams contain just two zones whereas each original diagram contains 6 zones).

6 Implementation

In this section, an implementation of abstract Euler diagram isomorphism testing in a software system is described. The system implements the optimizations discussed previously for isomorphism testing. However, it also includes additional improvements for listing all the unique abstract diagrams for a particular number of sets.

When listing all abstract diagrams up to isomorphism, the implementation only iterates through abstract diagrams that do not have the empty zone, $\emptyset$, present. To get the full list of abstract diagrams, the ones in the generated list simply have to have the empty zone added to them. This reduces the number of abstract diagrams built by a half. For example, with two labels, we will generate the abstract diagram $(\{A, B\}, \{A\})$ from which $(\{A, B\}, \{\emptyset, \{A\}\})$ can subsequently be generated.

We now outline our method for producing all abstract diagrams with some fixed number of labels. Firstly, the Venn diagram with the required number of labels, $|L|$, is found, and each of its zones is assigned an integer, from one to the number of zones. For example, when we require two labels, and take $L = \{A, B\}$, the zones in the Venn diagram are (excluding $\emptyset$) $\{A\}, \{B\}, \{A, B\}$; each zone is then assigned a number, $\{A\} \mapsto 1, \{B\} \mapsto 2$ and $\{A, B\} \mapsto 3$. Using this, we can give a binary word that describes the zone set; for example, a diagram that contains the zones $\{A\}$ and $\{A, B\}$ would be described by 101 since, for example, $\{A, B\}$ was assigned to 3 and its presence is indicate by a 1 in the third place of 101.

We then iterate from zero to the number of diagrams to be listed (minus one), $|D(L)| - 1$ (we subtract one since we have already generated the Venn diagram), and produce an abstract diagram by treating them number as a binary. In our two-contour example, the number of diagrams to be listed is six, since we do not include the zone $\emptyset$; there are 12 non-isomorphic diagrams with two zones. It is easy to see that there are six binary words containing three bits over the alphabet $\{0, 1\}$.

As stated above, the presence of a zone is given by a 1 in the binary description. To further illustrate, the list of all zones for 3 contours is $(\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\})$, seven zones (and 128 diagrams to be generated, as the empty zone is not present), with $\{A\}$ numbered one and $\{A, B, C\}$ numbered seven. Combination number 69, with binary representation 0100111, is the abstract diagram with zone-list $(\{B\}, \{A, C\}, \{B, C\}, \{A, B, C\})$.

The method above is used to produce all abstract diagrams with some fixed number of labels: each binary word is converted into a subset of the Venn zone set, giving all subsets of the Venn zone set. During the process of listing all such diagrams, any diagram which does not have A in its first zone are not generated. This is because such diagrams are isomorphic to another diagram with A in its first zone, and so the following processes do not need to be applied.

Each abstract diagram is converted into a normal form. This normal form maps the sets to labels so that they are in a particular order, with A the highest value label, B the next, and so on. The labels are, firstly, compared by checking each zone size (by zone size, we simply mean the cardinality of the zone). A label occurring in a smaller size zone than another label gets a higher value. Where there is no difference in the zone size that they appear in, then a label occurring in a zone with a higher ordered label where the other label does not gets a higher value. Otherwise, if an order between labels cannot be derived then they are ordered arbitrarily. The labels in zones are lexically ordered, then the zones in abstract diagrams are first ordered by size, then lexical ordering of same size zones is performed.

For example, given the diagram with zone set $\{\{B\}, \{D\}, \{A, C\}, \{B, C\}, \{B, C, D\}\}$, B and D are higher than A and C because they are in zones of size 1, B is ordered higher than D because it appears in zones of size 2. Moreover, C is higher than A because it appears next to B. Giving a mapping $\sigma(B) = A$, $\sigma(D) = B$, $\sigma(C) = C$ and $\sigma(A) = D$, the diagram has the normalized zone set $\{\{A\}, \{B\}, \{A, C\}, \{C, D\}, \{A, B, C\}\}$.

This means that many isomorphic diagrams will have the same label ordering. So, when listing all unique diagrams, if the normal form already occurs in the set of known diagrams, there is no need to do the further isomorphism tests. Of, course, this normalization repeats some of the steps in

the isomorphism optimization given below, and the isomorphism tests for two diagrams that reduce to the same normal form would tend to be quick in any case, as the brute force part of the algorithm would not have many alternative mappings to test. However, it is useful to have a consistent way of displaying abstract descriptions. The set of known unique abstract diagrams is stored in a hash set. This speeds up the test to determine whether a normalized abstract diagram is already in the set.

If equality by normalization cannot be derived then the current normalized abstract diagram is compared against the set of known unique abstract diagrams by using an isomorphism test. This test performs the optimizations described previously in sections 5.1 to 5.3: the invariants are compared first and, if equal, a brute force algorithm has to be executed to perform isomorphism tests. First, a partition of the label set is discovered. Each set in the partition contains the labels that can map to the same choices of labels in the other abstract diagram. Hence, members of a partition are those labels with indistinguishable values from the normalization process. Given that the sets can be placed in order, further optimizations can be carried out, such as ensuring each pair of sets in order from both diagrams are of equal size.

This table gives the number times that pairs of diagrams have been identified as non-isomorphic by simply checking the invariant when generating a list of all unique abstract diagrams with a given number labels. For example, for diagrams with 4 contours, the zone-partition sequence alone identifies 3446662 diagram pairs as non-isomorphic. Both the normalization and the absence of the zone $\emptyset$ (that contained by no contours) optimization have been used in computing table entries. As a result, the number of times a brute-force tests for isomorphism has been applied is greatly reduced for the more effective invariants. For example, using the label-zone sequence to identify non-isomorphic diagrams, the number of brute-force isomorphism tests for four labels was reduced to just 1886; this is a very small fraction of the number of isomorphism tests that were saved by using the invariant, as detailed in the table (3468679).

Invariant	$\|\boldsymbol{L}\|$			
	1	2	3	4
$\|\boldsymbol{L}\|$	1	9	231	138645
$\|\boldsymbol{Z}\|$	1	13	687	3018907
$\boldsymbol{LPS}$	1	14	784	3413693
$\boldsymbol{ZPS}$	1	15	814	3446662
$\boldsymbol{LZS}$	1	15	822	3468679

7 Conclusion

We have defined a notion of isomorphism between abstract Euler diagrams and presented a collection of invariants that can be used to reduce the time taken to identify whether two abstract diagrams are isomorphic. The resulting efficiency savings are likely to be useful when using a library of examples to display Euler diagrams, for instance. Our implementation includes some optimizations and we have provided data comparisons to demonstrate the effects of utilizing different invariants. Further optimizations are possible and, since the problem of determining isomorphism between Euler diagrams is identical to the hypergraph isomorphism problem, optimizations can be drawn from graph isomorphism and should be very feasible. For example, we could convert to a constraint satisfaction task [15, 19] or adapt the graph isomorphism methods of Nauty [18].

Another useful application of our work is in the use of the invariants such as the zone-partition sequence to help build an initial library of examples, enabling the generation of a wide spread of diagrams; the idea being that drawn diagrams with nice visual properties are stored for each abstract diagram in the library. The intention would be that such that a library based generation system would have methods for combining diagrams, and the abstract diagram generation and comparison techniques could help guide the design of the rules for diagram combination.

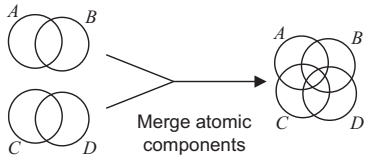

Figure 6. Combining diagrams.

It may be that diagrams in a library can be combined in sophisticated ways, such as taking two atomic components and merging them together to create a new diagram; figure 6 shows the merger of two atomic diagrams into a single atomic diagram (as opposed to a nested diagram). The theory required to facilitate this type of constructing is currently being developed and would reduce the number of diagrams required to be stored in the library.

Further detailed experiments to determine the best way in which to use the invariants to divide the search space are also required. The manner in which the invariants are used will impact the search through a library in order to extract a diagram which corresponds to an input abstract diagram description, for example.

Acknowledgements This research is supported by EPSRC

grants EP/E011160/01 and EP/E010393/01 for the *Visualization with Euler Diagrams* project. Thanks also to Ebru Keyman for useful discussions on aspects of this paper.

References

[1] G. Chartrand and O. Ollermann. *Applied and algorithmic graph theory*. McGraw-Hill, 1993.

[2] S. Chow and F. Ruskey. Drawing area-proportional Venn and Euler diagrams. In *Proceedings of Graph Drawing 2003, Perugia, Italy*, volume 2912 of *LNCS*, pages 466–477. Springer-Verlag, September 2003.

[3] S. Chow and F. Ruskey. Towards a general solution to drawing area-proportional Euler diagrams. In *Proceedings of Euler Diagrams*, volume 134 of *ENTCS*, pages 3–18, 2005.

[4] R. DeChiara, U. Erra, and V. Scarano. A system for virtual directories using Euler diagrams. In *Proceedings of Euler Diagrams 04*, volume 134 of *Electronic Notes in Theoretical Computer Science*, pages 33–53, 2005.

[5] J. Flower, A. Fish, and J. Howse. Euler diagram generation. *Journal of Visual Languages and Computing*, available online, 2008.

[6] J. Flower and J. Howse. Generating Euler diagrams. In *Proceedings of 2nd International Conference on the Theory and Application of Diagrams*, pages 61–75, Georgia, USA, April 2002. Springer.

[7] J. Flower, J. Howse, and J. Taylor. Nesting in Euler diagrams: syntax, semantics and construction. *Software and Systems Modelling*, 3:55–67, March 2004.

[8] A. Harrison. *Introduction to Switching and Automata Theory*. McGraw-Hill, 1965.

[9] P. Hayes, T. Eskridge, R. Saavedra, T. Reichherzer, M. Mehrotra, and D. Bobrovnikoff. Collaborative knowledge capture in ontologies. In *Proceedings of the 3rd International Conference on Knowledge Capture*, pages 99–106, 2005.

[10] J. Howse and S. Schuman. Precise visual modelling. *Journal of Software and Systems Modeling*, 4:310–325, 2005.

[11] J. Howse, G. Stapleton, and J. Taylor. Spider diagrams. *LMS Journal of Computation and Mathematics*, 8:145–194, 2005.

[12] S. Kent. Constraint diagrams: Visualizing invariants in object oriented modelling. In *Proceedings of OOPSLA97*, pages 327–341. ACM Press, October 1997.

[13] H. Kestler, A. Muller, T. Gress, and M. Buchholz. Generalized Venn diagrams: A new method for visualizing complex genetic set relations. *Journal of Bioinformatics*, 21(8):1592–1595, 2005.

[14] S.-K. Kim and D. Carrington. Visualization of formal specifications. In *6th Aisa Pacific Software Engineering Conference*, pages 102–109, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[15] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12:403–422, 2002.

[16] J. Lovdahl. *Towards a Visual Editing Environment for the Languages of the Semantic Web*. PhD thesis, Linkoping University, 2002.

[17] E. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proceedings of 31st Annual ACM Symposium on Theory of Computing*, 1999.

[18] B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[19] S. Sorlin and C. Solnon. A global constraint for graph isomorphism problems. In *6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 287–301. Springer, 2004.

[20] G. Stapleton, J. Masthoff, J. Flower, A. Fish, and J. Southern. Automated theorem proving in Euler diagrams systems. *Journal of Automated Reasoning*, 39:431–470, 2007.

[21] G. Stapleton, S. Thompson, J. Howse, and J. Taylor. The expressiveness of spider diagrams. *Journal of Logic and Computation*, 14(6):857–880, December 2004.

[22] N. Swoboda and G. Allwein. Using DAG transformations to verify Euler/Venn homogeneous and Euler/Venn FOL heterogeneous rules of inference. *Journal on Software and System Modeling*, 3(2):136–149, 2004.

[23] N. Swoboda and G. Allwein. Heterogeneous reasoning with Euler/Venn diagrams containing named constants and FOL. In *Proceedings of Euler Diagrams 2004*, volume 134 of *ENTCS*. Elsevier Science, 2005.

[24] J. Thièvre, M. Viaud, and A. Verroust-Blondet. Using euler diagrams in traditional library environments. In *Euler Diagrams 2004*, volume 134 of *ENTCS*, pages 189–202. ENTCS, 2005.

[25] A. Verroust and M.-L. Viaud. Ensuring the drawability of Euler diagrams for up to eight sets. In *Proceedings of 3rd International Conference on the Theory and Application of Diagrams*, volume 2980 of *LNAI*, pages 128–141, Cambridge, UK, 2004. Springer.