Generating Proofs with Spider Diagrams Using Heuristics

Jean Flower, Judith Masthoff and Gem Stapleton
Visual Modelling Group
University of Brighton
Brighton, UK
Email: {j.a.flower,judith.masthoff,g.e.stapleton}@brighton.ac.uk

***Abstract*— We apply the A^* algorithm to guide a diagrammatic theorem proving tool. The algorithm requires a heuristic function, which provides a metric on the search space. In this paper we present a collection of metrics between two spider diagrams. We combine these metrics to give a heuristic function that provides a lower bound on the length of a shortest proof from one spider diagram to another, using a collection of sound reasoning rules. We compare the effectiveness of our approach with a breadth-first search for proofs.**

I. Introduction

Simple diagrammatic systems that inspired spider diagrams are Venn and Euler diagrams. In Venn diagrams all possible intersections between contours must occur and shading is used to represent the empty set. Diagram d_1 in Fig. 1 is a Venn diagram. Venn-Peirce diagrams [11] extend the Venn diagram notation, using additional syntax to represent non-empty sets. Euler diagrams exploit topological properties of enclosure, exclusion and intersection to represent subsets, disjoint sets and set intersection respectively. Spider diagrams [4], [7], [8],

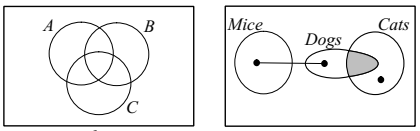

Fig. 1. A Venn diagram and a spider diagram.

[10] are based on Euler diagrams. *Spiders* are used to represent the existence of elements and *shading* is used to place upper bounds on the cardinalities of sets. A spider is drawn as a collection of dots joined by lines. The spider diagram d_2 in Fig. 1 expresses the statement "no mice are cats or dogs, no dogs are cats, there is a cat and there is something that is either a mouse or a dog". Sound and complete reasoning rules for spider diagram systems have been given [7], [8], [10].

As argued in [3], it is important for automated diagrammatic reasoning systems to produce proofs that are easy to understand by humans. For this reason, our ambition is to produce diagrammatic proofs using diagram transformations instead of converting to first order logic and using existing theorem provers. An existing theorem prover for spider diagrams successfully writes proofs [2], but they can be long and unwieldy. In [3], we presented a new approach to proof writing in diagrammatic systems, which is guaranteed to find shortest proofs and can be extended to incorporate other readability criteria. We applied the A^* algorithm and developed an admissible heuristic function to guide automatic proof construction. However, the work presented in [3] was limited to the simple case of so-called unitary spider diagrams. Here, we extend that work to the significantly more challenging general case of so-called compound spider diagrams.

II. Spider diagrams

We now informally introduce the spider diagram system.

A. Syntax and semantics of spider diagrams

In this section, we will give an informal description of the syntax and semantics of spider diagrams. Details and formal definitions can be found in [8]. A **contour** is a labelled closed curve in the diagram used to denote a set. The **boundary rectangle** is an unlabelled rectangle that bounds the diagram and denotes the universal set. A **zone**, roughly speaking, is a bounded area in the diagram having no other bounded area contained within it. A zone can be described by the set of labels of the contours that contain it and the set of labels of the contours that exclude it. A zone denotes a set by intersection and difference of the sets denoted by the contours. A **region** is a set of zones.

A **spider** is a tree with nodes, called **feet**, placed in different zones. A spider **touches** a zone if one of its feet appears in that zone. The set of zones a spider touches is called its **habitat**. A spider denotes the existence of an element in the set represented by its habitat. Distinct spiders represent the existence of distinct elements. A zone can be **shaded**. In the set represented by a shaded zone, all of the elements are represented by spiders. So, a shaded zone with no spiders in it represents an empty set. A **unitary diagram** is a finite collection of contours (with distinct labels), shading and spiders properly contained by a boundary rectangle.

The unitary diagram d_2 in Fig. 1 contains three contours and five zones, of which one is shaded. There are two spiders. The spider with one foot inhabits the zone inside (the contour labelled) $Cats$, but outside $Dogs$ and $Mice$. The other spider inhabits the region which consists of the zone inside $Mice$ and the zone inside $Dogs$ but outside $Cats$.

Unitary diagrams form the building blocks of **compound diagrams**. To enable us to present negated, disjunctive and

conjunctive information between unitary diagrams, we use connectives: $\neg$, $\sqcup$ and $\sqcap$. If D_1 and D_2 are spider diagrams then so are $\neg D_1$ ("not D_1"), $D_1 \sqcup D_2$ ("D_1 or D_2") and $D_1 \sqcap D_2$ ("D_1 and D_2"). The semantics of compound diagrams extend those of unitary diagrams in the obvious way.

B. Reasoning with spider diagrams

We now give informal descriptions of the sound but not a complete set of reasoning rules for spider diagrams. For formal descriptions see [8].

Add contour. A new contour can be added to a unitary diagram. Each zone is split into two zones (one inside and one outside the new contour) and shading is preserved. Each spider foot is replaced by a connected pair of feet, one in each of the two new zones. For example, in Fig. 2, d_2 is obtained from d_1 by adding a contour. This rule is reversible and we will refer to its reverse as **Delete contour**.

Add shaded zone. A new, shaded zone can be added to a unitary diagram. This rule is reversible and we will refer to its reverse as **Delete shaded zone**. For example, in Fig. 2, diagram d_3 is obtained from d_2 by deleting a shaded zone.

Erase shading. Shading can be erased from any zone in a unitary diagram.

Delete spider. A spider whose habitat is completely non-shaded can be deleted from a unitary diagram.

Add spider foot. In a unitary diagram, a foot can be added to a spider in a zone it does not yet touch.

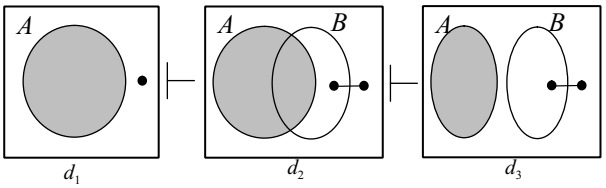

Fig. 2. Applications of "add contour" and "delete shaded zone".

All of the remaining diagrammatic rules are reversible.

Split spider. A unitary diagram d containing a spider s whose habitat has a partition into non-empty regions r_1 and r_2 can be replaced by $d_1 \sqcup d_2$, where d_1 and d_2 are copies of d except that the habitat of s is reduced to r_1 in d_1 and r_2 in d_2. For instance, diagram d in Fig. 3 has a spider with two feet. We can split this spider into two parts, giving $d_1 \sqcup d_2$.

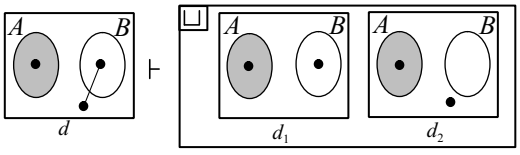

Fig. 3. An application of rule "split spider".

Excluded Middle. A unitary diagram d with a non-shaded zone z can be replaced by $d_1 \sqcup d_2$, where d_1 and d_2 are copies of d except that z is shaded in d_1 and contains an additional spider in d_2. For instance, d in Fig. 4 has a non-shaded zone $B - C$. Applying excluded middle to this zone yields $d_1 \sqcup d_2$.

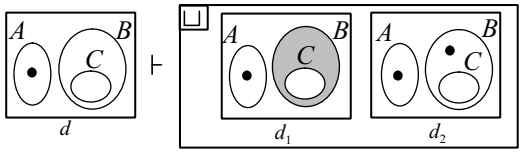

Fig. 4. An application of rule "excluded middle".

An **α-diagram** is a diagram in which each spider has exactly one foot. Two unitary α-diagrams with (essentially) the same zone set are in **contradiction** if a zone is shaded in one diagram and contains more spiders in the other. For example, in figure 4, the diagrams d_1 and d_2 are in contradiction because $B - C$ is shaded and contains no spiders in d_1 but contains one spider in d_2.

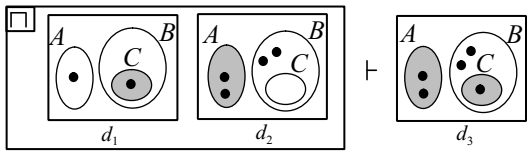

Fig. 5. An application of rule "Combining".

Combining. A compound diagram consisting of the conjunction of two non-contradictory α-diagrams d_1 and d_2 whose zone sets are the same can be combined into a single unitary diagram, d_3, with the same zone set. The number of spiders in any zone in d_3 is the maximum number of spiders in that zone in d_1 and d_2, and a zone is shaded in d_3 if it is shaded in either d_1 or d_2. For example, diagram $d_1 \sqcap d_2$ in Fig. 5 can be replaced by d_3. We note here that due to the non-deterministic nature of the reverse of this rule, we have not included the reverse in our implementation.

There are also reasoning rules that have analogies in logic. We include in our set of rules **Idempotency** (for example, $D \vdash D \sqcap D$), **De Morgan's laws**, **Involution** ($\neg\neg D \equiv_{\vdash} D$) and **Distributivity**. All of these rules are reversible. We also include the **Absorption** rules which state that $D_1 \sqcap (D_1 \sqcup D_2)$ can be replaced by D_1 and $D_1 \sqcup (D_1 \sqcap D_2)$ can be replaced by D_1. Whilst the reverses of the absorption rules are sound, due to the non-deterministic nature of the reverses these have not been included in our implementation.

If D_1 can be transformed into D_2 by a reversible rule, then any occurrence of D_1 in a compound diagram can be replaced by D_2. If D_1 can be transformed into D_2 by a non-reversible rule, then any occurrence of D_1 in a compound diagram can be replaced by D_2, provided the occurrence of D_1 being replaced is 'inside an even number of not's'. For example, in the diagram $\neg((D_1 \sqcap D_3) \sqcup (D_4 \sqcap \neg D_1))$ the first occurrence of D_1 is not inside an even number of not's, but the second is. We say that diagrams D_2 is **obtainable** from diagram D_1, denoted $D_1 \vdash D_2$, if and only if there is a sequence of diagrams $\langle D^1, D^2, ..., D^m \rangle$ such that $D^1 = D_1$, $D^m = D_2$ and, for each k where $1 \leq k < m$, D^k can be transformed into D^{k+1} by a single application of one of the reasoning rules. Such a sequence of diagrams is called a **proof** from **premise** D_1 to **conclusion** D_2.

III. A* APPLIED TO PROOF WRITING

To construct a proof, a rule needs to be applied to the premise diagram, followed by another rule to the resulting diagram, and so on, until the conclusion diagram is reached. At any stage, multiple rules might be applicable. The problem of deciding which rule to apply is an example of a more general class of so-called search problems, for which various algorithms have been developed (see [9] for an overview). A^* is a well known search algorithm [5].

A^* stores an ordered sequence of proof attempts. Initially, this sequence only contains a zero length proof attempt, namely the premise diagram. Repeatedly, A^* removes the first proof attempt from the sequence and considers it. If the last diagram of the proof attempt is the conclusion diagram, then an optimal proof has been found. Otherwise, it constructs additional proof attempts, by extending the proof attempt under consideration, applying rules wherever possible to the last diagram.

The effectiveness of A^* and the definition of "optimal" is dependent upon the ordering imposed on the proof attempt sequence. The ordering is derived from the sum of two functions. One function, called the **heuristic**, estimates how far the last diagram in the proof attempt is from the conclusion diagram. The other, called the **cost**, calculates how costly it has been to reach the last diagram from the premise diagram. The new proof attempts are inserted into the sequence, ordered according to the cost plus heuristic. A^* always finds the solution with the lowest cost, if one exists, provided the heuristic used is **admissible** [1]. A heuristic is admissible if it is **optimistic**, which means that it never overestimates the cost of getting from a premise diagram to a conclusion diagram. We define all rules to have a cost equal to one, which means that any admissible heuristic gives a lower bound on the number of proof steps needed in order to reach the conclusion diagram.

The amount of memory and time needed by A^* depends heavily on the quality of the heuristic used. For instance, a heuristic that is the constant function zero is admissible, but will result in a breadth-first search of the state space, giving long and impractical searches. The better the heuristic (in the sense of accurately predicting the lowest cost of a proof), the less memory and time are needed for the search.

IV. THE HEURISTIC FUNCTION

To define the heuristic function, we capture differences between the premise diagram and the conclusion diagram to give an estimate of the length of a shortest proof. In [3], we proposed several metrics to capture differences between two unitary diagrams, focussing on the difference in contour sets, zone sets, shaded zone sets, and spiders. These metrics were combined to provide a heuristic function for unitary diagrams. We will use similar metrics to judge the similarity between compound diagrams, in addition to new metrics to capture differences in the structure of a compound diagram. First, we must determine what we mean by the contour set of a compound diagram (and similarly for zones, etc). Perhaps surprisingly, the most useful approach is not the obvious one: to take the union of the sets of contours of the unitary parts. We now illustrate why this naive approach is not useful. Suppose we were to take the union of the contour sets of a diagram's unitary components as a measure of the contours in that diagram and to define the cardinality of the symmetric difference of the contour sets for D_1 and D_2 as our contour difference metric between D_1 and D_2. Such a metric should be good at guiding applications of the Add Contour and Delete Contour rules. Assume $D_1 \sqcap (D_1 \sqcup D_2)$ is our premise diagram, and D_1 our conclusion diagram. The diagram D_1 could have a vastly different contour set to D_2 (and, therefore, to $D_1 \sqcap (D_1 \sqcup D_2)$). Using the absorption rule, the premise diagram can be changed into the conclusion diagram in one step. Hence, for admissibility to hold, the heuristic must be at most 1 but the cardinality of the symmetric difference of the contour sets may be much larger than 1. We would have to cap the metric at 1, and this would will lead to a weak heuristic when we need to apply the Add/Delete Contour rules many times. Therefore, we will not use a simple union.

Actually, we would prefer a contour measure to be invariant under all the logic rules, and only to reflect the need for applications of the Add and Delete Contour rules. Each measure is designed to be invariant under many rules (if we apply a rule then the measure remains unchanged) and variant under few rules (if we apply a rule then the measure changes). So, we would prefer a measure of the contour set of $D_1 \sqcap (D_1 \sqcup D_2)$ to be the same as that of D_1. We would like to define $Contours(D_1 \sqcap D_2)$ and $Contours(D_1 \sqcup D_2)$ in such a way as to achieve this. There are two obvious operations which can be done on sets: union and intersection. If we define $Contours(D_1 \sqcap D_2)$ as the union of $Contours(D_1)$ and $Contours(D_2)$, and $Contours(D_1 \sqcup D_2)$ as the intersection of $Contours(D_1)$ and $Contours(D_2)$, then:

$$\begin{aligned} &Contours(D_1 \sqcap (D_1 \sqcup D_2)) = \\ &Contours(D_1) \cup Contours(D_1 \sqcup D_2) = \\ &Contours(D_1) \cup (Contours(D_1) \cap Contours(D_2)) = \\ &\qquad\qquad Contours(D_1). \end{aligned}$$

A similar result can be achieved by performing these operations the other way around (i.e., using intersection for conjunction, and union for disjunction). We will call the contour set obtained using the first definition m_1 (m for measure), and the contour set obtained using the second definition m_2. Both definitions are required because we have explicit negation of diagrams in our system. To make sure the contour sets are invariant under involution (and De Morgan's laws) we use $m_1(\neg D_1) = m_2(D_1)$ (and similarly, $m_2(\neg D_1) = m_1(D_1)$).

A similar approach can be taken to define measures for zones, shaded zones and spiders. To avoid repetition and show invariance under the logic rules, in the next section we will generalize this approach.

A. Building a set of independent metrics

To define our metrics, we first define various measures on diagrams. As discussed previously, a useful measure of the

contours in a diagram should be invariant under the logic rules but variant under the Add/Delete Contour rules in order to steer the proof writer towards applying the Add/Delete Contour rules when they are required.

In this section, we describe generic measures and show how these are invariant under logic rules. Intuition about these generic measures may be gained by comparing them with the specific example at the beginning of this section.

If D is a diagram, define a pair of measures $m_1(D)$ and $m_2(D)$ recursively using families of n-ary functions with domain X^n (X will be determined by the context) $g_{1,n}\colon X^n \to X$ and $g_{2,n}\colon X^n \to X$ (where $m_i(D) \in X$). For instance, in the example for contours given above, $g_{1,n}$ takes the union of n sets and $g_{2,n}$ takes the intersection of sets. We start by defining

$$m_i(\neg D) = m_j(D)$$

($i \neq j$) which ensures that m_1 and m_2 are invariant under involution. For example,

$$m_1(\neg\neg D) = m_2(\neg D) = m_1(D).$$

We extend the definition as follows:

$$m_i(D_1 \sqcap ... \sqcap D_n) = g_{i,n}(m_i(D_1), ..., m_i(D_n))$$

$$m_i(D_1 \sqcup ... \sqcup D_n) = g_{j,n}(m_i(D_1), ..., m_i(D_n))$$

where $j \neq i$.

By observing the subscripts i and j, we can see that these definitions already guarantee that the measures m_1 and m_2 are invariant under De Morgan's laws. For example

$$\begin{aligned} m_1(\neg(D_1 \sqcup D_2)) &= m_2(D_1 \sqcup D_2) \\ &= g_1(m_2(D_1), m_2(D_2)) \\ &= g_1(m_1(\neg D_1), m_1(\neg(D_2)) \\ &= m_1(\neg D_1 \sqcap \neg D_2). \end{aligned}$$

To ensure invariance under commutativity and associativity, we require functions which satisfy

$$g_{i,3}(x, y, z) = g_{i,2}(x, g_{i,2}(y, z))$$

and

$$g_{i,2}(x, y) = g_{i,2}(y, x),$$

for variables x, y and z. Provided that

$$g_{i,2}(x, g_{j,2}(y, z)) = g_{j,2}(g_{i,2}(x, y), g_{i,2}(x, z)),$$

we have invariance under distributivity laws. Finally, consider the absorption laws. For invariance, we need

$$g_{i,2}(x, g_{j,2}(x, y)) = g_{i,1}(x).$$

This set of conditions on the n-ary functions $g_{1,n}$ and $g_{2,n}$ are provided by the choices $g_{1,n} = max$, $g_{2,n} = min$ on numerical parameters and $g_{1,n} = \cup$, $g_{2,n} = \cap$ on set parameters. For example, for invariance under distributivity, we have $max(x, min(y, z)) = min(max(x, y), max(x, z))$, for integers x, y, z.

Of course, this recursive definition of measures m_1 and m_2 is incomplete without specification of a base case which defines $m_i(d)$ where d is a unitary diagram. Deriving $m_i(d)$ from the contours of the unitary diagram, for example, provides a pair of measures m_1 and m_2 which are invariant under all the logic rules, but which are variant under the reasoning rules Add Contour and Delete Contour.

In the following section, for each pair of measures, we will assume that the above recursive definition holds unless stated, and give only information about base cases. We also state how to combine the measures m_1 and m_2 to get a contribution to the heuristic function between D_1 and D_2.

B. Measure and metric definitions

*1) **Contours**:* Here we define two measures which are invariant under all logic rules but variant under the Add Contour and Delete Contour rules. These measures will be used to detect differences in the contour sets. The definition follows recursively as in section IV-A, with n-ary functions $g_{1,n} = \cup$ and $g_{2,n} = \cap$. The base cases are provided by $m_1(d) = m_2(d) = \{$the labels of the contours of $d\}$, where d is unitary. Note that $m_1(D) = m_2(D)$ holds for unitary D, but need not hold for compound D. That is, the measures m_1 and m_2 are not equal. For example, in Fig. 6, $m_1(d_1) = m_2(d_1) = \{A\}$, $m_1(\neg(d_2 \sqcap d_3)) = \emptyset$ and $m_2(\neg(d_2 \sqcap d_3)) = \{A, B, C\}$. If there is a contour label in $m_i(D_2)$ but not in $m_i(D_1)$ then

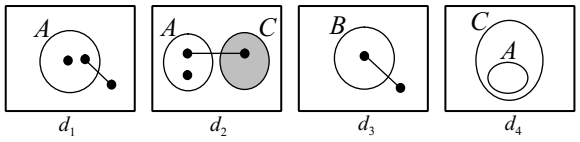

Fig. 6. Illustrating the measures.

we need to apply a reasoning rule to add that contour to D_1. Moreover, for each $C \in m_i(D_2) - m_i(D_1)$ we need to apply a reasoning rule to add C when we transform D_1 into D_2. That is, we need to add at least $|m_i(D_2) - m_i(D_1)|$ contours to D_1. Define for diagrams D_1, D_2, $i \in \{1, 2\}$:

$$AddC_i(D_1, D_2) = |m_i(D_2) - m_i(D_1)|$$

$$RemC_i(D_1, D_2) = |m_i(D_1) - m_i(D_2)|.$$

Combine these to give

$$CDiff_i(D_1, D_2) = AddC_i(D_1, D_2) + RemC_i(D_1, D_2).$$

Finally, the **contour difference metric** between diagrams D_1 and D_2 is defined to be

$$CM(D_1, D_2) = max\{CDiff_1(D_1, D_2), CDiff_2(D_1, D_2)\}$$

For the diagrams in Fig. 6, $CM(d_1 \sqcap d_3, d_4) = max\{2, 2\} = 2$ and $CM(\neg d_1, d_1 \sqcap d_3 \sqcap d_4) = max\{2, 1\} = 2$. We take the maximum because, for example, one application of the Add Contour rule can contribute to both $CDiff_1$ and $CDiff_2$ (we cannot take the sum $CDiff_1 + CDiff_2$). For example, if we introduce B to diagram d_4 in Fig. 6, yielding diagram d_4' in Fig. 7 then $CM(d_4, d_4') = max\{1, 1\} = 1$. The sum $1 + 1$ would not provide a lower bound on the length of a shortest proof from d_4 to d_4'.

*2) **Zones**:* We will define metrics that detect differences in the zone sets, using $g_{1,n} = \cup$ and $g_{2,n} = \cap$. The base cases are provided by $m_1(d) = m_2(d) = \{$the zones of $d\}$ where d is unitary. Note, again, that $m_1(D) = m_2(D)$ holds for unitary D, but need not hold for compound D. This will also be the case for the remaining measures we define with one exception. Before calculating the zone metrics for the heuristic function, we need to ensure that the unitary components of the premise and conclusion diagrams have the same contour sets. It has been argued in [3] why this is needed for unitary diagrams, and the same reasoning applies for compound diagrams. We apply the Add Contour rule to all unitary components of D_1 to make a new diagram, $CForm(D_1, D_2)$, in which each unitary diagram includes all contour labels from D_1 and D_2 (this being the union of the sets of contour labels of their unitary components). Similarly, we make a new diagram $CForm(D_2, D_1)$ by applying the Add Contour rule to D_2. For example, in Fig. 6, $CForm(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4)$ is $d'_1 \sqcap \neg(d'_2 \sqcup d'_3)$, shown in Fig. 7, obtained by adding contours to each unitary component d_1, d_2 and d_3. Similarly, $CForm(d_4, d_1 \sqcap \neg(d_2 \sqcup d_3))$ is d'_4. Define for diagrams D_1, D_2, $i \in \{1, 2\}$:

$$AddZ_i(D_1, D_2) = \begin{cases} 1 & \text{if } m_i(CForm(D_2, D_1)) \not\subseteq \\ & \quad m_i(CForm(D_1, D_2)) \\ 0 & \text{otherwise} \end{cases}$$

$$RemZ_i(D_1, D_2) = \begin{cases} 1 & \text{if } m_i(CForm(D_1, D_2)) \not\subseteq \\ & \quad m_i(CForm(D_2, D_1)) \\ 0 & \text{otherwise.} \end{cases}$$

The capping of $AddZ_i$, and $RemZ_i$ is similar to the capping

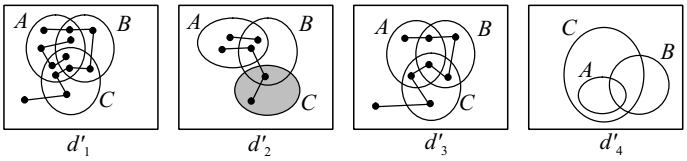

Fig. 7. Contour forms.

applied to $AddZone$ and $RemZone$ in the case of unitary diagrams [3]. This is due to the fact that a single application of either Add Shaded Zone or Delete Shaded Zone to D_1 can change the zone set in $CForm(D_1, D_2)$ by more than one zone. For example, in Fig. 8, we can add one zone to d_1, giving d_3 but $CForm(d_1, d_2)$ has two fewer zones than $CForm(d_3, d_2)$. We define two metrics (that we will use to define the zone difference metric) between diagrams D_1 and D_2 to be

$$AddZ(D_1, D_2) = max\{AddZ_1(D_1, D_2), AddZ_2(D_1, D_2)\}$$

$$RemZ(D_1, D_2) = max\{RemZ_1(D_1, D_2), RemZ_2(D_1, D_2)\}.$$

The reason for taking the maximum (as opposed to the sum) is that, for example, applying the rule Delete Shaded Zone can affect both $AddZ_1$ and $AddZ_2$ simultaneously (and similarly, $RemZ_1$ and $RemZ_2$). We define the **zone difference metric**

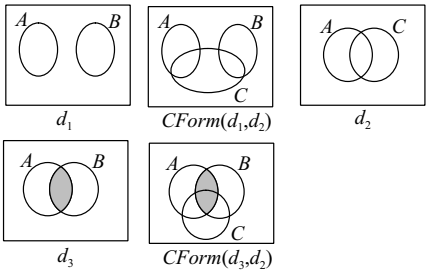

Fig. 8. Illustrating the zone measure capping.

$$ZM(D_1, D_2) = AddZ(D_1, D_2) + RemZ(D_1, D_2).$$

As an example, in Fig. 6, $ZM(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4) = 1+1 = 2$.

*3) **Shading**:* We will now define metrics that detect differences in the shading, using $g_{1,n} = \cup$ and $g_{2,n} = \cap$. The base cases are provided by $m_1(d) = m_2(d) = \{$the shaded zones of $d\}$ where d is unitary. Before calculating the shaded zone difference metric for the heuristic function, we need to ensure that the unitary components of the premise and conclusion diagrams have the same zone sets. It has been argued in [3] why this is needed for unitary diagrams, and the same reasoning applies for compound diagrams. We take the unitary components of diagram $CForm(D_i, D_j)$ and add shaded zones until they are in Venn form (every possible zone is present, given the contour label set), giving $Venn(CForm(D_i, D_j))$. Shown in Fig. 9 are the unitary components of $Venn(CForm(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4) = d''_1 \sqcap \neg(d''_2 \sqcup d_3)$ and $Venn(CForm(d_4, d_1 \sqcap \neg(d_2 \sqcup d_3)) = d''_4$, where d_1, d_2, d_3 and d_4 are in Fig. 6. Define for diagrams D_1, D_2,

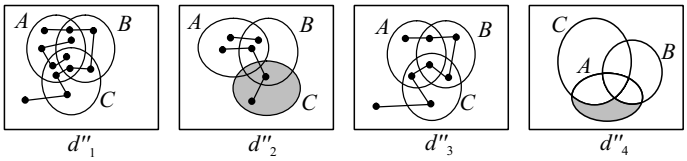

Fig. 9. Venn forms.

$i \in \{1, 2\}$:

$$AddSh_i(D_1, D_2) = \begin{cases} 1 & \text{if } m_i(Venn(CForm(D_2, D_1))) \\ & \quad \not\subseteq m_i(Venn(CForm(D_1, D_2))) \\ 0 & \text{otherwise} \end{cases}$$

$$RemSh_i(D_1, D_2) = \begin{cases} 1 & \text{if } m_i(Venn(CForm(D_1, D_2))) \\ & \quad \not\subseteq m_i(Venn(CForm(D_2, D_1))) \\ 0 & \text{otherwise.} \end{cases}$$

We define two metrics (that we will use to define the shading difference metric) between diagrams D_1 and D_2 to be:

$$AddSh(D_1, D_2) = max\{AddSh_1(D_1, D_2), AddSh_2(D_1, D_2)\}$$

$$RemSh(D_1, D_2) = max\{RemSh_1(D_1, D_2), RemSh_2(D_1, D_2)\}.$$

We combine these to give the **shading difference metric**

$$ShM(D_1, D_2) = AddSh(D_1, D_2) + RemSh(D_1, D_2).$$

For example, $ShM(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4) = 1 + 1 = 2$, where d_1, d_2, d_3 and d_4 are in Fig. 6.

*4) **Spiders**:* We now define metrics which detect differences in the spiders, using $g_{1,n} = \cup$ and $g_{2,n} = \cap$. The base cases are provided by $m_1(d) = m_2(d) = Sp(d)$ where

$$Sp(d) = \{(i, r) : 1 \leq i \leq n \text{ where } n \text{ is the number of spiders whose habitat is the region } r \text{ in } d\},$$

and d is unitary. For example, in Fig. 6,

$m_1(d_1) = \{(1, \{A\}), (2, \{A, U - A\}), (1, \{A, U - A\})\}$.

Informally, then, a spider is a pair, (i, r) and i indicates that (i, r) is the i^{th} spider inhabiting r. The set $m_1(d_1)$ includes one such pair for each spider in d_1. So, if there are no spiders inhabiting r in d_1 then this is represented by the absence of any element in $m_1(d_1)$. Define for diagrams D_1, D_2, $i \in \{1, 2\}$:

$$AddSp_i(D_1, D_2) = |m_i(CForm(D_2, D_1)) - m_i(CForm(D_1, D_2))|$$

$$RemSp_i(D_1, D_2) = |m_i(CForm(D_1, D_2)) - m_i(CForm(D_2, D_1))|.$$

We define two metrics (that we will use to define the spider difference metric) between diagrams D_1 and D_2 to be:

$$AddSp(D_1, D_2) = max\{AddSp_1(D_1, D_2), AddSp_2(D_1, D_2)\}$$

$$RemSp(D_1, D_2) = max\{RemSp_1(D_1, D_2), RemSp_2(D_1, D_2)\}.$$

Note here that, for example, a single application of the Excluded Middle rule can impact both $AddSp$ and $RemSp$. Moreover, the rule Split Spider can introduce two new spiders. Thus we define the **spider difference metric**

$$SpM(D_1, D_2) = max\{AddSp(D_1, D_2), RemSp(D_1, D_2)\}/2.$$

For example, $SpM(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4) = max(0, 4)/2 = 2$, where d_1, d_2, d_3 and d_4 are in Fig. 6.

*5) **Connectives**:* In this section we define metrics which detect differences in the connectives, using $g_{1,n} = max$ and $g_{2,n} = min$. The base cases are provided by, for unitary d,

$$m_1(d) = m_2(d) = 0.$$

For these measures we over-ride part of the generic definition of the measures. As usual, use

$$m_i(D_1 \sqcap ... \sqcap D_n) = g_i(m_i(D_1), ..., m_i(D_n))$$

$$m_i(D_1 \sqcup ... \sqcup D_n) = g_j(m_i(D_1), ..., m_i(D_n))$$

(where $j \neq i$) but, where possible, over-ride this definition with, for $i = 1, 2$

$$m_i(d_1 \sqcap D_2 \sqcap ... \sqcap D_n) = 1 + m_i(D_2 ... \sqcap D_n)$$

where d_1 is a unitary diagram. One effect of this is to contrast $m_1(d_1) = 0$ with $m_1(d_1 \sqcap d_1) = 1$. Application of the idempotency rules can increase or decrease these measures, almost doubling or halving their value. For this reason, we use log_2 to create measures which count potential rule applications. Other rules, such as Excluded Middle, can increase the measures from 0 to 1.

The two metrics (that we will use to define the connective difference measure) between diagrams D_1 and D_2 are defined to be, for $i = 1, 2$, in the case when $m_i(D_1), m_i(D_2) > 0$

$$CnnM_i(D_1, D_2) = |log_2(m_i(D_1)) - log_2(m_i(D_2))|,$$

and in the case when $m_i(D_1) = 0$ and $m_i(D_2) > 0$

$$CnnM_i(D_1, D_2) = 1 + log_2(m_i(D_2)),$$

and in the case when $m_i(D_1) > 0$ and $m_i(D_2) = 0$

$$CnnM_i(D_1, D_2) = 1 + log_2(m_i(D_1)),$$

otherwise we define

$$CnnM_i(D_1, D_2) = 0.$$

A single application of an idempotency rule can affect both $CnnM_1$ and $CnnM_2$ simultaneously, so to prevent multiple-counting of these rule applications, the contribution to the heuristic function is the maximum of $CnnM_1$ and $CnnM_2$. We define the **connective difference metric** to be

$$CnnM(D_1, D_2) = max\{CnnM_1(D_1, D_2), CnnM_2(D_1, D_2)\}.$$

For example, $CnnM(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4) = max(2, 0) = 2$.

*6) **Not metric**:* In this section we define measures which detect differences in the numbers of 'nots', using $g_{1,n} = max$ and $g_{2,n} = min$. The base cases are provided by, for unitary d, $m_1(d) = 0$ and $m_2(d) = 1$. For these measures we over-ride part of the generic definition of the measures. Instead of $m_i(\neg D) = m_j(D)$ we define, for non-unitary diagrams D

$$m_i(\neg D) = 1 + m_j(D).$$

One effect of this is to contrast $m_2(\neg\neg d) = 2$ with $m_2(d) = 0$. Application of the involution rule can increase or decrease these measures by 2. For this reason, we use half their value before we evaluate their contribution to the heuristic function.

Define the **not difference metric** between diagrams D_1 and D_2 to be

$$NM(D_1, D_2) = \frac{max\left\{\begin{array}{l} |m_1(D_1) - m_1(D_2)|, \\ |m_2(D_1) - m_2(D_2)| \end{array}\right\}}{2}$$

For example, $NM(d_1 \sqcap \neg(d_2 \sqcup d_3), d_4) = 1$. The reason for taking the maximum is that, for example, the excluded middle rule can impact m_1 and m_2 simultaneously. The reason for dividing by two is that a single application of the involution rule can increase m_1 and m_2 by two (and its reverse subtract two).

C. Compound heuristic

Define the **compound diagram heuristic**, H, between D_1 and D_2 to be the sum

$$H(D_1, D_2) = CM(D_1, D_2) + ZM(D_1, D_2) + NM(D_1, D_2) + max\{ShM(D_1, D_2), SpM(D_1, D_2), CnnM(D_1, D_2)\}.$$

Note that we take the maximum of the shading metric, spider metric and connective metric because, for example, a single application of one of the rules Excluded Middle and Split Spider can affect all these measures simultaneously.

We generated a random sample of 500,000 pairs of diagrams for which the heuristic function was optimistic. We conjecture that the heuristic function is admissible.

V. Implementation and evaluation

We have implemented this heuristic search as part of a spider diagram reasoning tool. The search can either stop when a proof is found, or seek the set of all optimal proofs. The application keeps a record of the number of proof attempts stored during the search. An initial comparison of the effectiveness of the heuristic was conducted by building random proofs (within small but arbitrary limits on complexity) and searching for the proofs using a breadth first search (zero heuristic) as compared to the heuristic outlined in this paper. The benefits gained are assessed by considering the data set of ratios (number of proof attempts with our heuristic)/(number of proof attempts with breath first search).

The number of proof attempts with the zero heuristic ranged from 34 to 443,000, and with our heuristic, ranged from 15 to 270,000. We collected data for 178 random proofs. The ratio of numbers of proof attempts ranged from 1 (where the zero heuristic searches the same space as our heuristic) to 0.004 (where our heuristic vastly reduces the search space size). The median ratio was 0.184, an 81.6% reduction in the size of the search space. More spectacular results were obtained for longer proofs. Further work is needed to establish why, in some cases, our heuristic gives no saving in the size of the search space.

VI. Conclusion

In this paper, we have demonstrated how a heuristic A^* approach can be used to automatically generate shortest proofs in a spider diagram reasoning system. We regard this as an important step towards generating readable proofs. Our work can be extended in a number of ways. The cost element of the evaluation function can be altered to incorporate factors that impact readability. For example:

- Comprehension of rules. There may be a difference in how difficult each rule is to understand. We can model a difference in the relative difficulty of rules by assigning different costs. Currently, we are conducting an experiment to determine the relative understandability of the rules.
- Drawability of diagrams. As discussed in [6] not all diagrams are drawable, subject to some well-formed conditions. We can increase the cost of a rule application if the resulting diagram is not drawable.

Another extension of this work is to include further reasoning rules. The rule set in this paper forms part of a sound and complete set. However, enlarging the collection of reasoning rules available to the heuristic proof writer may affect the admissability of the heuristic function. Moreover, using additional rules enlarges the search space. Even if the heuristic function is admissible with the addition of a further reasoning rule, it may be the case that the heuristic function becomes less effective because the search space is larger. However, the benefit of adding further rules is that there will be more cases where proofs can be found: if $D_1 \models D_2$ and all proofs from D_1 to D_2 require a rule that we have excluded then, currently, no proof will be found.

In addition to its use for automatic theorem proving, our heuristic function can also be used to support interactive proof writing. It can advise the user on the probable implications of applying a rule (for example "Adding contour B will decrease the contour difference measure, so might be a good idea"). Possible applications of rules could be annotated with their impact on the heuristic value. The user could collaborate with the tool to solve complex problems.

Acknowledgment Gem Stapleton thanks the UK EPSRC for support under grant number 01800274. Jean Flower was partially supported by the UK EPSRC grant GR/R63516. Thanks also to Andrew Fish for his comments on earlier drafts of this paper.

References

[1] R. Dechter, and J. Pearl. Generalized best-first search strategies and the optimality of A^*. *Journal of the Association for Computing Machinery, 32*, 3, pages 505-536. 1985.

[2] J. Flower, and G. Stapleton. Automated Theorem Proving with Spider Diagrams. In proceedings of Computing: The Australasian Theory Symposium (CATS 04), pages 116-132, ENTCS,Science Direct, 2004.

[3] J. Flower, and J. Masthoff, and G. Stapleton. Generating Readable Proofs: A Heuristic Approach to Theorem Proving with Spider Diagrams. In proceedings of Diagrams 2004, pages 166-181, Springer-Verlag, 2004.

[4] J. Gil, and J. Howse, and S. Kent. Formalising spider diagrams. In *Proceedings of IEEE Symposium on Visual Languages (VL99)*, pages 130–137. IEEE Computer Society Press, 1999.

[5] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Science and Cybernetics*, 4, 2, pages 100-107, 1968.

[6] J. Flower, and J. Howse. Generating Euler Diagrams. Proceedings of Diagrams 2002, pages 61–75, Springer-Verlag, 2002

[7] J. Howse, F. Molina, and J. Taylor. SD2: A sound and complete diagrammatic reasoning system. In *Proc. IEEE Symposium on Visual Languages (VL2000)*, IEEE Computer Society Press, 2000.

[8] J. Howse, G. Stapleton, and J. Taylor. Reasoning with Spider diagrams. Avaiable from www.cmis.brighton.ac.uk/research/vmg, 2004.

[9] G.F. Luger. Artificial intelligence: Structures and strategies for complex problem solving. Fourth Edition. Addison Wesley: 2002.

[10] F. Molina. *Reasoning with extended Venn-Peirce diagrammatic systems*. PhD thesis, University of Brighton, 2001.

[11] S.-J. Shin. *The Logical Status of Diagrams*. Camb. Uni. Press, 1994.