

A Case Base Similarity Framework*

Hugh R. Osborne and Derek G. Bridge

University of York

Abstract. Case based systems typically retrieve cases from the case base by applying similarity measures. The measures are usually constructed in an ad hoc manner. This paper presents a theoretical framework for the systematic construction of similarity measures. In addition to paving the way to a design methodology for similarity measures, this systematic approach facilitates the identification of opportunities for parallelisation in case base retrieval.

1 Case Memory Systems

In this paper we present a framework for the construction of similarity measures. Great flexibility is achieved by constructing complex similarity measures from more basic measures using a variety of connectives that we define. The concepts introduced in this paper are illustrated by an extensive example in the appendix.

A case memory system will be considered to consist of a case base and a retrieval mechanism. The case base will be modelled as a finite set, Θ , of cases, equipped with *projection* functions for accessing the component elements of these cases. While the cases in the example in the appendix are all tuples, and the projection functions the standard projection functions for tuples, a case may be a more complex structure, with correspondingly more complex projection functions. The projection functions might even implement considerable inferencing [9, 14], perhaps to obtain “deep” features [3] from “surface” features.

A retrieval request is presented to the system as a pair, consisting of an element, ϑ , of Θ and a similarity measure, σ . The case ϑ , known as the *seed* will, in combination with the similarity measure, represent the “best possible” case. This is in contrast to the earliest approaches, e.g. [11], in which the seed was the ideal case, and the similarity measure measured the closeness of retrieved cases to this ideal. In the approach taken here, the similarity measure can, for example, include negation, so that distance from, rather than closeness to, the seed becomes the measure of suitability.

We take a very general view of what cases are. The problem description, its wider situation or context, its solution, the solution’s outcome, etc. may all be features that may be projected from a case. In some case memory implementations, only a subset of these might be stored directly as fields of the cases; others

* In: *Proceedings of the 3^d European Workshop on Case-Based Reasoning, EWCBR’96*, Advances in Case-Based reasoning, Ian Smith and Boi Faltings (Eds.), Lecture Notes in Artificial Intelligence 1168, Springer Verlag, 1996

might be part of an indexing structure (as, e.g., with the explanations of case applicability in [5]). However, even in these systems, the information has to be associated with the case in some fashion, and so we can, without loss of generality, assume that the information can be obtained by applying a projection function to a case.

By taking this broad view of what cases are and by allowing similarity measures to apply to any of the features that can be projected from the case, our framework also encompasses proposals that retrieval should be sensitive to aspects of the case other than the problem description (e.g. the adaptability of the solution, as in [12, 24]). If, on the other hand, only problem descriptions are to be compared, σ will be designed to ignore other features.

Finally, we should note that, in systems in which the case memory is indexed, case base interrogation is often a two-stage process [13, 1]: a retrieval step exploits the indexes to restrict computational effort to cases that are similar to the seed on those characteristics encoded as indexes, but the final ranking and case selection requires application of a similarity measure to this retrieved set of cases.

There is a sense in which this two-step process is equivalent to the application of two similarity measures: one that is “hard-coded” as indexes, and one that is then applied to the results of the application of the first. From this point of view, our framework encompasses systems of this kind.

In passing, we note that at stake here is whether to take a *representational* or a *computational* view of similarity [19], or, in Richter’s terminology [21], whether the similarity measure is *compiled* knowledge or knowledge that is *interpreted* at run-time. In the representational approach, cases reside in a data structure, such as a DAG, where, e.g., proximity in the data structure denotes similarity. Representational approaches can afford considerable efficiency in retrieval. The data structure is effectively optimised towards retrieval according to the “hard-coded” similarity measure. This can be of especial value when similarity assessment requires the application of large bodies of domain-specific knowledge: that knowledge will be applied once per case at case base update time, rather than being applied afresh on every retrieval [19]. However, this form of optimisation can lead to a loss of flexibility [15] as it may be hard or inefficient to access the case base in different ways as might be needed to give more context-sensitivity or to use the case base for multiple tasks [4]. The computational approach, on the other hand, will, in its most extreme form, compute similarity “from scratch” on each retrieval. This can be a flexible approach as nothing is hard-coded; it may be more amenable to user manipulation of the similarity measure (as allowed in many case-based reasoning shells, e.g., [10]) or even manipulation through some learning process, e.g. [22]; but there may be an efficiency price to be paid. (A “spin-off” of our own work has been the identification of opportunities for parallelisation in pure computational approaches using our similarity framework [17], and this may help to make computational approaches more widely usable. See also [16].) The two-stage process mentioned above is clearly a compromise between pure representational and pure computational approaches. We repeat that

(while our implementational work has focused on computational approaches) our framework is general enough to cover the full spectrum of possibilities.

The first half of this paper — Sect. 2 — will discuss “ordinal” similarity measures, both defining simple “atomic” similarity measures and introducing methods of combining these to form more complex measures. A model demonstrating some of the results from Sect. 2 has been implemented [17, 18]. Section 3 will then consider how the results from Sect. 2 can be applied to “cardinal” similarity measures, and how both types of measure may be combined. Finally Sect. 4 draws some conclusions.

2 Ordinal Similarity Measures

This section presents a repertoire of comparison operations, or similarity measures, and operations on these similarity measures used to construct new similarity measures. These similarity measures are *ordinal*, i.e. they are symbolic and do not give a numeric value for cases, but order them in terms of their similarity to a seed, i.e. a similarity measure is a function from (features of) cases to partial orders over (features of) cases: $\sigma :: \Theta \rightarrow \Theta \rightarrow \Theta \rightarrow \mathbf{bool}$. Cardinal similarity measures, giving numeric “scores”, will be considered in Sect. 3. In this section, Sects. 2.1 and 2.2 will discuss the construction of similarity measures for individual features of cases, while Sects. 2.3, 2.4 and 2.5 will show how to combine these to form more complex similarity measures for complete cases.

2.1 Atomic Similarity Measures

Some standard similarity measures are defined in Fig. 1, and illustrated by an example applied to the set $\{1, 2, 3, 4, 5\}$, with the usual total order \leq . The inverse function, $\sigma^{-1} \vartheta x y = \sigma \vartheta y x$, is also defined for similarities, where the inverse of a similarity σ returns the inverse of the order returned by σ .

2.2 Orders from Other Structures

The orders above (with the exception of `flat`) were all based on an existing ordering. The general approach was to define a function (e.g. `is`) which given a *seed* (e.g. ϑ) would return an ordering (e.g. `is` ϑ). This section will discuss the derivation of orderings from other structures. A similar approach will be taken — functions will be defined to generate orders from seeds. These functions will make use of auxiliary functions, again applied to seeds, mapping elements of the domain to some ordered set — e.g. \mathbf{N} . These auxiliary functions will reflect some notion of distance from the seed, and will usually be written “ \dashrightarrow ”.

Trees. A tree in which only leaves contain elements is defined by: `Tree elem ::= Leaf elem | Node [Tree]`. The distance of an element from a seed can be defined to be the depth of that element in the smallest subtree containing both the

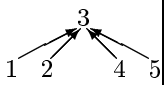
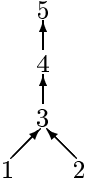
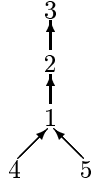
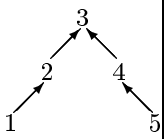
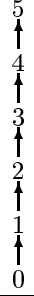
flat: No elements are related	flat 3	is: The seed is better than all others	is 3
$\frac{x \text{ (flat } \vartheta) y}{x = y}$	1 2 3 4 5	$\frac{x \text{ (is } \vartheta) y}{(y = \vartheta)}$ \vee $(x = y)$	
minimal: The seed is a minimum requirement	minimal 3	maximal: The seed is a maximum requirement	maximal 3
$\frac{x \text{ (minimal } \vartheta) y}{(x \geq y \wedge x \geq \vartheta)}$ \vee $(x = y)$		$\frac{x \text{ (maximal } \vartheta) y}{(x \geq y \wedge x \leq \vartheta)}$ \vee $(x = y)$	
best: The seed is best	best 3	id: Ignore the seed	id 3
$\frac{x \text{ (best } \vartheta) y}{(y \leq x \leq \vartheta)}$ \vee $(y \geq x \geq \vartheta)$		$\frac{x \text{ (id } \vartheta) y}{x \geq y}$	

Fig. 1. Atomic similarity measures

element and the seed. The range of the distance function is the ordered set $\mathbf{N}_\infty = \mathbf{N} \cup \{\infty\}$. The definition of the distance function $\dashv\rightarrow$ makes use of three other functions: depth (giving the depth of an element in a tree), \sqsubset_{Tree} (the subtrees of a tree), and \in_{Tree} (which tests if an element appears in a tree). These functions are defined in Fig. 2. Since $\vartheta (\dashv\rightarrow t)$ clearly defines a function from elements to the ordered set \mathbf{N}_∞ , it can be used to define a similarity measure generator for trees, \sqsubset_{Tree} , also given in Fig. 2.

Directed Acyclic Graphs. The functions in Fig. 2 have been defined in such a way that they can easily be adapted to apply to DAGs. Details can be found in [18]. The reader should note that graphs are being used in this paper to define distances between elements and seeds. This is quite distinct from assessing the similarity of two graph structures by some sort of subgraph algorithm, as is

$\text{depth} :: \text{elem} \rightarrow \overline{\text{Tree}} \text{ elem} \rightarrow \mathbb{N}_\infty$
$\text{depth } e \text{ (Leaf } l) = 0, \text{ if } e = l$ $= \infty, \text{ otherwise}$
$\text{depth } e \text{ (Node } n) = 1 + (\min \{\text{depth } e \ t \mid t \in n\})$
$\text{C}_{\text{Tree}} :: \overline{\text{Tree}} \text{ elem} \rightarrow \{\overline{\text{Tree}} \text{ elem}\}$
$\text{C}_{\text{Tree}} \text{ (Leaf } l) = \{\text{(Leaf } l)\}$
$\text{C}_{\text{Tree}} \text{ (Node } n) = \{\text{(Node } n)\} \cup (\bigcup_{t \in n} \text{C}_{\text{Tree}} \ t)$
$\in_{\text{Tree}} :: \text{elem} \rightarrow \overline{\text{Tree}} \text{ elem} \rightarrow \mathbf{bool}$
$e \in_{\text{Tree}} \text{ (Leaf } l) = e = l$
$e \in_{\text{Tree}} \text{ (Node } n) = \exists t \in n : e \in_{\text{Tree}} \ t$
$\mapsto :: \overline{\text{Tree}} \text{ elem} \rightarrow \text{elem} \rightarrow \text{elem} \rightarrow \mathbb{N}_\infty$
$\vartheta (\mapsto \ t) \ e = \min \{\text{depth } e \ t' \mid t' \in (\text{C}_{\text{Tree}} \ t) \wedge \vartheta \in_{\text{Tree}} \ t'\}$
$\sqsubset_{\text{Tree}} :: \overline{\text{Tree}} \text{ elem} \rightarrow \text{elem} \rightarrow (\text{elem} \rightarrow \text{elem} \rightarrow \mathbf{bool})$
$e_1 (\sqsubset_{\text{Tree}} \ t \ \vartheta) \ e_2 = \vartheta (\mapsto \ t) \ e_1 < \vartheta (\mapsto \ t) \ e_2$

Fig. 2. Similarity measure generating functions for trees

found in many case based reasoning systems [2, 6]. There is, however, no reason why such an algorithm could not be used to define an ordering on graphs, and then apply this ordering in the way presented in this paper.

Graphs. A similar construction can be used for graphs in general, by defining the distance function to return the length of the shortest path between the seed and an element.

User Defined Types. The same method can be applied to user defined types. A metric should be defined giving the “closeness” of an element to a seed, and this can then be used to define an ordering on that type. Indeed this can be used to implement more representational approaches [19], with the user defined type being some representation of the positioning of a case in a structured case base. The similarity measure will then reflect the indexing of cases in the case base structure.

2.3 Boolean Connectives

Sections 2.1 and 2.2 presented a repertoire of similarity measures for individual features of a case. It is now necessary to consider how to combine these similarity measures to form more complex similarity measures for whole cases.

The first obvious candidates for combining orderings are the usual boolean operators. These are covered in this section. The section starts with a presentation of the application of boolean operators to construct complex similarity

measures from simpler ones. This is followed by a discussion of the determination of maxima from these similarity measures. It will then be shown how this can be done in parallel, by transforming similarity measures to a normal form.

Sections 2.4 and 2.5 will then introduce other methods of constructing more complex similarity measures, these being *filters*, *priorities* and *preferences*. These new connectives also allow a normal form to be determined, and the computation of the maxima to be executed in parallel.

Boolean Operators. The usual boolean operators (\wedge , \vee and \neg) may be applied to similarity measures to form new similarity measures. This is done by “lifting” the point-wise boolean operators to operate on similarities. If \oplus is a binary boolean operator then the lifted operator $\hat{\oplus}$, acting on similarities σ_1 and σ_2 , applied to seed ϑ and cases x and y , is defined by: $(\sigma_1 \hat{\oplus} \sigma_2) \vartheta x y = (\sigma_1 \vartheta x y) \oplus (\sigma_2 \vartheta x y)$.

Determining Maxima. Note that if \oplus is a boolean operator other than \wedge then, even if $\sigma_1 \vartheta$ and $\sigma_2 \vartheta$ are partial orders, $(\sigma_1 \hat{\oplus} \sigma_2) \vartheta$ is not necessarily a partial order. Since maxima are not defined for arbitrary relations it is necessary to extend the definition of maxima to do this.

The maxima of a partial order \sqsubset are usually defined as

Definition 1. $\sqcap \sqsubset S = \{x \in S | \forall y \in S : x \sqsubset y \Rightarrow x = y\}$.

The usual method for determining maxima of an arbitrary relation \oplus is to take the reflexive transitive closure, \oplus^* , of that relation, thus giving a pre-order, and then taking the maxima of the partial order over the equivalence classes generated by the equivalence relation $\otimes = \{(p, q) | p \oplus^* q \wedge q \oplus^* p\}$. A different approach will be taken here, generalising the concept of maxima to apply to arbitrary relations, avoiding the necessity of generating either the reflexive transitive closure or the equivalence classes.

Since, for a partial order, $(x \sqsubset y \Rightarrow x = y) \equiv (x \sqsubset y \Rightarrow y \sqsubset x)$, Def. 1 is equivalent to

Definition 2. $\sqcap \sqsubset S = \{x \in S | \forall y \in S : x \sqsubset y \Rightarrow y \sqsubset x\}$.

and this will be taken as the definition of the “maxima” of any relation, i.e. ‘ \sqsubset ’ in Def. 2 may be any arbitrary relation.

Definition (2) has the following properties:

Property 1 $\sqcap (\hat{\neg} \oplus) = \sqcap \oplus^{-1}$,

Property 2 $\sqcap (\oplus^1 \hat{\vee} \oplus^2) \supseteq (\sqcap \oplus^1) \hat{\wedge} (\sqcap \oplus^2)$.

Property (1) states that the maxima of the negation of a relation are equivalent to the maxima of the inverse of the relation. This can be useful, since the inverse of a partial order is a partial order, while the negation is not. Property (2)

ensures that the intersection of the maxima of two relations will be an acceptable approximation of the maxima of the disjunction of those relations.

These, and other properties, given in this paper are stated without proof. Proofs are given in [18].

A sufficient condition for the inclusion in Prop. 2 to be an equality is that the two relations involved have a degree of consistency in their inverses. If x is less than y in the first ordering, and greater than y in the second, then it must also be greater than y in the first, and vice versa. I.e. if, for all x and y in S :

$$(x \oplus^1 y \wedge y \oplus^2 x \Rightarrow y \oplus^1 x) \wedge (x \oplus^2 y \wedge y \oplus^1 x \Rightarrow y \oplus^2 x) \quad (1)$$

then

$$\sqcap (\oplus^1 \hat{\vee} \oplus^2) S = ((\sqcap \oplus^1) \hat{\cap} (\sqcap \oplus^2)) S .$$

Since this condition holds if $\oplus^1 \hat{\vee} \oplus^2$ is a partial order, then, in this case, the intersection of the maxima will be the maxima of the disjunction.

These results can be applied to determine the maxima of a relation constructed by application of the boolean operators. This can be done by taking the disjunctive normal form of the boolean expression and determining, in parallel, the maxima of the constituent terms of the normal form, and then taking the intersection.

2.4 Filters

Another possibility is to first “filter” the set through some predicate before applying the maximising function. Normally a filter will take a predicate, and when applied to a set will give a subset of that set. In keeping with the approach taken in the rest of this paper a filter here will take a *relation* over a type, and apply it to a seed to give the predicate that will be applied to a set. The symbol “ \triangleleft ” will be used for filters.

$$\begin{aligned} \triangleleft &:: (\tau \rightarrow \tau \rightarrow \mathbf{bool}) \rightarrow \tau \rightarrow \{\tau\} \rightarrow \{\tau\} \\ \triangleleft \oplus \vartheta S &= \{x \in S \mid x \oplus \vartheta\} \end{aligned}$$

Filters can be used to express concepts such as “only when” and “except when”. Filters can also be used to construct more complex preferences. A feature of a case may be a set of constituents. Filters can then be used to select cases containing a minimum (or maximum) set of constituents, to eliminate cases containing (or not containing) some specific constituent, or even, in combination with the operators given in Sect. 2.1, to order cases according to the closeness of their list of constituents to some ideal.

Filters can be expressed as similarity measures. Given a relation \oplus that is to be applied in a filter, it is possible to define a similarity measure σ_{\oplus} such that, except for one special case, $\triangleleft \oplus = \sqcap \sigma_{\oplus}$. The exception is when $\triangleleft \oplus S = \emptyset$, in which case $\sqcap \sigma_{\oplus} S = S$.

Property 3 *Let \oplus be any binary relation. Define σ_{\oplus} by $\sigma_{\oplus} \vartheta x y = y \oplus p$. Then $\triangleleft \oplus = \sqcap \sigma_{\oplus}$.*

Applying σ_{\oplus} makes it possible to combine filters with similarities by applying the two following properties:

Property 4 $(\triangleleft \oplus) \hat{\cdot} (\sqcap \sigma) = \sqcap (\sigma_{\oplus} \hat{\vee} \sigma)$,

Property 5 $(\sqcap \sigma) \hat{\cdot} (\triangleleft \oplus) = \sqcap (\sigma_{\oplus} \hat{\vee} \sigma \hat{\wedge} \sigma_{\oplus})$.

2.5 Priorities and Preferences

Another possible type of connective is one which will take one similarity measure as being more significant than another. There are two possible approaches to this. The first applies to the similarity measures themselves, the second to the process of determining maxima. The first of these will be referred to as a *priority* (after [23]), the second as a *preference* (after [8]).

Priorities. The *prioritisation* of relation \oplus^1 over relation \oplus^2 , notation $\oplus^1 \gg \oplus^2$, is a generalisation of lexicographic ordering defined for relations, and is the relation defined by:

Definition 3. $x (\oplus^1 \gg \oplus^2) y = (x \oplus^1 y \wedge \neg(y \oplus^1 x)) \vee (x \oplus^1 y \wedge y \oplus^1 x \wedge x \oplus^2 y)$.

The two terms $(x \oplus^1 y \wedge \neg(y \oplus^1 x))$ and $(x \oplus^1 y \wedge y \oplus^1 x \wedge x \oplus^2 y)$ in this disjunction satisfy (1), since both antecedents in this condition will be false. As a consequence, the intersections of the maxima of the two terms will be equal to the maxima of the the prioritisation.

A prioritisation of similarity measures is a prioritisation of relations “lifted” to similarity measures:

$$(\sigma_1 \gg \sigma_2) p = (\sigma_1 p) \gg (\sigma_2 p) .$$

Property 6 *When taking maxima of priorities the first term $(x \oplus^1 y \wedge \neg(y \oplus^1 x))$ may be replaced by $x \oplus^1 y$, since $x \oplus^1 y \wedge \neg(y \oplus^1 x) \Rightarrow y \oplus^1 x \wedge \neg(x \oplus^1 y)$ is equivalent to $x \oplus^1 y \Rightarrow y \oplus^1 x$.*

The prioritisation of similarity σ_1 over similarity σ_2 will therefore be defined as:

Definition 4. $\sigma_1 \gg \sigma_2 = \sigma_1 \hat{\vee} (\sigma_1 \hat{\wedge} \sigma_1^{-1} \hat{\wedge} \sigma_2)$.

thus avoiding the need for negation.

Property 7 *Prioritisation distributes to the right over disjunction and, when taking maxima, if the two similarity measures being disjoined satisfy (1), also to the left.*

Preferences. An alternative approach is to first select maxima for the first similarity measure, and then take the maxima over these according to the second — i.e. the second similarity measure is applied only to discriminate between the maxima of the first. The *preference* of similarity measure σ_1 over similarity measure σ_2 is defined by:

Definition 5. $\sqcap (\sigma_1 \triangleright \sigma_2) = (\sqcap \sigma_2) \cdot (\sqcap \sigma_1)$.

Relating Priorities and Preferences. Priorities and preferences satisfy the following, for σ_1, σ_2 pre-order generating similarities:

Property 8 $(\sqcap \sigma_1) \hat{\sqcap} (\sqcap \sigma_2) \vartheta S \subseteq \sqcap (\sigma_1 \triangleright \sigma_2) \vartheta S \subseteq \sqcap (\sigma_1 \gg \sigma_2) \vartheta S \subseteq \sqcap \sigma_1 \vartheta S$.

3 Cardinal Similarity Measures

Another type of similarity measure is one which returns a numeric value for a case, rather than a partial order on cases — i.e. rather than a similarity measure being a function $\sigma :: \Theta \rightarrow \Theta \rightarrow \Theta \rightarrow \mathbf{bool}$ it will be a function that “scores” cases $\sigma_{\mathbf{N}_\infty} :: \Theta \rightarrow \Theta \rightarrow \mathbf{N}_\infty$, or, alternatively, $\sigma_{[0,1]} :: \Theta \rightarrow \Theta \rightarrow [0,1]$. This scoring approach is less general than the one developed in Sect. 2 because the orders defined by the scores are always total; it does not, for example, allow the possibility of incomparable cases. But [20] numeric measures may have the advantage of giving cardinal as well as ordinal information to the user.

The definition of cardinal similarity measures again begins by considering comparison of individual elements of a case to seed values, and then considers how to combine the atomic measures, including the use of weightings.

3.1 Atomic Similarity Measures

The particular difficulty in defining numeric measures is how to treat elements of a case that have non-numeric types, e.g. how to score a case whose spiciness is *mild*, when the seed specifies a desired value of *hot*. One approach is to order cases as in Sect. 2.1 and then convert from the order to a numeric score. This is discussed in Sect. 3.3, where it is shown that introducing the necessary cardinal information after ordering is problematic.

The alternative is to map non-numeric values to numeric ones. In what follows, functions that carry out this mapping will be denoted as f . Obviously, for numeric valued attributes of cases, f will typically be the identity function, or, if the similarity measures are to return normalised values — i.e. some value in $[0,1]$, rather than \mathbf{N}_∞ — a normalising function. In most work on numerical measures, equality of a value in a case to the seed is taken as a sign of similarity (corresponding to *is below*), or the difference between two values is taken as a sign of dissimilarity (corresponding to *best below*). But numeric correlates of *flat*, *minimal* and *maximal* can also be defined. In the definitions in Fig. 3, the higher the measure, the less similar the cases (some would call this a distance measure or dissimilarity measure). An additional family of similarity measures, $\mathbf{const}_n \ e \ x = n$, can be defined which will be useful in discussing weightings in Sect. 3.2. These measures will return a constant value for any case. Clearly, *flat* is a special case of \mathbf{const}_n , with $n = \infty$, and taking the inverse of a measure (subtracting, rather than adding the score given by that measure) is equivalent to multiplying by \mathbf{const}_{-1} .

flat	is
flat $e x = \infty$	$is\ e x = 0,$ if $f x = f e$ $= \infty,$ otherwise
maximal	minimal
maximal $e x = f e - f x,$ if $f x \leq f e$ $= \infty,$ otherwise	minimal $e x = f e - f x,$ if $f x \geq f e$ $= \infty,$ otherwise
best	id
best $e x = f x - f e $	id $e x = f x$

Fig. 3. Cardinal similarity measures

Other structures. It is also possible to derive cardinal similarity measures from the structures discussed in Sect. 2.2 — trees, DAGs, graphs, user defined structures. All that is required is that a distance function (such as \mapsto in Sect. 2.2) be defined for these structures which can then be used to give the “score” for each case, rather than using the distance to define an ordering, as was done in Sect. 2.2.

This could even be applied to numeric valued features by defining a distance function on numbers — e.g. a logarithmic distance function — and applying this directly. This provides an alternative to defining some function f and using operations such as best.

3.2 Combining Numeric Measures

Numeric similarity measures can be combined using basic arithmetic operations in a manner analogous to that presented in Sect. 2.3. Using the operators $+$, $-$ (unary and binary), and \times , measures can be added, subtracted, multiplied and, using the const_n measures and multiplication, weighted. Again, a normal form can be derived — a sum of products — and the products computed in parallel.

3.3 Switching Types of Similarity Measure

It is fairly easy to switch from cardinal to ordinal similarity measures. To transform a cardinal measure to an ordinal measure the cardinal information can be used to generate an ordinal measure by comparing values. If $\sigma_{\mathbf{N}_\infty}$ is a cardinal similarity measure an ordinal measure can be defined:

$$\sigma \vartheta x y = (\sigma_{\mathbf{N}_\infty} \vartheta x) \geq (\sigma_{\mathbf{N}_\infty} \vartheta y),$$

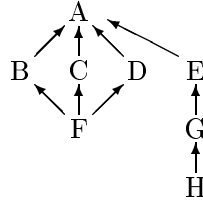
The scores determine the ordering. Obviously the cardinal information — how much more highly one case scores than another — is lost in the transformation.

The transformation from an ordinal measure to a cardinal measure is more complex. The problem is in creating ordinal information where none was previously available, and in transforming a partial order to a total order. One possibility is to take the number of cases that can be found between two cases as an ordinal measure of the similarity of those two cases. Note that, as before, the higher the measure, the less similar the cases. However, if the cases are incomparable, there will be *no* objects between the two cases, and the measure will have to be adapted to deal with this. A possible transformation is, therefore

$$\begin{aligned} \sigma_{\mathbf{N}_\infty} \vartheta x &= \infty, \text{ if } |\{y|x(\sigma \vartheta) y(\sigma \vartheta) \vartheta \vee \vartheta(\sigma \vartheta) y(\sigma \vartheta) x\}| = 0 \\ &= |\{y|x(\sigma \vartheta) y(\sigma \vartheta) \vartheta \vee \vartheta(\sigma \vartheta) y(\sigma \vartheta) x\}|, \text{ otherwise} \end{aligned}$$

Note that if ϑ and x are related, then the cardinality of the set of elements appearing between ϑ and x will never be zero, since both ϑ and x appear “between” ϑ and x .

This measure can, however, give possibly counter-intuitive results. Consider the ordering, for seed A :



The cardinal measure proposed will return a higher (worse) value for F than for H , because there are five values between A and F (including A and F themselves), and only four between A and H . An alternative would be to take the shortest path between the elements, given by:

$$\begin{aligned} \text{minpath } x \vartheta &= \infty, \text{ if } \neg(x(\sigma \vartheta) \vartheta) \wedge \neg(\vartheta(\sigma \vartheta) x) \\ &= 0, \text{ if } x = \vartheta \\ &= 1 + \min \{\text{minpath } x' \vartheta | x' \in \text{neighbours}\}, \text{ otherwise} \\ &\text{ where neighbours} = \{y|x \oplus y \wedge x \neq y \wedge \nexists z \notin \{x, y\} : x \oplus z \oplus y\} \\ &\quad \oplus = \sigma \vartheta, \text{ if } x(\sigma \vartheta) \vartheta \\ &\quad = (\sigma \vartheta)^{-1}, \text{ otherwise .} \end{aligned}$$

4 Conclusions

We have presented a repertoire of tools for constructing similarity measures, both numeric and symbolic. These tools make it possible to construct similarity measures systematically and/or incrementally, in which a more refined similarity measure is derived from the result of applying a simpler measure.

The implied loss of efficiency that the use of more flexible similarity measures entails can be compensated for by the opportunities this method offers for parallel evaluation.

A system has been developed that demonstrates the ideas presented [17, 18]. Currently this provides a graphical demonstration of the method. Work is in progress [7] at York to develop this into a realistic, efficient system, and to extend the work to cover other knowledge manipulation systems.

A An Example

A.1 The Case Base

Consider the problem of taking a guest out for a meal. The meals under consideration — the case base — are given in Fig. 4. The fields indicate the name of the dish, the set of ingredients, the type of meat, the degree of spiciness (one of *mild*, *medium mild*, *medium*, *medium hot*, *hot*, *extra hot*, *killer* and *suicide*), the number of calories, and the price. These fields will be accessed by the projection functions π_{name} , π_{ingr} , π_{meat} , π_{spic} , π_{cal} and π_{price} respectively. The two letter abbreviations to the left of the cases will be used to identify the cases.

lm	lamb casserole	{meat,tomato}	lamb	medium mild	700	£12
vb	vegetable biryani	{tomato,nuts,chilli}	none	medium	650	£ 8
cv	chicken vindaloo	{meat,nuts,chilli}	chicken	extra hot	700	£15
ps	pasta	{tomato,chilli}	none	extra hot	850	£ 8
tm	truite meuniere	{fish}	fish	mild	650	£16
bb	bœuf bouguignone	{meat,tomato,onion}	beef	medium mild	800	£13
pl	paella	{fish,meat,onion}	chicken	mild	850	£17
cc	couscous	{onion,chilli,tomato}	none	killer	800	£15
∅	seed	{meat,nuts,tomato}	turkey	hot	0	£15

Fig. 4. A simple case base and a seed

A.2 Atomic Similarity Measures

Assume your guest likes meat, nuts and tomato; their preferred meat is turkey; they like their food hot; and they are watching their weight. In addition, you require the meal to cost less than £15. A seed — also given in Fig. 4 — can be defined reflecting these requirements.

The budgetary restraint can be achieved by applying a filter: $\triangleleft (\pi_{\text{price}} <)$. The simplest of the guest’s requirements to model is their weight watching, as this is simply the inverse of the usual ordering on integers: $\pi_{\text{cal}} \text{id}^{-1}$. Their desire for hot dishes is also fairly simple, requiring an application of the **best** similarity measure, which will “break the back” of the spiciness ordering at **hot**. The required dishes will be selected by $\pi_{\text{spic}} \text{best}$.

The remaining two preferences are slightly more complex. For the desired list of ingredients a directed acyclic graph can be defined — the standard lattice representing the subset ordering on sets of ingredients — and the distance function for DAGs applied to order sets of ingredients according to their proximity to the ideal set of ingredients. The ordering generated — which will be called $\sqsubset_{\text{DAG}} \text{ingr}$ — reflects the fact that the distance function in this DAG is a measure of the number of elements common to the seed and the set under consideration. The meals best fitting the desired list of ingredients will be selected by $\pi_{\text{ingr}} (\sqsubset_{\text{DAG}} \text{ingr})$.

The final preference requires a tree to be defined representing a taxonomy of meat types, to which a distance function can be applied to select those meat types most similar to turkey. This tree is given in Fig. 5. The required meals will be selected by $\pi_{\text{meat}} (\sqsubset_{\text{Tree}} \text{meat})$. The four orders discussed here are presented in Fig. 6.

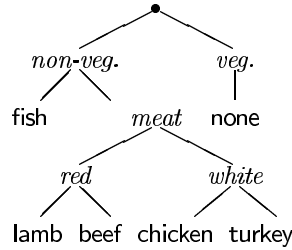


Fig. 5. A taxonomy of meat types

A.3 Retrieval

Assume that we first wish to filter out the more expensive meals, and then select according to our guest’s preferences. Assume also that our guest’s preferred list of ingredients and desire for meat similar to turkey is to be given priority over their weight watching and preference for hot food. The “best” meals will be selected by: $((\pi_{\text{ingr}} (\sqsubset_{\text{DAG}} \text{ingr}) \hat{\wedge} \pi_{\text{meat}} (\sqsubset_{\text{Tree}} \text{meat})) \gg (\pi_{\text{cal}} \text{id}^{-1} \hat{\wedge} \pi_{\text{spic}} \text{best})) \hat{\wedge} (\pi_{\text{price}} \triangleleft <)$, applied to ϑ . Application of the transformations presented in this paper shows this to be equivalent to the disjunctive normal form:

$$(\pi_{\text{price}} \sigma <)$$

lm,vb,cv,bb ↑ ps,pl,cc ↑ tm	cv,pl ↑ lm,bb ↑ tm ↑ ps,vb,cc	cv,ps vb ↑ ↑ cc lm,bb ↑ tm,pl	vb,tm ↑ lm,cv ↑ bb,cc ↑ pl,ps
$\pi_{\text{ingr}} (\sqsubset_{\text{DAG}} \text{ ingr})$	$\pi_{\text{meat}} (\sqsubset_{\text{Tree}} \text{ meat})$	$\pi_{\text{spic}} \text{ best}$	$\pi_{\text{cal}} \text{ id}^{-1}$

Fig. 6. Some atomic similarity measures

$$\begin{aligned}
 & \hat{\vee} (\pi_{\text{ingr}} (\sqsubset_{\text{DAG}} \text{ ingr}) \hat{\wedge} \pi_{\text{meat}} (\sqsubset_{\text{Tree}} \text{ meat}) \hat{\wedge} \pi_{\text{price}} \sigma_{<}) \\
 & \hat{\vee} (\pi_{\text{ingr}} (\sqsubset_{\text{DAG}} \text{ ingr}) \hat{\wedge} \pi_{\text{meat}} (\sqsubset_{\text{Tree}} \text{ meat}) \hat{\wedge} \pi_{\text{ingr}} (\sqsubset_{\text{DAG}} \text{ ingr})^{-1} \\
 & \quad \hat{\wedge} \pi_{\text{meat}} (\sqsubset_{\text{Tree}} \text{ meat})^{-1} \hat{\wedge} \pi_{\text{cal}} \text{ id}^{-1} \hat{\wedge} \pi_{\text{spic}} \text{ best} \hat{\wedge} \pi_{\text{price}} \sigma_{<}) .
 \end{aligned}$$

The three terms in this expression will select as maximal cases $\{\mathbf{lm}, \mathbf{vb}, \mathbf{ps}, \mathbf{bb}, \mathbf{pl}\}$, $\{\mathbf{lm}, \mathbf{bb}, \mathbf{cv}\}$ and $\{\mathbf{lm}, \mathbf{vb}, \mathbf{cv}, \mathbf{tm}, \mathbf{pl}, \mathbf{ps}, \mathbf{cc}\}$ respectively, the intersection of which is $\{\mathbf{lm}\}$. Consequently, the recommendation will be to serve the lamb casserole.

References

1. A. Aamodt and E. Plaza. Case based reasoning: Foundational issues, methodological variations and system approaches. *AI Communications*, 7(1):39–59, 1994.
2. R. Altermann. Adaptive planning. *Cognitive Science*, 12:393–421, 1988.
3. K.D. Ashley and E.L. Rissland. A case-based approach to modeling legal expertise. *IEEE Expert*, 3(3):70–77, 1988.
4. R. Bareiss, J.A. King, J. Ashley, K. Kolodner, B. Porter, and P. Thagard. Panel on “similarity metrics”. In *Proceedings of DARPA Case Based Reasoning Workshop*, pages 66–84. Morgan Kaufmann, 1989.
5. R. Barletta and W. Mark. Explanation-based indexing of cases. In *Proceedings of AAAI-88*, pages 541–546, 1988.
6. M. Brown. *A Memory Model for Case Retrieval by Activation Passing*. PhD thesis, Department of Computer Science, University of Manchester, 1994. Technical Report 94-2-1.
7. D. K. G. Campbell, H. R. Osborne, A. M. Wood, and D. G. Bridge. Generic operations for CBR in LINDA. Technical Report (to appear), Department of Computer Science, University of York, 1996.
8. A.D. Griffiths and D.G. Bridge. Formalising the knowledge content of case memory systems. In Ian D. Watson, editor, *Progress in Case-Based Reasoning: First United Kingdom Workshop in Case-Based Reasoning*, pages 32–41. Springer Verlag *Lecture Notes in Computer Science; 1020; Lecture Notes in Artificial Intelligence*, 1995.
9. T.R. Hinrichs. *Problem Solving in Open Worlds: A Case Study in Design*. Lawrence Erlbaum, 1992.

10. P. Klahr and G. Vrooman. Commercialising case based reasoning technology. In I.M. Graham and R.W. Milne, editors, *Research and Development in Expert Systems VIII*, pages 18–24. Cambridge University Press, 1991.
11. J.L. Kolodner. Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7:243–280, 1983.
12. J.L. Kolodner. Judging which is the “best” case for a case-based reasoner, 1989. published as part of [4], pages 77–81.
13. J.L. Kolodner. *Case Based Reasoning*. Morgan Kaufmann, 1993.
14. P. Koton. Reasoning about evidence in causal explanations. In *Proceedings of AAAI-88*, pages 256–261, 1988.
15. R. McCartney and K.E. Sanders. The case for cases: A call for purity in case-based reasoning. In *Proceedings of AAAI Symposium on CBR*, pages 12–16, 1990.
16. P. Myllymäki and H. Tirri. Massively parallel case-based reasoning with probabilistic similarity measures. In S. Wess, K.D. Althoff, and M.M. Richter, editors, *Proceedings of the 1st European Workshop on Case-Based Reasoning, Lecture Notes in Computer Science No. 837*, pages 144–154. Springer Verlag, 1994.
17. Hugh Osborne and Derek Bridge. Parallel retrieval from case bases. In Ian D. Watson, editor, *Proceedings of the 2nd UK CBR Workshop*, pages 43–54, 1996.
18. Hugh Osborne and Derek Bridge. A formal analysis of case base retrieval. Technical report, Department of Computer Science, University of York, 1997.
19. B.W. Porter. Similarity assessment: Computation vs. representation. In *Proceedings of DARPA Case Based Reasoning Workshop*. Morgan Kaufmann, 1989.
20. M.M. Richter. Classification and learning of similarity measures. In *Proceedings der Jahrestagung der Gesellschaft für Klassifikation, Studies in Classification, Data Analysis and Knowledge Organisation*. Springer Verlag, 1992.
21. M.M. Richter. The knowledge content of similarity measures. Invited talk at ICCBR, Sombria, Portugal, 1995.
22. M.M. Richter and S. Wess. Similarity, uncertainty and case-based reasoning in PATDEX. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honour of Noody Bledsoe*, pages 249–265. Kluwer, 1991.
23. M. Ryan. Prioritising preference relations. In S.J.G. Burn and M.D. Ryan, editors, *Theory and Formal Methods 1993: Proc. Imperial College Department of Computer Science Workshop on Theory and Formal Methods*. Springer Verlag, 1993.
24. B. Smyth and M. Keane. Adaptation-guided retrieval: Using adaptation knowledge to guide the retrieval of adaptable cases. In *Proceedings of the 2nd UK Workshop on CBR*, pages 2–15, 1996.