Decision Diagrams: Fast and Flexible Support for Case Retrieval and Recommendation*

Ross Nicholson, Derek Bridge, and Nic Wilson

University College Cork,
Cork, Ireland
`rn1@cs.ucc.ie, d.bridge@cs.ucc.ie, n.wilson@4c.ucc.ie`

Abstract. We show how case bases can be compiled into Decision Diagrams, which represent the cases with reduced redundancy. Numerous computations can be performed efficiently on the Decision Diagrams. The ones we illustrate are: counting characteristics of the case base; computing the distance between a user query and all cases in the case base; and retrieving the k best cases from the case base. Through empirical investigation on four case bases, we confirm that Decision Diagrams are more efficient than a conventional algorithm. Finally, we argue that Decision Diagrams are also flexible in that they support a wide range of computations, additional to the retrieval of the k nearest neighbours.

1 Introduction

Speed of response is important in Case-Based Reasoning (CBR). It is important in embedded case-based systems: in a rapidly changing environment, actions must be timely. It is important in interactive case-based systems: users will only wait so long for an answer. The challenges of providing timely responses to users of interactive case-based systems are increasing because the systems must now support a wider range of computations. In conversational recommender systems, for example, systems must increasingly support question selection [4] and the computation of explanations [6].

Compilation has a long history in Computer Science as a way of improving performance. But it is rarer in CBR. One example is Case Retrieval Nets, where we can think of the case base as having been compiled into a graph of information entities [3]. Another example is any system that automatically constructs an index structure into the case base (e.g. the k-d trees described in [8]).

In this paper, we compile the case base into a *Decision Diagram*. Decision Diagrams are described by Wilson [9], extending the approach of [1]. They, along with [1,2], represent a strand of research into compiling Constraint Satisfaction Problems (CSPs). Our contribution is to apply them to CBR, especially case-based retrieval and to current conversational recommender systems.

* The support of the Informatics Commercialisation initiative of Enterprise Ireland is gratefully acknowledged; this work was also supported by Science Foundation Ireland under Grant No. 00/PI.1/C075. We are grateful to Dr. Lorraine McGinty and Mr. James Reilly for making their Laptops and Cameras data available to us.

Location	**Bathrooms**	**Furnished**	**Bedrooms**
Chelsea	1	Yes	2
Chelsea	1	No	2
Chelsea	2	Yes	4
Chelsea	2	Yes	3
Clapham	2	Yes	2
Clapham	1	Yes	2
Clapham	1	No	2

Fig. 1. Example case base

In Section 2.1, we explain how to compile a case base into a Decision Diagram; in Section 2.2 we use the Decision Diagram to efficiently count properties of the case base; in Section 2.3 we show how to use the Diagram to efficiently compute the distance between a user's query and every case in the case base; and in Section 2.4 we explain how to efficiently retrieve the best k cases from the Diagram. Section 3 reports empirical results on four case bases, comparing operation counts and timings, and showing much improved performance when using Decision Diagrams. Finally, in Section 4, we argue that Decision Diagrams are also extremely flexible in that they support a wide range of useful computations.

2 Decision Diagrams

A Decision Diagram is a directed graph, having distinguished Source and Sink nodes. Each complete path, i.e. from Source to Sink, represents a solution to the Constraint Satisfaction Problem (CSP), or a case in the case base, from which the Decision Diagram was compiled. Values representing the degree to which constraints are satisfied, or the degree to which cases match a user's query, are propagated and aggregated across the graph to efficiently implement key CSP and CBR operations.

In our work, we assume that cases are 'flat' vectors of attribute-value pairs. The case base we use as a running example is shown in Figure 1. The example case base contains seven case descriptions of the kind used in case-based recommender systems. Here, each is a London property rental.

Figure 2 shows one possible Decision Diagram for the example case base. For m attributes, there are $m + 1$ layers (columns) of nodes, which we will refer to as layers 0 to m. In the example, four attributes gives five layers of nodes. There is only one node in layer 0, referred to as Source, and one node in layer m, referred to as Sink. In our specialisation of the Decision Diagram formalism [9], each of layers 0 to $m - 1$ is associated with a case attribute, and the edges that connect to the next layer are labelled with values for that attribute. In the example, Source is associated with the *Location* attribute and so edges that connect to the next layer are labelled with values *Chelsea* or *Clapham*; the next layer is associated with the *Bathrooms* attribute and so edges that connect to the next layer are labelled with values 1 or 2; and so on. Sink is not associated with any attribute. Each complete

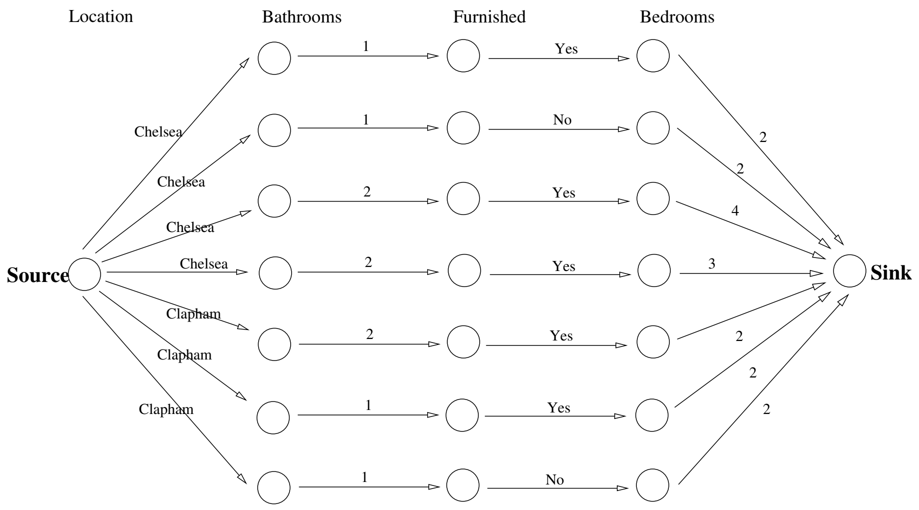

Fig. 2. DD(7): One possible Decision Diagram for Figure 1's case base

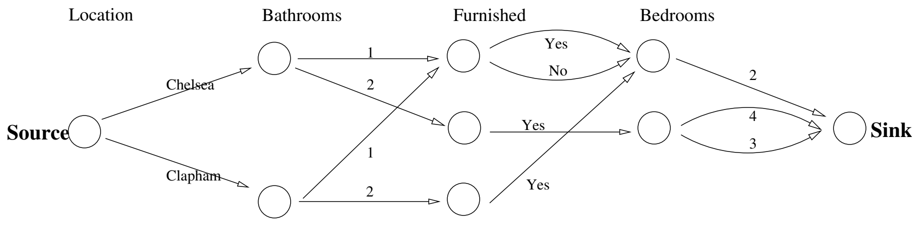

Fig. 3. DD(3): Another possible Decision Diagram for Figure 1's case base

path through the graph, from Source to Sink, corresponds to a case in the case base. In Figure 1 there are seven cases; in Figure 2 there are seven complete paths.

Now consider Figure 3, which is another Decision Diagram for the same case base. Nodes which were separate in Figure 2 have, in effect, been *merged* in Figure 3. The number of layers is the same and, crucially, there are still seven paths through the Decision Diagram, corresponding to the seven cases.

We define the width of a Decision Diagram at any point to be the number of edges connecting layer i to layer $i + 1$. For example, in Figure 3, the width between layers 0 and 1 is two. We define the width of the Decision Diagram *as a whole* to be the average of the widths between each layer, rounded to the nearest integer. The width of Figure 2 is therefore seven: it is this wide at all points. The width of Figure 3 is three: $(2+4+4+3)/4$. In this paper, we use the width as a concise way of referring to a Diagram: the Diagram in Figure 2, we call DD(7); and the one in Figure 3, we call DD(3). Obviously, DD(3) is more compact than DD(7): it is narrower at all points. It is this reduction in redundancy that will give us performance improvements over traditional case retrieval algorithms.

Algorithm 1. Building a Decision Diagram from case base CB, whose set of attributes is $A = \{a_0, \ldots, a_{m-1}\}$

```
insert a new node, Source, at layer 0
for all $c \in CB$ do
  $current :=$ Source
  $R := \{\}$
  $P := A$
  for $i := 0$ upto $m - 1$ do
    insert attribute-value pair $\langle a_i, v\rangle$ from case $c$ into set $R$
    remove attribute $a_i$ from set $P$
    if there is already an edge, labelled by value $v$, from $current$ to some node $next$ then
      $current := next$ {i.e. traverse the edge}
    else {i.e. need a new edge}
      $slice := \pi_P(\sigma_R(CB))$
      if there already exists a node $next$ at layer $i + 1$ for which slice$(next) = slice$ then
        insert an edge $current \xrightarrow{v} next$
      else {i.e. need a new node as well}
        insert a new node $next$ at layer $i + 1$
        slice$(next) := slice$
        insert an edge $current \xrightarrow{v} next$
      end if
    end if
  end for
end for
```

2.1 Compiling a Decision Diagram from a Case Base

Our algorithm for creating a Decision Diagram from a case base, Algorithm 1, is a novel specialisation of the one described in [9] for compiling a Decision Diagram from a CSP. We will explain this algorithm with reference to the first two cases in the case base shown in Figure 1.

Figure 4 depicts the Decision Diagram after we have processed the first case in the case base. The Decision Diagram initially contained only the Source node. Hence, edges were created for each attribute-value pair in the first case. Alongside the nodes, we show case base ‘slices’. These are explained in the next paragraph.

Now consider processing the second case in the case base. If possible, we follow an existing path through the Decision Diagram; we only insert new edges and nodes if necessary. We start at Source. The case’s first attribute-value pair is $\langle \textit{Location}, \textit{Chelsea}\rangle$ and there is an edge in Figure 4 labelled *Chelsea* emanating from Source; so we follow this edge. The next attribute-value pair in the case is $\langle \textit{Bathrooms}, 1\rangle$. There is an edge in Figure 4 labelled 1, and so we are able to follow this edge. We are now at the third node from the left in Figure 4.

The next attribute-value pair is $\langle \textit{Furnished}, \textit{No}\rangle$. There is no edge in Figure 4 labelled *No* emanating from our current node. One thing is certain: we will need to insert a new edge into the Diagram. The question is: will we also insert a new

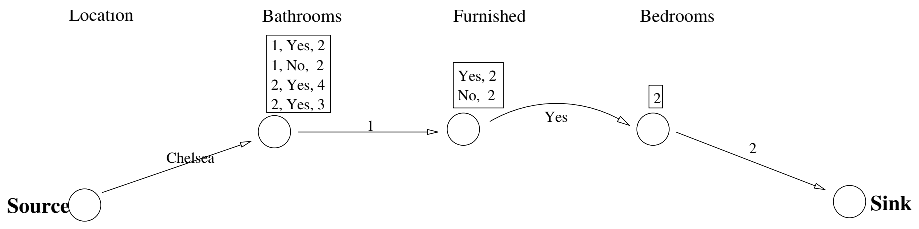

Fig. 4. Building the Decision Diagram: the result of processing the first case

node as the destination of this edge, or will we be able to use an existing node? This is where the case base slices are used.

In our specialisation of the Decision Diagrams in [9], slices are computed using σ and π, which are the standard select and project operators of relational algebra [5]. In the *select* operator, $\sigma_R(S)$, R is a Boolean condition. The operator returns all tuples (cases) in S that satisfy R: a kind of horizontal subset. In the *project* operator, $\pi_P(S)$, P is a set of attributes. The operator returns all tuples (cases) in S but confined to the attributes in P: a kind of vertical subset.

In our algorithm, R contains the attribute-value pairs that we have looked at so far from the case that we are processing. In the example, we have looked at $\{\langle \mathit{Location}, \mathit{Chelsea}\rangle, \langle \mathit{Bathrooms}, 1\rangle, \langle \mathit{Furnished}, \mathit{No}\rangle\}$. So, in forming our new slice, we select cases that agree with these attribute-value pairs. In our algorithm, P is the set of attributes that we have not looked at so far in the case that we are processing. In the example, there is just one attribute that we have not yet looked at: *Bedrooms*. So, in forming our new slice, we project the result of the selection operator on this attribute. Hence, in this example, we are computing:

$$\pi_{\{\mathit{Bedrooms}\}}(\sigma_{\{\langle \mathit{Location}, \mathit{Chelsea}\rangle, \langle \mathit{Bathrooms}, 1\rangle, \langle \mathit{Furnished}, \mathit{No}\rangle\}}(CB))$$

The resulting slice contains just one tuple, having one attribute, and the value of this attribute in that tuple is 2. Figure 5 shows that, because this new slice equals a slice already associated with a node in the destination layer, we do not need to insert a new node. Instead, we insert an edge to the node whose slice equals the new slice.

This concludes processing of the second case in the case base. The other cases are each processed in a similar fashion, giving rise to the Decision Diagram that we showed in Figure 3.

Of course, as can be seen in Figure 5, each slice can be computed incrementally from the previous one; it does not need to be computed 'from scratch'.

When we run our algorithm, we employ a heuristic that generally increases the opportunities for merging of nodes, thereby reducing the size of the Diagram. Specifically, we sort the *attributes* by ascending domain size. We have already done this in Figure 1: the *Bedrooms* attribute, whose domain size is three, comes after the other attributes, all of which have domain sizes of two. The order of the *cases* in the case base, however, is not significant: irrespective of their order, the algorithm produces the same Decision Diagram.

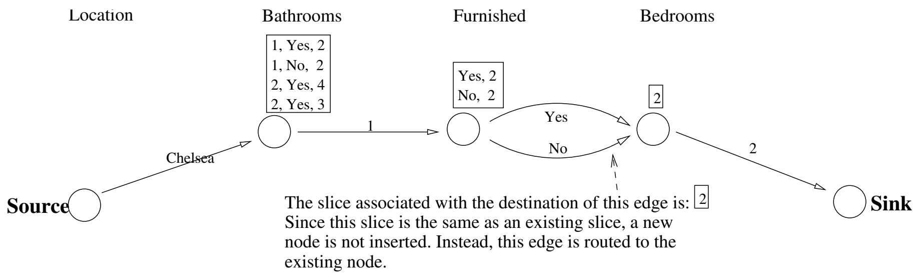
Location
Bathrooms
Furnished
Bedrooms
1, Yes, 2
1, No, 2
2, Yes, 4
2, Yes, 3
Yes, 2
No, 2
2
1
Yes
No
Chelsea
2
Source
Sink
The slice associated with the destination of this edge is: 2
Since this slice is the same as an existing slice, a new node is not inserted. Instead, this edge is routed to the existing node.

Fig. 5. Building the Decision Diagram: the result of processing the second case

A newly-retained case can easily be incorporated incrementally into the Decision Diagram. In essence, this involves running just the inner for-loop in Algorithm 1. However, computing and comparing slices means inserting a case into a Decision Diagram will be slower than inserting a case into a flat case base.

2.2 Counting Cases

We have shown how we compile a case base into a Decision Diagram, which is usually more compact than the original case base. What remains to be shown is how we can efficiently compute with the Decision Diagram. This involves the propagation and aggregation of values across the graph. We begin with something simple: counting properties of the case base from its Decision Diagram. We look at this first because doing so aids the exposition: it has the virtue of being readily understandable and it generalises to more useful operations.

Each node n in a Decision Diagram is associated with two values, which we refer to as its f-value and its g-value. We define $f(n)$ inductively as follows:

$$f(n) =_{\text{def}} \begin{cases} 1 & \text{if } n = \text{Source} \\ \sum\limits_{n' \longrightarrow n} f(n') & \text{otherwise} \end{cases} \tag{1}$$

where $n' \longrightarrow n$ signifies an edge from n' to n. In other words, the f-value of n is the sum of the f-values of its ‘parents’ over all edges entering n.

We define g-values analogously, this time summing g-values of ‘child’ nodes over all edges leaving n:

$$g(n) =_{\text{def}} \begin{cases} 1 & \text{if } n = \text{Sink} \\ \sum\limits_{n \longrightarrow n'} g(n') & \text{otherwise} \end{cases} \tag{2}$$

We can use a breadth-first ‘search’ from Source to Sink to compute f-values. A breadth-first ‘search’ from Sink to Source will compute g-values. The complexity of these procedures is linear in the size of the Decision Diagram.

Figure 6 shows f- and g-values computed in this way. For example, the node labelled by an asterisk has three edges entering it. We obtain its f-value by summing, for each edge entering the node, the f-values of the parent: $2+2+1 = 5$

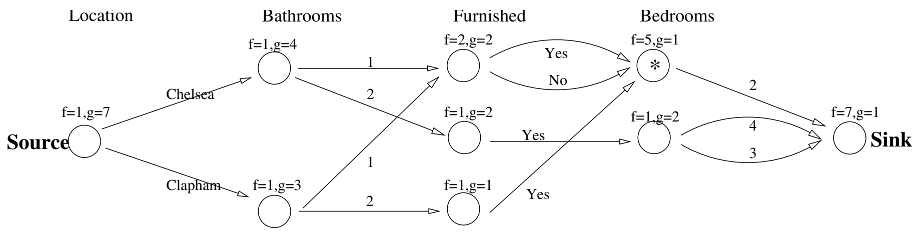

Fig. 6. Decision Diagram with f- and g-values for counting solutions. (The asterisk is referred to in the text.)

(by Equation (1)). We obtain its g-value by summing, for each edge leaving the node, the g-values of the child: 1 (by Equation (2)).

We see that $f(\text{Sink})$ and $g(\text{Source})$ both denote the number of paths (cases) in the graph, in this case seven.

The f-value at a node n summarises information about paths from Source to n; the g-value at n summarises information about paths from n to Sink. For example, the node labelled with an asterisk in Figure 6 is reached by five paths from Source and by one path to Sink.

Why would we want to compute *both* f- and g-values? Together, they allow us to compute how many complete paths pass through a *node* n. We do this by multiplying $f(n) \times g(n)$. For example, the number of complete paths passing through the node labelled with an asterisk is $5 \times 1 = 5$. In a similar vein, we can find out how many complete paths include a particular *edge* $n \longrightarrow n'$ by multiplying $f(n) \times g(n')$. For example, the number of complete paths that include the edge between the node with an asterisk and Sink is $5 \times 1 = 5$.

This can be used to compute how many complete paths (cases) contain a particular attribute-value pair. For example, we can determine in how many of the rental properties *Furnished* = *Yes*. We sum the number of complete paths that pass through each edge labelled *Yes*. There are three such edges, and the value we compute is $2 \times 1 + 1 \times 2 + 1 \times 1 = 5$. Assuming f- and g-values have already been computed, the complexity of this computation is proportional to the width of the Diagram at this point. While this may, in the worst-case, be equal to the number of cases, it will often, as in the example, be much lower.

In the next section, we will generalise these ideas to similarity-based retrieval.

2.3 Similarity-Based Retrieval

A user's query or probe q is a set of attribute-value pairs. We may wish to compute, for each case c in the case base, its degree of similarity to the query. In fact, our presentation will be in terms of distance, rather than similarity. We assume a set of local distance measures dist_a, one for each attribute $a \in A$. We take global distance dist to be a weighted sum of the local distances:

$$\text{dist}(q, c) =_{\text{def}} \sum_{a \in A} w_a \times \text{dist}_a(q.a, c.a) \tag{3}$$

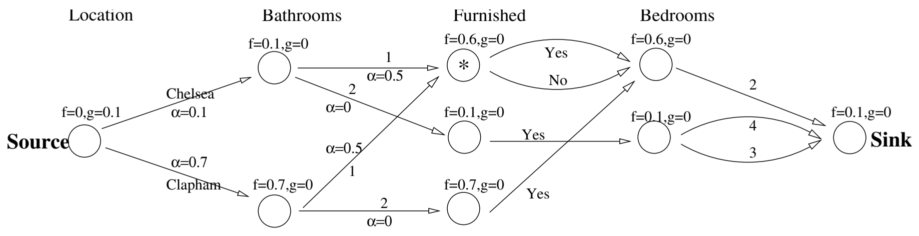

Fig. 7. Decision Diagram with f- and g-values for computing distances. The user query is *Bathrooms* $= 2$, *Location* $=$ *Pimlico*. Where α-values are not shown, they are 0. (The asterisk is referred to in the text.)

where w_a are the weights and $q.a$ and $c.a$ are q's and c's values for attribute a. We will constrain weights and local distances to be in $[0, 1]$, hence global distance will be in $[0, m]$, where m is the number of attributes, with 0 being the 'best' value and m being the 'worst'.

We use f- and g-values to propagate and aggregate the measures of distance. But, additionally, we associate with every edge $n \longrightarrow n'$ a value, which, following [9], we denote by $\alpha(n \longrightarrow n')$. It is important to appreciate that the α-value of an edge is distinct from the attribute-value of that edge. The attribute-values are the values that label the edges in Figures 2–5, e.g. that the *Location* is *Chelsea*. α-values, by contrast, denote local distances.

Initially, all α-values are set to 0. For each attribute-value pair $a = v$ in the user's query, we update the α-values of all edges that emanate from nodes that are in the layer of the Decision Diagram pertaining to attribute a. Specifically, if node n is in the layer that pertains to attribute a and there is an edge $n \longrightarrow n'$ labelled with attribute-value v', then

$$\alpha(n \longrightarrow n') := w_a \times \text{dist}_a(v, v') \tag{4}$$

Now, f- and g-values are defined inductively as follows:

$$f(n) =_{\text{def}} \begin{cases} 0 & \text{if } n = \text{Source} \\ \min\limits_{n' \longrightarrow n} (f(n') + \alpha(n' \longrightarrow n)) & \text{otherwise} \end{cases} \tag{5}$$

$$g(n) =_{\text{def}} \begin{cases} 0 & \text{if } n = \text{Sink} \\ \min\limits_{n \longrightarrow n'} (\alpha(n \longrightarrow n') + g(n')) & \text{otherwise} \end{cases} \tag{6}$$

As before, a breadth-first search from Source to Sink will compute f-values. But, in fact, we reduce effort by starting, not from Source, but from leftmost nodes entered by an edge with an altered α-value. We compute g-values in the opposite direction and starting from rightmost nodes exited by an edge with an altered α-value. The cost of computing these f- and g-values is, as before, linear in the number of edges in the Decision Diagram.

We show an example in Figure 7. In the example, the user prefers two bathrooms. We take $\text{dist}_{Bathrooms}(2, 1) = 0.5$ and $\text{dist}_{Bathrooms}(2, 2) = 0$. She would also like to live in Pimlico, and we will assume that $\text{dist}_{Location}(Pimlico, Chelsea) = 0.1$ and $\text{dist}_{Location}(Pimlico, Clapham) = 0.7$. We take all weights to be one. As shown in Figure 7, the local distances are used to update the α-values of all edges emanating from layers 0 and 1. The α-values of all other edges remain at 0, and are not shown in the figure in order to reduce clutter.

The node labelled by an asterisk has two edges entering it. We obtain its f-value by taking the minimum, over each edge entering the node, of the f-values of the parent added to the α-value of the edge: $\min(0.1 + 0.5, 0.7 + 0.5) = 0.6$ (by Equation (5)).

We see that $f(\text{Sink})$ and $g(\text{Source})$ both denote the distance between the best case in the case base and the user's query: the two properties in Chelsea with 2 bathrooms are 0.1 distant from the query.

We can use these f- and g-values in ways that are analogous to the uses we found in Section 2.2. We can compute the distance from the query of the best complete path that passes through a *node* n by adding $f(n)+g(n)$. For example, the best complete path passing through the node labelled with an asterisk has a distance of $0.6 + 0 = 0.6$ from the query. And, we can compute the distance from the query of complete paths that include a particular *edge* $n \longrightarrow n'$, again by adding $f(n) + \alpha(n \longrightarrow n') + g(n')$. For example, the best complete path that includes the edge labelled *Yes* that emanates from the node with an asterisk is $0.6+0+0 = 0.6$ distant from the query. And, we can compute the least distance from the query of complete paths that contain a particular attribute-value pair. For example, suppose we ask this question of furnished properties. For each edge $n \longrightarrow n'$ labelled *Yes*, we compute $f(n) + \alpha(n \longrightarrow n') + g(n')$, and we take the minimum of these values. There are three such edges, and the value we compute is $\min(0.6 + 0 + 0, 0.1 + 0 + 0, 0.7 + 0 + 0) = 0.1$.

2.4 *k* Nearest Neighbours

Of course, in CBR rarely would we just want to know how distant the best case is from the user's query. Rather, we wish to *retrieve* the best case, or the best k cases, or the cases whose distances are less than some threshold θ.

Algorithm 2 is a new algorithm that efficiently retrieves the best k cases from the Decision Diagram. An analogous algorithm can extract those cases whose distances are less than θ.

The algorithm maintains a priority-ordered queue of paths and repeatedly extends the best of these paths until it has found k paths that reach Sink or it has run out of paths to extend. It is like an A^* search inasmuch as the items on the queue are ordered by a combination of their path cost so far and an estimate of the cost of the cheapest path from the current node to Sink ($\text{cost}(p') + g(n')$ in the algorithm). However, there is a crucial difference. Typically in A^* the estimated part of the cost is truly an estimate, computed by some heuristic. In our algorithm, by contrast, it is not an estimate at all: it is the g-value, which is the *actual* cost of the cheapest path from the current node to Sink.

When a path p' is inserted onto the queue, it should be placed ahead of anything already on the queue of the same or lower priority. Then the algorithm will

Algorithm 2. Retrieving the k best cases from a Decision Diagram

```
$bestK := \{\}$
create $agenda$, an empty priority-ordered queue
create a path $p$ containing just Source
$\mathrm{cost}(p) := 0$
insert $p$ into $agenda$
while $|bestK| < k \wedge agenda$ is not empty do
  remove path $p$ from front of $agenda$
  $n :=$ the last node in path $p$
  if $n = $ Sink then
    insert $p$ into $bestK$
  else
    for all edges $n \longrightarrow n'$ do
      $p' := p$ extended by $n \longrightarrow n'$
      $\mathrm{cost}(p') := \mathrm{cost}(p) + \alpha(n \longrightarrow n')$
      insert $p'$ into $agenda$ where the priority of $p'$ is $\mathrm{cost}(p') + g(n')$
    end for
  end if
end while
return $bestK$
```

unerringly enumerate the best k cases from the Diagram in increasing order of cost. And, assuming f- and g-values have already been computed, its complexity will be approximately linear in k and m (the number of attributes): it will not depend on the size of the Diagram or the number of cases.

3 Efficiency Experiments

We compiled Decision Diagrams from four case bases: Travel, Laptops, Cameras and Lettings. Prior to compiling, we ordered the attributes by ascending domain size. Characteristics of the case bases and the corresponding Decision Diagrams are given in Table 1. We see from the table (row F) that between 83% and 94% of these case bases are repetitions of the same attribute-value pairs. In the Decision Diagrams, this redundancy is reduced to between 68% and 80% (row G). In terms of the representation of attribute-value pairs, the Decision Diagrams save between 23% and 53% of the representation of the case base (row H). However, because Decision Diagrams are more complex data structures, with greater space overheads, their memory footprint is larger than that of a case base in a text file. In two of the case bases, the footprint doubles in size; in one it is a little less than double; in another it is considerably more (row I). However, none of the footprints is unreasonable: all are less than 302kb (row E).

In fact, for each case base, we built *two* Decision Diagrams. In one we disallowed ‘merging’ of nodes, hence these Diagrams are like the one depicted in Figure 2. In the other, we allowed ‘merging’, as explained in Algorithm 1, hence these Diagrams are more like the one depicted in Figure 3. It is the latter Diagrams that are characterised in Table 1. (The characteristics of the former are

Table 1. Characteristics of case bases and corresponding Decision Diagrams

	Travel	Laptops	Cameras	Lettings	
Case Bases					
# of cases	1470	693	210	794	
# of attributes	8	14	9	6	
Domain sizes — smallest	4	2	5	2	
— largest	839	438	165	175	
— average	119	37	35	46	
# of attribute-value pairs	11760	9702	1890	4764	(A)
# of distinct attribute-value pairs	954	520	317	273	(B)
Size as text file (kb.)	132	105	15.8	43.1	(C)
Decision Diagrams					
# of edges	3771	2598	1006	1100	(D)
Width — at narrowest point	4	2	5	4	
— at widest point	1387	438	195	601	
— average	471	186	112	183	
Size as serialised Java object (kb.)	301.5	230.5	99.6	62.7	(E)
Build time (ms.) averaged over 10 runs	1156	718	156	266	
Comparison					
Redundancy in case base ($A - B$ as % of A)	92%	90%	83%	94%	(F)
Redundancy in DD ($D - B$ as % of D)	75%	80%	68%	75%	(G)
Size saving (D as % of A)	32%	28%	53%	23%	(H)
Size cost (E as % of C)	228%	220%	630%	145%	(I)

basically the same as the characteristics of the case bases themselves, e.g. they have uniform width which is, at every point, the same as the number of cases.)

Recall that we refer to Diagrams using their average width. So the two Travel Decision Diagrams (one without and one with 'merging') are referred to as DD(1470) and DD(471); the two Laptops Decision Diagrams are DD(693) and DD(186); the two Cameras Decision Diagrams are DD(210) and DD(112); and the two Lettings Decision Diagrams are DD(794) and DD(183).

To evaluate efficiency, we ran experiments to compare the number of operations and the times needed to compute twenty nearest neighbours. We used the leave-one-in methodology: each case in the case base is taken in turn and used to form queries. From a given case, we form all queries that comprise just one attribute-value pair taken from the case; then we form all queries that include two attribute-value pairs from the case; and so on until we form a query that involves using all the attribute-value pairs from the case. Hence, the number of queries is ${}^mC_1 + {}^mC_2 + \cdots + {}^mC_m$, where m is the number of attributes. In the histograms (Figures 8 and 9), we have columns for each query size and also columns (the rightmost) for all queries taken together, irrespective of size.

The queries are evaluated by three different systems. The first, denoted kNN, is an implementation of the nearest neighbours algorithm applied to the

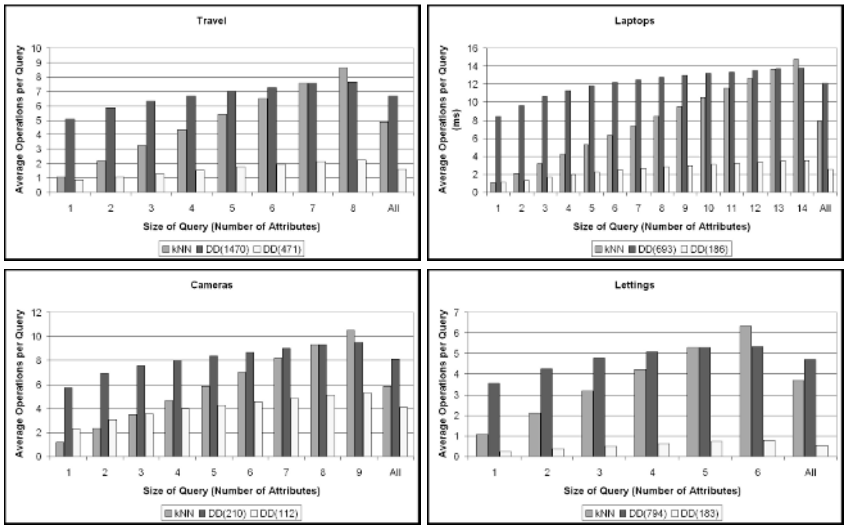

Fig. 8. Average operation counts by query size and for all queries

original case base: it does a linear search through all the cases of the case base, but it includes a degree of caching to give it a fairer chance against Decision Diagrams. The second system uses the Decision Diagram in which merging is disallowed, and the third system uses the Diagram in which merging is allowed.

Operation counts and timings both have well-recognised weaknesses as ways of comparing the efficiency of different algorithms. We hope by including both counts and times that the weaknesses of using one are compensated for by using the other. In any case, both sets of results tell much the same story.

In Figure 8 we show, for each system on each case base, the number of operations needed to compute the global distance between the query and every case in the case base. Hence, we are counting the number of local distances that the systems compute and the number of addition operations needed to aggregate the local distances into global distances. In these operation counts, we exclude any consideration of retrieving the best k cases. We did this because the different systems (kNN on the one hand and Decision Diagrams on the other hand) use such different ways of retrieving the winning cases that we could not find a fair basis for comparison in terms of operation counts. However, retrieval *is* included in the timing results reported in Figure 9 below.

We see in Figure 8, as we would expect, that the narrower Diagrams always perform fewer operations than the other two systems. Taken over all queries, irrespective of size, the narrower Decision Diagrams perform 67%, 68%, 30% and 85% fewer operations than kNN on Travel, Laptops, Cameras and Lettings respectively. Interestingly, Decision Diagram performance is much less variable over different query sizes than is kNN performance.

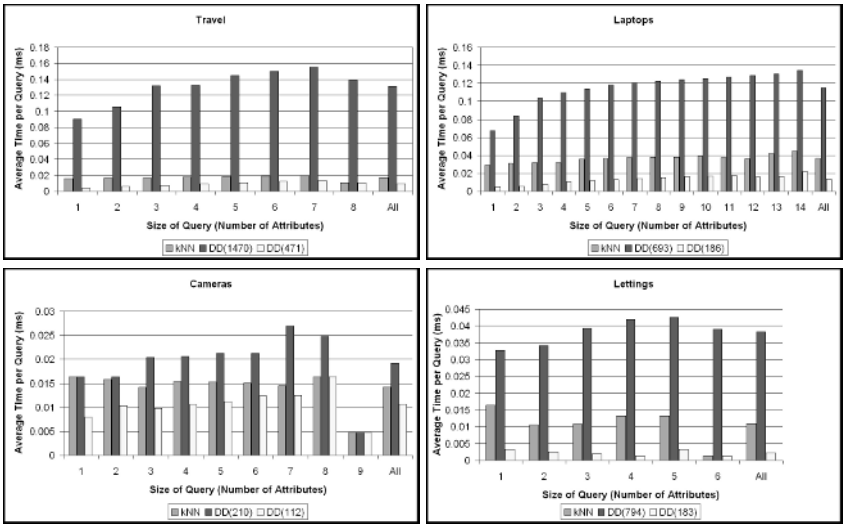

Fig. 9. Average time (ms.) by query size and for all queries

In Figure 9 we show, for each system on each case base, the time in milliseconds needed to retrieve the twenty nearest neighbours of the query. Hence, we are measuring how long it takes to compute distances *and* to retrieve the twenty winning cases. The figures used are averages over ten runs.

The Figure shows that the narrower Decision Diagrams are never outperformed, but there is a handful of query sizes where kNN is competitive on some of the case bases. Taken over all queries, irrespective of size, the narrower Decision Diagrams take 46%, 64%, 25% and 79% less time than kNN on Travel, Laptops, Cameras and Lettings respectively.

4 Concluding Discussion

We have shown two uses of the f- and g-values in Decision Diagrams: efficiently propagating and aggregating counts and distances. But the values can be generalised further for even greater flexibility; this is the way Decision Diagrams are presented in [9]. There it is shown that the f- and g-values can be drawn from any semiring. In [9], a semiring is defined as a set of values S containing different elements $\mathbf{0}$ and $\mathbf{1}$, and two operations on S, $\oplus$ and $\otimes$, that satisfy the following properties: $\oplus$ is associative and commutative with identity $\mathbf{0}$ (i.e. $a \oplus (b \oplus c) = (a \oplus b) \oplus c$, $a \oplus b = b \oplus a$ and $a \oplus \mathbf{0} = a$); $\otimes$ is associative and commutative with identity $\mathbf{1}$ (i.e. $a \otimes (b \otimes c) = (a \otimes b) \otimes c$, $a \otimes b = b \otimes a$ and $a \otimes \mathbf{1} = a$); $\mathbf{0}$ is a null element (i.e. $a \otimes \mathbf{0} = \mathbf{0}$); and $\otimes$ distributes over $\oplus$ (i.e. $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$). It is important to note that $\mathbf{0}$ and $\mathbf{1}$ are special elements of set S; they are not necessarily the integers 0 and 1.

Then, in general, f- and g-values are defined as follows

$$f(n) =_{\text{def}} \begin{cases} \mathbf{1} & \text{if } n = \text{Source} \\ \bigoplus\limits_{n' \longrightarrow n} (f(n') \otimes \alpha(n' \longrightarrow n)) & \text{otherwise} \end{cases} \tag{7}$$

$$g(n) =_{\text{def}} \begin{cases} \mathbf{1} & \text{if } n = \text{Sink} \\ \bigoplus\limits_{n \longrightarrow n'} (\alpha(n \longrightarrow n') \otimes g(n')) & \text{otherwise} \end{cases} \tag{8}$$

Initially, prior, e.g., to the imposition of user preferences, α-values are $\mathbf{1}$.

In Section 2.2, where we were counting properties of the case base, we were using the following: $S = \mathbb{N}$, the natural numbers (including zero); $\mathbf{0} = 0$; $\mathbf{1} = 1$; $\oplus = +$; and $\otimes = \times$. Equations (1) and (2) are instantiations of Equations (7) and (8) where $\otimes = \times$ and the α-value of every edge is 1.

One way of viewing what we were doing in Section 2.3, where we were using distance measures, is: $S = [0, m]$, the real numbers between 0 and m, where m is number of attributes; $\mathbf{0} = m$; $\mathbf{1} = 0$; $\oplus = \min$; and $\otimes = +$.[1] You can see that Equations (5) and (6) are special cases of Equations (7) and (8). What this also means is that Algorithm 2, our new algorithm for efficiently extracting best paths from a Decision Diagram, can be used for many semirings. The changes required are that path costs be initialised to the semiring's $\mathbf{1}$ element and, where addition occurs in Algorithm 2, it be replaced by the semiring's $\otimes$ operator. However, the semiring must additionally satisfy the *addition-is-max* property [9]: for all $a, b \in S$ either $a \oplus b = a$ or $a \oplus b = b$. This property ensures that there is a total order on S, which is necessary for ordering elements on the queue.

The great advantage of this generalisation is that the same framework (and, indeed, the same software) can be used in multiple ways, requiring only a change of semiring. For example, suppose we wished to do exact-matching, instead of using distance measures. In other words, we want to find cases that *exactly* match the attribute-values in the user's query. We need only switch to the following semiring: $S = \{\mathit{true}, \mathit{false}\}$; $\mathbf{0} = \mathit{false}$; $\mathbf{1} = \mathit{true}$; $\oplus = \vee$; and $\otimes = \wedge$.

In our software, we allow the nodes of the Decision Diagram to have multiple f- and g-values, based on possibly different semirings, so that we can simultaneously compute and store different consequences of the user's query.

In ongoing work, we are using Decision Diagrams to build recommender systems. We believe that the efficiency and flexibility of the Diagrams give an ideal foundation. For example, a number of recommender systems now use a two-stage approach, in which an exact-matching stage precedes an inexact-matching stage (e.g. [7]); as we have explained, both forms of matching are supported by Decision Diagrams. Entropy-based approaches to question selection (e.g. [4]) rely on

[1] In fact, strictly, in order to satisfy the requirement that the $\mathbf{0}$ element (in this case, m) be a null element (i.e. $a \otimes \mathbf{0} = \mathbf{0}$), we cannot let $\otimes = +$. Instead, we let $\otimes = \overset{m}{+}$, where $\overset{z}{+}$ is simply addition in which the result is not allowed to exceed z: $x \overset{z}{+} y =_{\text{def}} \min(x + y, z)$. $\overset{m}{+}$ does have m as its null element. However, this nicety plays no part in our software because, the way we are computing with local distances, they will never sum to more than m.

dynamically counting cases, in the light of incrementally-supplied user preferences; as we have shown, counting is efficient in Decision Diagrams; incremental processing is also easy. Explanations of retrieval failure are also important, especially in systems that use exact matching (e.g. [6]). Although not described here, we have generalised the ideas in [1] to allow us to efficiently support computations related to retrieval failure.

References

1. Amilhastre, J., Fargier, H. & Marquis, P.: Consistency restoration and explanations in dynamic CSPs — Application to configuration, *Artificial Intelligence*, vol.135(1–2), pp.199–234, 2002
2. Bryant, R.E.: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, *ACM Computing Surveys*, vol.24, no.3, pp.292–318, 1992
3. Burkhard, H.-D. & Lenz, M.: Case Retrieval Nets: Basic ideas and extensions, in H.-D.Burkhard & M.Lenz (eds.), *Procs. of the 4th German Workshop on CBR*, pp.103–110, Humboldt University, 1996.
4. Doyle, M. & Cunningham, P.: A Dynamic Approach to Reducing Dialog in On-Line Decision Guides, in E.Blanzieri & L.Portinale (eds.), *Procs. of the 5th European Workshop on Case-Based Reasoning*, pp.49–60, Springer, 2000
5. Elmasri, R. & Navathe, S.B.: *Fundamentals of Database Systems (5th edn.)*, Addison-Wesley, 2006
6. McSherry, D.: Explanation of Retrieval Mismatches in Recommender System Dialogues, in D.W.Aha (ed.), *Procs. of the Workshop on Mixed-Initiative Case-Based Reasoning*, pp.191–199, Norwegian University of Science and Technology, 2003
7. Ricci, F., Arslan, B., Mirzadeh, N. & Venturini, A.: ITR: A Case-Based Travel Advisory System, in S.Craw & A.Preece (eds.), *Procs. of the 6th European Conference on Case-Based Reasoning*, pp.613–641, Springer, 2002
8. Wess, S., Althoff, K.-D. & Derwand, G.: Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning, in S.Wess et al., (eds.), *Procs. of the 1st European Workshop on Case-Based Reasoning*, pp.167–181, Springer, 1993
9. Wilson, N.: Decision Diagrams for the Computation of Semiring Valuations, *Procs. of the 19th IJCAI*, pp.331-336, 2005