The Lumberjack Algorithm for Learning Linked Decision Forests

William T. B. Uther and Manuela M. Veloso[*]

Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, U.S.A.
{uther,veloso}@cs.cmu.edu

Abstract. While the decision tree is an effective representation that has been used in many domains, a tree can often encode a concept inefficiently. This happens when the tree has to represent a subconcept multiple times in different parts of the tree. In this paper we introduce a new representation based on trees, the *linked decision forest*, that does not need to repeat internal structure. We also introduce a supervised learning algorithm, Lumberjack, that uses the new representation. We then show empirically that Lumberjack improves generalization accuracy on hierarchically decomposable concepts.

1 Introduction

Trees have been used for the representation of induced concepts in numerous areas of AI, including supervised learning with decision trees (Breiman *et al.* 1984; Quinlan 1992) and reinforcement learning (RL) with tree based representations (Chapman and Kaelbling 1991; McCallum 1995; Uther and Veloso 1998). Trees are a powerful representation. However, to represent some concepts they may need to represent some subconcepts multiple times. For example, to represent the boolean concept $AB \lor CD$ a decision tree has to repeat the representation of either AB or CD (see Fig. 1a where CD is repeated).

a) A tree representing the boolean
concept $AB \lor CD$

b) A concept with repeated
internal structure
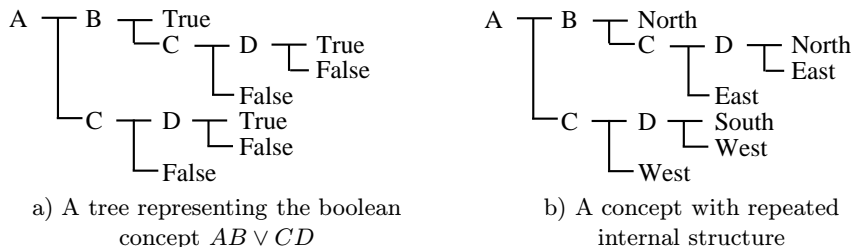
Fig. 1. Trees with repeated structure

This repetition of entire subtrees is well known and has been studied by several researchers (see Section 2). In addition, we have found many RL domains in

---

[*] Prof. Veloso is currently visiting faculty at the MIT AI Laboratory, Cambridge, MA.

which the tree repeats *internal* structure. For example, consider a concept mapping boolean inputs, $\{A, B, C, D\}$, to action outputs, {North,South,East,West} as shown in Fig. 1b. The internal structure of the CD subtree is repeated even though the leaves are not. It chooses between either North and East, or South and West depending upon the value of A.

In most inductive systems work must be performed to learn each part of the tree. If a subconcept is represented twice then it must be learnt twice. Moreover, each individual representation of a sub-concept will be learnt using only part of the available data. For example, in Fig. 1a the representation of CD when A is true is learned separately from the representation of CD when A is false. In an RL domain, these repeated structures can be viewed as subtasks or macros.

In this paper we present a new representation, the *linked decision forest*,[1] which, while similar to a tree, does not have to repeatedly represent, and so repeatedly relearn, sub-concepts. We also introduce an algorithm, Lumberjack, for growing these linked forests. Through this representation we can give Lumberjack a bias towards learning abstract decompositions of the target concept. We show empirically that Lumberjack generalizes more effectively than a simple tree based approach on hierarchically decomposable concepts.

2   Related Work

Duplicated subtrees, as in Fig. 1a, are a well known problem. Two decision tree-like systems that attempt to factor out repeated substructure are Pagallo and Haussler's (1990) FRINGE system and Oliver and Wallace's (1992) decision graph induction system. Kohavi's (1995) read once oblivious decision graphs are also related, though less closely as they use a significantly different method to generate the graph.

The FRINGE system works by first growing a normal decision tree. Once this tree is fully grown, the last two decisions above each leaf in the tree (the fringe of the tree) are processed to form new attributes. The original tree is discarded and a new tree is grown using both the original attributes and the new attributes. The whole process is repeated, with the number of attributes constantly growing, until accuracy on a separate dataset starts dropping. The fact that attributes are not removed if they turn out not to be useful is an efficiency concern, as is the repeated re-growing of the tree.

In Oliver and Wallace's (1992) system, decision graphs are inferred directly using the Minimum Message Length Principle (MML) (Wallace and Boulton 1968; Quinlan and Rivest 1989; Wallace and Patrick 1993). The system proceeds much as would a decision tree learner, except for two changes. Firstly, instead of a depth first approach to recursively splitting the dataset, the splits are introduced

---

[1] The term 'decision forest' has been used previously in the machine learning literature to refer to a collection of different decision trees, each separately representing the same concept (Murphy and Pazzani 1994). We introduce the term 'linked decision forest' to refer to a collection of decision trees with references between the trees so the forest as a whole, not just the individual trees, represents a concept.

in a best first manner. The location and decision in the next decision node to be introduced are chosen using MML. Secondly, instead of introducing a new decision node, the system can join two leaves together.

Kohavi's (1995) HOODG system is very closely related to Ordered Binary Decision Diagrams (Bryant 1992). These have a number of differences from arbitrary decision graphs. They both require an ordering among the variables and will only generate a graph that tests the variables in that order. As discussed by Kohavi, this limits the representation so that it is less efficient than an arbitrary decision graph. However, it allows a canonical representation to be found that is often compact. Most importantly as far as the authors are concerned, the algorithm is not incremental and so cannot be transferred to RL using the techniques of (Chapman and Kaelbling 1991; McCallum 1995; Uther and Veloso 1998) (see Hoey *et al.* (1999) for the application of OBDDs to RL).

Both Oliver and Wallace (1992) and Kohavi (1995) use a decision graph representation. A decision graph is not capable of factoring out structure which is only repeated internally, like the repeated CD structure in Fig. 1b. Additionally, Oliver and Wallace's (1992) decision graph algorithm chooses when to factor out repeated structure (join two leaves) using MML. The algorithm is choosing subtrees to 'join' based on comparison of their outputs, without any comparison of the structures required to represent the correct subconcepts (which haven't been grown at the time the decision to join is made).

In addition to the related work on decision graphs, our work is based on Nevill-Manning's (1996) work on the automatic decomposition of strings. Given a linear sequence of symbols with no prior structure, his SEQUITUR algorithm forms a simple grammar where repeated substrings are factored out. For example, given the string $S \rightarrow abcdababcd$, SEQUITUR produces the grammar: $A \rightarrow ab, B \rightarrow Acd, S \rightarrow BAB$. This grammar re-represents the original string in a compact form.

3   The Linked Forest Representation

For linear strings the grammar is a well known representation for a hierarchical decomposition. We introduce the linked forest representation which allows hierarchical decomposition of trees. A linked forest is composed of trees with references between them in the same way a grammar is composed of rewrite rules with references between them. One tree in the linked forest is marked as the root tree. The root node of this tree is the starting point for classification by the forest. Figure 2 shows an example of a boolean linked decision forest.

The inter-tree references take two forms. When a node makes a *value reference* to another tree the semantics are similar to a jump instruction; processing simply continues in the new tree. When a node makes an *attribute reference* to another tree the semantics are similar to a function call. The referencing node has children which are in one-to-one correspondence with the leaves of the referenced
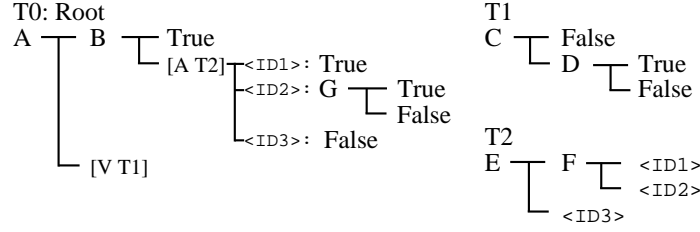
Fig. 2. A linked decision forest showing the root tree T0, and the trees T1 and T2; T0 includes a value reference to T1, [V T1], and an attribute reference to T2, [A T2]

tree. Control is passed across to the referenced tree until a leaf is reached, then passed back to the corresponding child of the referencing node.

If a tree is only referenced by attribute references, an *attribute tree*, then it does not require class labels or other data in its leaves. It simply has ID values that allow the corresponding children to be found. Lumberjack does not yet form value trees. They are mentioned for comparison purposes. One can view FRINGE as forming attribute trees, like Lumberjack, and the Decision Graph induction algorithm as forming value trees.

4 The Lumberjack Algorithm

Nevill-Manning's (1996) SEQUITUR algorithm detects common subsequences in strings by tracking digrams. For our algorithm we define a structure similar to a digram for trees, a *di-node*, that can be hashed for fast duplicate detection.

A di-node is defined as a pair of internal nodes in the forest such that one node is a child of the other. Two di-nodes are defined to be equal if the parent nodes are equal, the child nodes are equal, and the child is in the same location in both di-nodes (i.e. child ordering is important). For example, in Fig. 1a the two nodes labelled A and B form a di-node. The two nodes labelled A and B in Fig. 2 also form a di-node. These di-nodes are equal; the difference in nearby nodes is irrelevant. There are also two di-nodes made up of nodes labelled C and D in Fig. 1a. Those di-nodes are equal, but they are not equal to the di-node made up of nodes labelled C and D in Fig. 2; the D node is not in the same location relative to its parent.

Note that either, or both, of the nodes in a di-node could be a reference to another tree, and so a di-node can represent an arbitrarily large set of nodes. Also note that two di-nodes will be equal if and only if they represent equivalent sets of nodes that have been decomposed in the same way.

We are now in a position to give an overview of the Lumberjack algorithm. Table 1 shows the algorithm in detail. Initially the forest starts as a single tree with a single leaf node. Leaves are then split and a new decision nodes added, one at a time. As the forest is updated a hash table records all di-nodes currently in the forest. We use the Minimum Description Length (MDL) principle to choose the next decision node and to decide when to stop growing the forest (the details

of the MDL selection are discussed later). Once an internal node has been added the forest is checked for duplicate di-nodes using the di-node hash. Any non-overlapping duplicates are extracted to form a new attribute tree and the original di-nodes are replaced with references to the new tree. Any trivial attribute trees (trees referenced only once or having less than two internal nodes) are removed and their structure reinserted into the referencing tree(s).

This extraction and reinsertion of di-nodes removes all duplicated substructure from the forest. Because duplicate di-nodes are detected in all trees, it is common for the structure to be more than two levels deep.

Note that we only ever form attribute trees from internal nodes. In SEQUITUR it is possible to form a rewrite rule containing the last character of a string because the end of the string is unique. If we merge leaves in Lumberjack then we lose the ability to differentiate the positions where we might wish to add further nodes. While this might sometimes be useful for linked decision forests, as shown by Oliver and Wallace (1992), it is difficult to find a suitable criterion for doing this while retaining the ability to form attribute references.

4.1   Altering the Inductive Bias

In the previous text we didn't supply all the details of the algorithm. If the decision criteria for new nodes are chosen from only the original attributes, and only leaves of the root tree are extended, then the concept learned will be the same as that learned by a normal tree induction algorithm; there will be no change in inductive bias. The representation will have all repeated structure separated into other trees, but this is only a change in representation, not concept. We can change the inductive bias of the algorithm, and hence the concept learned, by extending the ways the forest is grown.

The first change is to allow the induction algorithm to split not only on the original attributes, but also to introduce an attribute reference to any tree in the forest. This can be viewed as a form of macro replay. The one restriction is that the use of this tree not introduce a cycle in the forest. Lumberjack records which trees reference which other trees in a directed acyclic graph (DAG). No split that would introduce a cycle in this graph is allowed.

The second change is to allow the algorithm to grow the forest not only at leaves of the root tree, but at the leaves of any tree in the forest. This can be viewed as a form of macro refinement. Again there is a restriction. Recall that leaves of attribute trees correspond with the children of the nodes that reference them. If you split a leaf of an attribute tree, then you must split the corresponding children of the referencing node(s). If any of the corresponding children is not a leaf then we do not allow the split.

Growing attribute trees changes the number of outcomes of decision nodes elsewhere in the forest. That in turn changes the number of outcomes of other decision nodes, etc. Because the trees form a DAG, it is possible to update the trees in reverse topological order and know that all trees being referenced by the tree currently being updated are themselves up to date.

- Begin with a single leaf, empty di-node hash and empty tree DAG
- Record this forest as the best forest so far
- Repeat until no further splits are possible
  - Set best description length this iteration to ∞
  - For each leaf in the root tree or other trees:
    - Check if splitting this leaf would mean splitting a non-leaf elsewhere
    - If so, continue with next leaf
    - For each possible split criterion
      - If this split causes a cycle in the tree DAG then continue with next split
      - Introduce new decision node with this split
      - Update forest structure (see part b)
      - Calculate description length
      - If length is less than the best length this iteration, remember this split
      - Remove new decision node from forest
      - Update forest structure (see part b)
  - Reintroduce node with best split
  - Update forest structure (see part b)
  - Add new di-node to hash
  - While there are duplicate di-nodes, single use trees or degenerate trees
    - Use non-overlapping duplicate di-nodes to form a new attribute tree and replace original di-nodes
    - Reinsert any trees used only once
    - Reinsert any degenerate trees (less than two internal nodes)
    - (all while maintaining the di-node hash and tree DAG)
  - If forest has a shorter code length than current best forest, remember it
- Return best forest

a) The main linked forest learning algorithm

- For each tree in reverse topological order of the tree DAG
  - For each node in a post-order traversal of the tree
    - If this node is not a reference to an attribute tree then continue to next node
    - Delete each child which corresponds to a leaf no longer in the referenced tree
    - Insert a new child (leaf) for each new leaf in the referenced tree

b) The subroutine to update forest structure

Table 1. The Lumberjack algorithm

Node Type	Bits
Leaf	$\log_2(\frac{b}{b-1})$
decision node	$\log_2(b) + \log_2(N_A + N_{PT})$

Table 2. Costs to encode a non-root node

Finally, the correspondence between the leaves of an attribute tree and the children of a node that references that tree is important for the hashing of di-nodes. The hash table should use that correspondence rather than child numbering for generating hash codes and testing equality. By avoiding the use of child numbering the algorithm does not have to re-hash di-nodes when an attribute tree grows or shrinks.

4.2   MDL Coding of Linked Decision Forests

The Minimum Description Length (Rissanen 1983), or Minimum Message Length (Wallace and Boulton 1968), Principle is a way of finding an inductive bias. It uses Bayes' Rule, $P(T|D) \propto P(T)P(D|T)$, and Shannon's information theory, the optimal code length of a symbol that has probability p is $-\log_2(p)$, to choose between competing models for data. The model and data are both encoded according to a coding scheme. The model which has the shortest total code length is chosen.

The Lumberjack algorithm could also be used with other decision node selection criteria. MDL was chosen for ease of implementation and because it supplies a stopping criterion.

Our coding scheme for MDL comparisons is a minor change from the Wallace and Patrick (1993) scheme for decision trees. Let N_T be the number of trees, N_A the number of attributes, N_{PT} the number of trees after the current tree in the topological ordering and b be the branching factor of our parent node. First, the number of trees in the forest is encoded using $L^*(N_T)$ bits.[2] Then the trees are encoded in reverse topological order. Each tree is encoded by performing a pre-order traversal of the tree and encoding each node using the number of bits shown in Table 2.

The one cost not yet specified is the cost to encode the root nodes of the trees. These have no parent node; b is undefined. When there is only one tree, a leaf at the root is encoded using $\log_2(N_A)$ bits and a decision node is encoded using $\log_2(\frac{N_A}{N_A-1}) + \log_2(N_A)$ bits, as in Wallace and Patrick (1993). When there is more than one tree, we know that none of the root nodes are leaves. The root decision nodes can be encoded using only $\log_2(N_A + N_{PT})$ bits. Finally, the examples are encoded using the costs in Table 3.

---

[2] $L^*(X) = \log_2^*(X) + \log_2(c)$, where $c \simeq 2.865064$, is a code length for an arbitrary integer. $\log_2^*(X) = \log_2(X) + \log_2(\log_2(X)) + \ldots$ summing only positive terms (Rissanen 1983).

Table 3. Costs of MDL example coding

5 Experiments

We tested Lumberjack using standard supervised learning experiments. We compared the generalization accuracy of a decision tree learner and Lumberjack, each using the same MDL coding. The results are shown in Figs. 3 and 4. The graphs show the averages over 10 trials. We tested for significance using a paired Wilcoxon rank-sum test ($p = 0.05$).
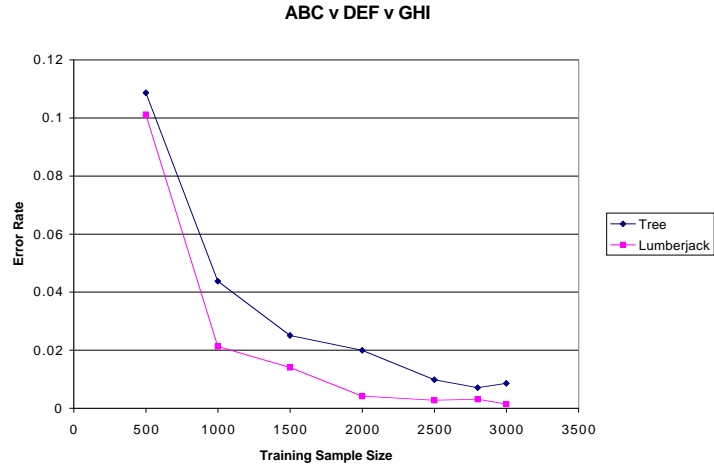
Fig. 3. Experimental results learning the concept $ABC \vee DEF \vee GHI$

The first set of results are for the boolean function $ABC \vee DEF \vee GHI$. Training samples were sampled with replacement from the concept, then the output was flipped in 10% of the samples. The testing dataset was a complete dataset without noise. Results are shown in Fig. 3. The difference in error rate

between the tree and forest algorithms is significant for sample sizes 1000 through 2500 inclusive, and also for the 3000 sample dataset.[3]

The second set of results uses a dataset generated by mapping a reinforcement learning problem back into a supervised learning problem. In this domain a two legged robot learns to walk about a simple 10×10 maze. The Robot cannot slide its feet along the ground, nor can it hover with two feet in the air - it requires a sequence of movements to walk. The robot knows its X, Y location and the ΔX, ΔY and ΔHeight differences between its legs. There are eight actions; the robot can raise or lower either foot or it can move the raised foot, if any, in any of the compass directions. This problem was fed into a traditional Markov Decision Problem algorithm, and the resulting policy was used as a dataset for our supervised learning experiment. The domain is discrete - the Δ's each have only 3 possible values, and the X, Y location was encoded using a series of variables $\{X < 1, X < 2, \ldots, X < 9, Y < 1, Y < 2, \ldots, Y < 9\}$. This coding is similar to the one implicitly used by C4.5 for continuous variables. Training datasets were generated by randomly sampling, with replacement, from this true dataset. The testing dataset was the full true dataset.

Again the results, in Fig. 4, are the averages over 10 trials. There is a significant difference between the two algorithms for sample sizes of 2000 or more. While the results at sample size 4000 are still significant, it is clear that both algorithms are converging again as sample size increases; the tree has enough data to grow the repeated structure. Looking at the forest for large sample sizes it is possible to see the separation of structure representing the maze from structure representing the ability to walk.

6   Conclusion

We have introduced a new tree-based representation, the linked decision forest, and a learning algorithm, Lumberjack that can use the linked forest representation. This representation doesn't need to repeat substructure leading to more efficient use of data.

The Lumberjack algorithm combines SEQUITUR style decomposition with decision tree learning. Unlike decision graph generation algorithms, Lumberjack only decomposes internal nodes in the forest. This improves the representational power of Lumberjack over decision graph algorithms, but this comes at a price: Lumberjack currently does not join leaves even when this would be useful. We have empirically shown improved generalization accuracy over a simple tree.

---

[3] We also compared with C4.5. C4.5 is always significantly better than the MDL tree learning system. This is a well known deficiency of MDL vs. C4.5 and is orthogonal to the use of Lumberjack style decomposition. With 1000 or more datapoints, C4.5 and Lumberjack perform similarly.
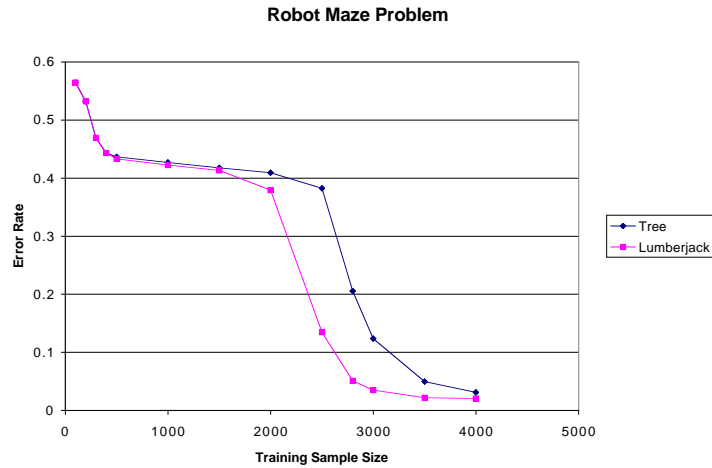
Fig. 4. Experimental results learning a policy to walk through a maze

References

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Wadsworth and Brooks/Cole Advanced Books and Software, Monterey, CA, 1984.

Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 726–731, Sydney, Australia, 1991.

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI-99)*, Stockholm, Sweden, 1999. Morgan Kaufmann.

Ron Kohavi. *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. Ph. d. thesis, Department of Computer Science, Stanford University, 1995.

Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1995.

Patrick M. Murphy and Michael J. Pazzani. Exploring the decision forest: An empirical invesitgation of occam's razor in decision tree induction. *Journal of Artificial Intelligence Research*, 1:257–275, 1994.

Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structures in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

Craig G. Nevill-Manning. *Inferring Sequential Structure*. Ph. d. thesis, Computer Science, University of Waikato, Hamilton, New Zealand, 1996.

J. Oliver and C. S. Wallace. Inferring decision graphs. Technical Report 91/170, Department of Computer Science, Monash University, November 1992.

Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80(3):227–248, 1989.

J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1992.

Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

William T. B. Uther and Manuela M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 769–774, Madison, WI, 1998.

William T. B. Uther and Manuela M. Veloso. The lumberjack algorithm for learning linked decision forests. In *Symposium on Abstraction, Reformulation and Approximation (SARA-2000)*, volume 1864 of *Lecture Notes in Artificial Intelligence.* Springer Verlag, 2000.

C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Journal*, 11(2):185–194, 1968.

C. S. Wallace and J. D. Patrick. Coding decision trees. *Machine Learning*, 11:7–22, 1993.