2006

Binary Decision Diagrams and Crisp Possibilistic Reconstructability Analysis

Martin Zwick
Portland State University, zwick@pdx.edu

Alan Mishchenko
University of California - Berkeley

Citation Details
Zwick, Martin & Mishchenko, Alan (2006). International Conference on Complex Systems (NECSI), Boston, June 25-30.

Binary Decision Diagrams and Crisp Possibilistic Reconstructability Analysis

Martin Zwick
Systems Science Ph.D. Program
Portland State University
Portland OR 97207-0751
zwick@pdx.edu

Alan Mishchenko
Department of EECS
University of California
Berkeley CA
alanmi@eecs.berkeley.edu

Abstract - *The paper discusses the application of Binary Decision Diagrams (BDDs) in the reconstructability analysis of crisp possibilistic systems. In particular, we show how BDDs can be used to represent set-theoretic relations and implement the three basic operations of reconstructability analysis.*

Keywords: reconstructability analysis, crisp possibilistic systems, binary decision diagrams, decomposition of relations and functions

1 Introduction

Reconstructability analysis [1,2,3] is well developed for two types of data: set-theoretic relations (crisp possibilistic systems) and frequency or probability distributions (probabilistic systems). This paper explores the use of binary decision diagrams (BDD) in the reconstructability analysis (RA) of crisp possibilistic systems. By focusing on BDD, we treat only systems here whose variables are binary, but the results of this paper can be extended to multi-valued variables using multi-valued decision diagrams (MDD). MDDs are a generalization of BDDs for variables taking values from finite domain. In most practical applications, MDDs are encoded using binary variables and manipulated using BDDs. This is why discussion in this paper is relevant for the case of multi-valued variables as well.

The BDD representation of completely specified Boolean functions [4] is of interest for two reasons: (1) it can be used to represent Boolean relations compactly, and (2) it leads to a faster computation of operations on relations that are performed in RA.

The first of these is a certain gain: for N binary variables, the size of a relation – the number of tuples needed to define it – goes up as 2^N, but the size of the BDD representation for the relations appearing in practical problems often has a lesser complexity. The compression achieved by using BDDs is the greater, the more structure is present in the data set. For random or pseudo-random collections of data, however, the BDD size is still exponential. If RA operations for problems with inherent structure can be performed directly on BDD representations, then this representation could become standard for crisp possibilistic RA because of its space-saving advantage. If, in addition, RA operations are faster when done on BDD representations than on standard tuple representations, then BDD representation would offer not only a space but also a time enhancement of RA computation. Such speed increases are often encountered in BDD implementations because BDD store information in an *implicit* form contrasted with the *explicit* representation of relations in terms of tuples.

In the implicit representation, a single object (such as a tuple of a relation) corresponds to a path in the BDD graph. It is known that the number of paths in the graph can be exponential in the number of nodes. Hence, the best-case exponential compression of the data sets represented as BDDs. This is also the reason why single operations on the graph nodes in this implicit representation can accomplish multiple effects simultaneously when looked at from the point of view of manipulating the original tuple objects (encoded as paths in the graph). This reflects the polynomial dependence of BDD representation size for some practical problems, as compared to the exponential dependence on the number of variables of the alternative explicit representations. For example, the BDD representation of the Exclusive-OR function of *n* variables is linear in the number of variables, while the tuple representation is exponential in the number of variables.

In RA of crisp possibilistic systems [5,6], a set-theoretic relation is decomposed into a set of lower ordinality relations, which together with the maximum uncertainty principle yield a calculated relation that is either the same as the original relation (lossless decomposition) or different from it (lossy decomposition). An RA model (decomposition) is assessed in three steps [7,8]:

(1) *projection* of the data into the relations constituting the model,
(2) *composition* of these relations using the maximum uncertainty principle, and
(3) *evaluation* of the calculated relation by comparing it to the original data relation.

Zwick, Martin & Mishchenko, Alan (2006). International Conference on Complex Systems (NECSI), Boston, June 25-30.

Consider, for example, a set-theoretic relation ABC, and a model AB:BC. To assess this model, one generates the AB and BC projections of ABC. Next one composes these projections into a calculated $ABC_{AB:BC}$ relation, given by $ABC_{AB:BC}$ = (AB ⊗ C) ∩ (BC ⊗ A). The Cartesian products which are intersected might be called the "expanded" AB and BC, relations, respectively; these are the maximum uncertainty ABC relations consistent with AB and BC, respectively. Finally, one checks if $ABC_{AB:BC}$ = ABC and if not what the error in the model is.

These three operations can be done directly on BDD representations of relations, which may, for many datasets, lead to faster processing than what is achievable using tuple representations. Also, the space saving feature of BDDs can offer a significant advantage in RA modeling.

These three operations define the steps needed in the *confirmatory* mode of RA, that is, these are the steps needed to test one specific RA model. This is the focus of the present paper. However, BDD might be used also in the *exploratory* mode, where one considers a wide range of possible models. Also BDD or MDD might be useful not only in crisp-possibilistic RA, but in probabilistic RA as well. These possible uses of RA are considered in the Discussion section later.

2 Representation

Consider the ABC relation, defined in Table 1 for three binary variables. The table shows also two of its three projections, AB and BC. The third projection, AC, has no constraint, i.e., consists of all four possible pairs.

Table 1. An ABC relation and its AB and BC projections

A	0	0	0	1	1
B	0	1	1	1	1
C	0	0	1	0	1

0	0	1
0	1	1

0	1	1
0	0	1

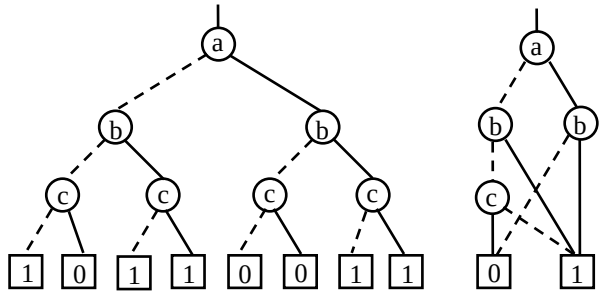
If it were necessary to define the relation by specifying for each of the tuples (minterms) whether it is in the relation or not in the relation, one would have the graph (decision tree) shown on the left in Figure 1. There are 2^N final 0 or 1 nodes in the tree, which specify for all tuples whether they are in the relation or not.

The BDD representation of this relation, which is much more compact than the decision tree, is shown on the right. A dashed line (called a "low edge") corresponds to a variable value of 0 and a solid line (called a "high edge") corresponds to a value of 1. At the bottom of the graph, the squares containing 0 and 1 indicate whether a tuple is absent or present, respectively, in the relation. The BDD graph of ABC shows that if A=0, then if B is 1, then the tuple is present in the relation, but if B is 0, then if C is 0, the tuple is present, but if C is 1, the tuple is absent. For A=1, if B=1, the tuple is present, but if B=0, the tuple is absent.

In the explicit representation of the decision tree on the left, there is a terminal node for every state (tuple) in the relation and also for every state not in the relation, hence the 2^N dependence of this kind of representation. The relation given in Table 1 thus requires 7 nodes in the decision tree (not counting the 0- and 1-squares that specify the presence or absence of the tuple.) Even if we only specified the tuples present in the relation, or, alternatively, those absent in it, we would still be left with of order 2^{N-1} tuples. By contrast, the BDD representation in this case requires only 4 nodes. In general, the BDD size can be anything from a constant to exponential in the number of variables, but the empirical observation is that, for many typical data sets arising in the practical applications, the BDD size is manageable.

The reason why the BDD representation is often more compact is because it stores the information about the relation in the *paths*, rather than in the terminal nodes, of a graph. Because the number of paths goes up exponentially with the number of nodes, this makes it possible for the number of nodes in the representation of data for some problems to go up linearly with the number of variables.

Figure 1. Decision tree (left) and BDD (right) for the ABC relation of Table 1. Dashed and solid lines stand for variable values of 0 and 1, respectively.



The BDD is obtained from the tree on the left by applying two reduction rules [4]:

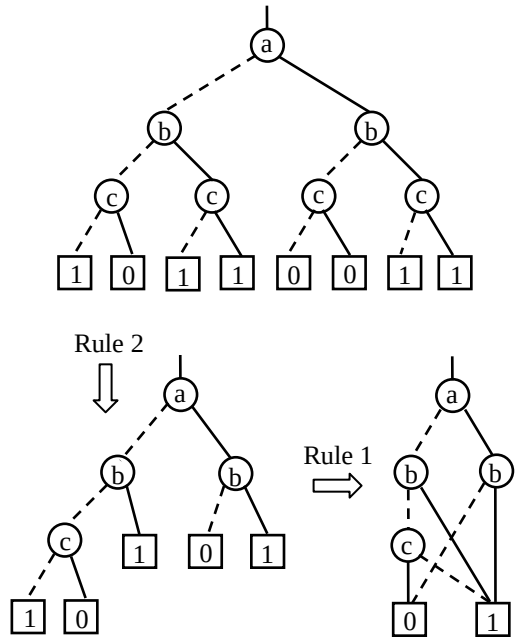
Rule (1): If several nodes are labeled by the same variable and have identical successors, only one of them is allowed to remain in the graph, and

Rule (2): If both edges of a node have the same successor, the node is removed from the graph.

By rule (2), the 01c node, which has both edges pointing to the 1-box, is dropped and replaced with the 1-box; similarly the 10c and 11c nodes are replaced with the 0-box and 1-box, respectively. Then, by rule (1), all 0-boxes are fused into one, and similarly for all 1-boxes. This process of

reduction of the decision tree resulting in the BDD is shown in Figure 2.

Figure 2. Reduction of decision tree resulting in BDD.



At this point, it is convenient to define how the *complement* of a set-theoretic relation is taken. This is done using

Rule (3): The complement of a BDD is obtained by swapping the 0 and 1 terminal nodes.

3 Projection, Composition, Evaluation

The BDD formalism implements projection by using existential abstraction (or quantification) with respect to a set of variables. Essentially, a tuple belongs to the resulting relation (the projected or quantified relation) iff there exists an assignment of variables projected away, which when added to it, make it a tuple of the original relation.

The computation of the projection with respect to a set of variables on the BDD representation can be performed in two steps that are combined in the practical implementation.

Step 1. First one computes the cofactor relations for each set of values of the projected variables. For example, if we are projecting away variable C to get the AB projection, we compute cofactor relations for (C=0) and (C=1). Each of these cofactors can be computed by traversing the BDD and removing nodes that depend on C. In doing so, all the incoming edges of the nodes depending on C are redirected to the nodes pointed by the corresponding edge of C. For example, if we are computing the cofactor relation for C=1, we redirect any edge going to a C-node to the node pointed by the high edge of the C-node. Subsequently, the resulting BDD is reduced using Rules 1 and 2. These steps and reductions are implemented in the BDD package [9], a library of software procedures for the manipulation of Boolean functions in the BDD form.

Step 2. Next, one computes the union of the relations, which is the same as the Boolean OR (the sum) of the Boolean functions represented by the corresponding BDDs. The Boolean operations can be computed on the BDDs using the operator If-Then-Else (ITE). For three functions, F, G, and H, the operation is: ITE(F, G, H) = F&G+ F′&H, where & is logical AND, + is logical OR, and ′ is the complement. This operation is efficiently implemented in the BDD package [9]. The OR operation is reduced to the ITE operation as follows: F+G = ITE(F,1,G).

The composition operation is the Boolean AND (the product) of the Boolean functions represented by the operand BDDs. Similarly to the Boolean OR, the AND operation is reduced to ITE: F&G = ITE(F,G,0).

The variables missing in the operands are added to the BDD by simply assuming that an original BDD depends on the additional variables. No actual change to the BDD nodes has to be performed. In the software implementation, the BDD package has to expand the set of its support variables, but the BDDs in the package remain unchanged.

Composition yields a calculated relation which must be evaluated relative the observed relation. Either the calculated relation loses constraint with respect to the data, or it is lossless, i.e., equivalent to the original relation. In the above example of composition, there is no loss of constraint, and the equivalence of the BDD representations of the original and composed relation can be ascertained by comparing the pointers to the two BDD representations. Because the BDD is a canonical representation, the two BDD pointers are equal if and only if the two relations are equivalent.

We are interested also in the constraint loss (error), which occurs when the reconstructed relation is not equivalent to the data. The error is the set of tuples in the calculated relation, $ABC_{AB:BC}$, that are not found in the data, i.e., the intersect of $ABC_{AB:BC}$ and ABC ′, the complement of ABC :

$$E = ABC' \cap ABC_{AB:BC}$$

This intersect operation can be implemented using the BDD AND operation, as described above for composition. It may in this case be more efficient to perform projection and then to apply a specialized BDD traversal to check the existence of error without computing ABC ′& $ABC_{AB:BC}$. To find out what tuple(s) constitute the error, however, it would still be necessary to compute ABC ′& $ABC_{AB:BC}$.

4 Discussion

From the perspective of the current state of research in BDDs and Boolean satisfiability [10] (determining whether the given Boolean formula has a satisfying assignment), the problem of finding error in the model can be also solved using Boolean satisfiability without BDDs. Most likely, neither of these two approaches will dominate another in general, but will show certain advantages on different types of problems.

The three operations of projection, composition, and evaluation involved in assessing a model in *confirmatory* RA can be done with BDD representations. In *exploratory* RA, one searches through the Lattice of Structure (LoS), or some sub-lattice, to find the best (simplest and least lossy) model that fits the data (the observed relation). For some purposes, one might be interested not in a *single* best model, but in a set of good models; and for other purposes – although only for low ordinality relations -- one might be interested in the evaluation of *all* models in the LoS (or some sub-lattice of it).

BDDs might contribute to the exploratory mode in a modest way by assisting in the generation of structures to be assessed, i.e., in the generation of all or parts of the LoS. If BDDs can examine multiple models in parallel, it would offer much more substantial additional benefits to RA modeling. The number of structures in the LoS is exponential or hyper-exponential in the number of variables. If the implicit representation of BDD would allow it to "simultaneously" assess multiple models, its contribution to RA would be formidable.

So far, only the application of BDD (or MDD) to RA for crisp possibilistic RA has been explored, but one might consider also the possible use of MDD in probabilistic RA. In probabilistic RA, a frequency or probability is attached to every tuple. If this frequency/probability is "binned" into discrete values, the distribution becomes a mapping which might be analyzed by crisp possibilistic RA. This possibility and other approaches to probabilistic RA are under investigation.

The following tutorial papers provide additional details on the theory and implementation of BDDs: [11,12].

References

[1] G. Klir, *The Architecture of Systems Problem Solving*. Plenum Press, New York, 1985.

[2] K. Krippendorff, *Information Theory: Structural Models for Qualitative Data (Quantitative Applications in the Social Sciences #62)*, Sage, Beverly Hills, 1986.

[3] *International Journal of General Systems* (IJGS) Special Issue on GSPS, Vol 24, pp. 1-2, 1996.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Transactions on Computers, Vol C-35, No. 8, pp. 677-691, 1986.

[5] R. C. Conant, "Set-Theoretic Structure Modeling," *Int. J. General Systems*, Vol 7, pp. 93-107, 1981.

[6] M. Zwick and H. Shu, "Set-Theoretic Reconstructability of Elementary Cellular Automata," *Advances in Systems Science and Applications*, Special Issue, Vol 1, pp. 31-36, 1996.

[7] M. Zwick, "Wholes and Parts in General Systems Methodology," In: *The Character Concept in Evolutionary Biology*, edited by Gunter Wagner. Academic Press, New York, pp. 237-256, 2001. http://www.sysc.pdx.edu/res_struct.html

[8] M. Zwick, "An Overview of Reconstructability Analysis," *Kybernetes*, Vol 33, No. 5/6, pp. 877-905, 2004. http://www.sysc.pdx.edu/res_struct.html

[9] F. Somenzi, BDD Package CUDD, 2004. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html

[10] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, Vol 48, No. 5, pp. 506-521, 1999.

[11] H. R. Andersen, "An Introduction to Binary Decision Diagrams," 1997. http://www.itu.dk/people/hra/notes-index.html

[12] F. Somenzi, "Binary Decision Diagrams," 1999. http://citeseer.nj.nec.com/somenzi99binary.html