Incremental Compilation of Object-Oriented Bayesian Networks

Christina Merten

merten@cs.ubc.ca
Department of Computer Science, University of British Columbia,
2366 Main Mall, Vancouver, BC, V6T1Z4, Canada

Abstract. Object-oriented paradigms have been applied to Bayesian networks to provide a modular structure which allows greater flexibility and robustness. These object-oriented Bayesian networks may be used over larger and more complex domains. However, as the networks get larger, the computational cost of triangulation and junction tree construction grows. The process of creating new junction trees when changes are made to object-oriented Bayesian networks is particular inefficient. Bangso et al [1] describe a method for using the networks maximal prime subgraph decomposition to only reconstruct the parts of the junction trees effected by these changes. This project is an implementation of that algorithm.

1 Introduction

Object-oriented paradigms have long been applied in the context of programming languages to make code more coherent and aid its reuse. Object-oriented programming languages [3] provide a modular structure that increases the robustness and flexibility of programs. Thus, they are particularly well-suited to the design of large-scale programs. Changes made in one class naturally propagate to all objects of that class. Similarly, this approach may be applied to Bayesian networks to make them more efficient and flexible, allowing their use over more complex domains. Object-oriented networks have also proven to be particularly well-suited to highly dynamic domains [3].

As these object-oriented Bayesian networks grow in size, the process of constructing junction trees becomes more computationally expensive. This process must be repeated whenever changes are made to the network. In a large network, much of this work is redundant, especially when the changes made are small. In this case, time is often wasted triangulating the unchanged part of the network.

An alternative approach, described by Bangso et al [1], involves using the maximal prime subgraph decomposition to quickly identify and isolate the modified part of the larger network. A junction tree may then be created for the smaller, changed fragment to replace the appropriate piece in the existing junction tree. This eliminates the unnecessary step of reprocessing the unchanged part of the network, building the section of the new junction tree that is identical to the old part. This project is an implementation of the more efficient algorithm.

2 Object-Oriented Bayesian Networks

In an object-oriented Bayesian Network (OOBN), the basic unit is an object [1]. It may be viewed as a set of set of properties that are associated with some entity in our domain. An object has identity, state and behavior. It also belongs to a *class*, which is a description of a set of objects with the same structure behavior and characteristics.
Specifically, an *object X* consists of three sets of nodes [4] :

- *I(X)* : These are input nodes that correspond to nodes which are not in the class but which may be used as parents of nodes inside instances of the class. These nodes may not have parents inside of the class.
- *E(X)*: These are encapsulated nodes that can only have parents and children inside of the class.
- *O(X)*: These are the output nodes that can have children outside of the class.

An object may also be denoted by its *full variable* $V^{+}(X) = <E_1, .., E_m, O_1, \ldots, O_n>$ or by its *output variable* $V(X) = <O_1,\ldots,O_n>$, where the $E_1,..,E_m$ are the object's encapsulated nodes and the $O_1,\ldots O_n$ are the object's output nodes. An object's *interf*ace [1] consists of its input nodes and output nodes. Objects in an OOBN are d-separated from each other by their interface nodes [4].

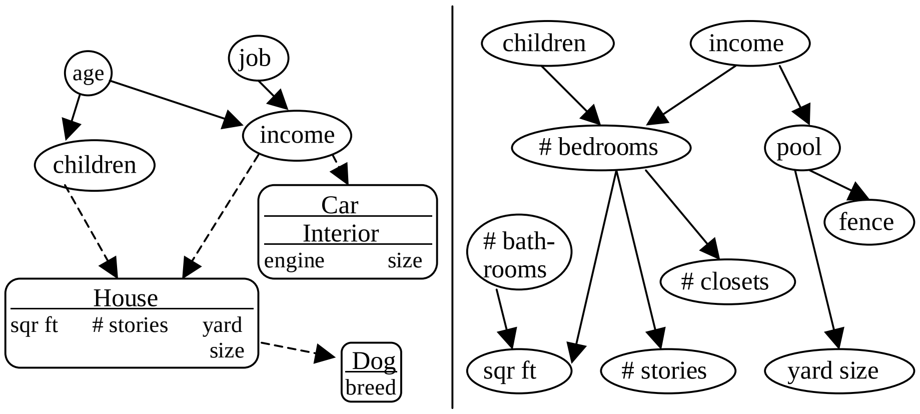

Figure 1. An OOBN depicting attributes of a person appears on the left. On the right a close-up image of the object *House* is shown.

Figure 1 shows an example of an object-oriented Bayesian network depicting attributes of a person. On the left, an overview of the network as a whole is shown. The objects appearing in this network are *House*, *Car, Interior* and *Dog*. Object *Interior* is defined inside of object *Car* in this example. On the right, a close-up image of the object *House* appears. In object *House*, the input nodes are *children* and *income* and the output nodes are *sqr ft*, *# stories* and *yard size*. The encapsulated nodes are *# bedrooms, # bathrooms, # closets, pool and fence.*

As mentioned earlier, while input nodes are part of the specification of an object, they do not actually appear in the class to which the object belongs. Instead, they correspond to nodes outside of the class. In an OOBN, the use of links from nodes outside of an object to input nodes inside is analogous to "passing parameters" by value to the object. These links are known as *reference links* [1] and they appear as dotted lines in figure 1.

One of the trademarks of the object-oriented approach is the ability to define classes that inherit the properties of other classes. Class A is a *subclass* [4] of class B if, for input sets I_A and I_B of classes A and B and output sets O_A and O_B of A and B, $I_B \subseteq I_A$ and $O_B \subseteq O_A$. So class A *inherits* the properties of class B and may have additional attributes of its own. Any changes made to B are passed along to A, increasing the overall flexibility and robustness of the network. This also leads to a natural hierarchy of object generalization and refinement.

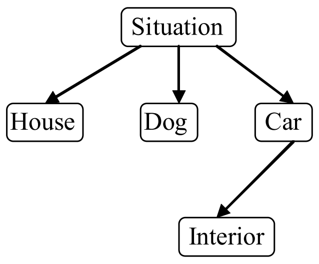

Figure 2. Instance tree for network depicting attributes of a person

In figure 1, the object *Interior* is defined inside of the object *Car*, just as *Car* is defined inside of the larger OOBN. In fact, each object of an OOBN is defined inside of a unique *encapsulating class* [1]. Here the OOBN as a whole may be referred to as the *situation* object that contains all of the outermost objects in the network. This leads to another hierarchy of object encapsulation. An *instance tree* [4] may be used to represent all of these relationships. Here each object appears as the child of its encapsulating class. An instance tree for the OOBN in figure 1 appears in figure 2. Instance trees will be shown to be useful in the triangulation of the overall OOBN.

3 Initial Compilation of an OOBN

The first step of this algorithm involves creating an initial junction tree that may be updated as changes are made to the original network. The process of triangulating a Bayesian network and constructing a junction tree is known as *compilation* [2]. It begins with moralization of the OOBN.

Suppose there exist two or more nodes in an OOBN that are parents of the same node. If the child node is a protected or output node, then all of its parents must belong to the same object. If the child node is an input node, then all of its parents must belong to the unique encapsulating object. In any case, parents sharing a child must belong to the same object and so moralization doesn't involve the addition of any link that crosses from one object to another. Thus each object (including the outermost situation object) may be moralized independently of the others to maintain the structure of the overall OOBN [1].

Since objects in the OOBN are d-separated from each other by their interfaces, each object in the OOBN may be triangulated separately, resulting in a junction tree. In this case, fill-in edges may sometimes be created that cross from one object to another so some communication between objects is needed. Here instance trees play an important role [1].

First the leaves of the instance tree i.e., those objects in which no other object is defined, are triangulated. The only constraint here is that, in each of these objects, none of the interface nodes are eliminated. In each of these triangulations, as in any, a set of fill-in edges results. Fill-ins that appear between two interface nodes in an objects could effect the triangulation of the encapsulating object so they must be propagated upwards in the instance tree [6, 7]. All other fill-ins are considered protected and thus not propagated.

This process is repeated recursively, traveling up the instance tree [6, 7]. In each object a triangulation is performed in which the interface nodes shared with encapsulating object are not eliminated. Note that interface nodes that are shared with the object's children in the instance tree are eliminated here. Thus, through this process, every node in the OOBN will eventually be eliminated. This results in a valid triangulation in which no fill-in edge exists between two variables in different objects.

4 Incremental Compilation of an OOBN

Traditionally, when changes are made to an OOBN, the entire network is then recompiled to form a new junction tree. This process, particularly the triangulation step, is very computationally expensive. It is also largely inefficient, especially when the changes made to the network are small. In this case, most of the time is spent recreating the part of the junction tree that corresponds to the unchanged piece of the network. Through the method of in-cremental compilation, this work can be avoided. Here only he part of the network that has been changed - or whose junction tree may be effected by those changes - will be recompiled. The smaller junction tree for this network fragment will then be "merged' with the original junction tree over the entire network. The maximal prime sub-graph decomposition is the key to identifying and isolating those parts of the network that need to be recompiled.

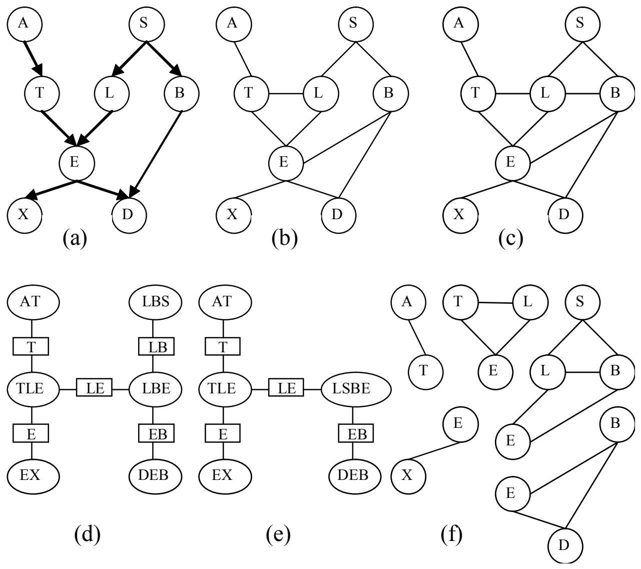

Figure 3. (a) The Asia network. (b) moral graph. (c) triangulated graph. (d) junction tree. (e) Maximal Prime Subgraph Decomposition tree. (f) Maximal Prime Subgraph Decomposition

The maximal prime sub-graph decomposition

The Maximal Prime Sub-graph Decomposition (MPSD) follows easily from the junction tree of a Bayesian network [2]. Given a network G, moralization must first be performed. Then the junction tree that corresponds to the optimal triangulation may be constructed. The maximal prime sub-graphs (MPSs) of G may then be formed by combining cliques which include a fill-in edge from the triangulation. An MPSD tree has the MPSs of G as nodes. As in the junction tree, MPSs with nonempty intersections share edges which correspond to these intersections.

Figure 3 shows this process for the well known Asia network of [1, 2, 5]. The original network appears in part (a). Figure 3 (b) shows the moralized graph. The optimal triangulation and its corresponding junction tree are given in parts (c) and (d). Finally, figures 3 (e) and (f) present the MPSD tree and the corresponding MPSs.

Incremental compilation using the MPSD

An overview of this process is given in figure 4. Here G is the original Bayesian network and G^M is its moral graph. G^T is G^M triangulated and T is the corresponding junction tree. T_{MPD} is the MPSD tree for G. At the start of the incremental compilation process, changes are made to G which result in a new network G' The goal is to construct T' and T'_{MPD}. G' is first moralized to produce G'^M. Then, rather than continuing to build the trees for G‘, the fragment g^M of G'^M which is the piece of G'^M in which changes appear, is identified and isolated. Next, g^M is triangulated to form g^T and the corresponding junction tree t. Finally the smaller, updated trees g^T and t are merged with the larger original trees T and T_{MPD}. This creates trees T' and T'_{MPD}, the desired results [2].

$$\begin{tikzcd}
G \arrow[r] \arrow[dd] & G^M \arrow[r] & G^T \arrow[r] & T \arrow[r] & T_{MPD} \arrow[dlll, dashed] \\
 & g^M \arrow[r] & g^T \arrow[r] & t \arrow[r] \arrow[d] & t_{MPD} \arrow[d] \\
G' \arrow[r] & G'^M \arrow[u] & & T' & T'_{MPD}
\end{tikzcd}$$

Figure 4. Overview of the Incremental Compilation process using the MPSD

An example of this process appears in Figure 5. Here the link L → E is removed from the Asia network. In (a) we see the Asia network after the change and (b) shows the fragment of the graph that might be effected by the change. Figure 5 (c) gives the junction tree for this fragment. The original junction tree for Asia and its update after aggregation with the tree in (c) appear in (d) and (e).

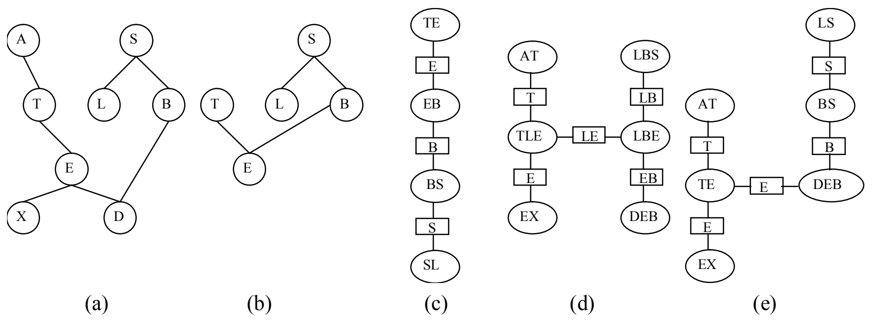

Figure 5. (a) Removal of link L → E from the Asia network. (b) the effected graph fragment. (c) the resulting junction tree. (d) the original junction tree. (e) the final result.

In order to determine which part of the graph might be effected by some change, the set of nodes involved in the change is first constructed. In the Asia example of figure 5, those nodes are L and E. The MPSs that include those nodes are then collected and merged. This forms the graph fragment g of figure 4.

The following changes may be made to an OOBN [1,2] :

- **Adding/removing a link:** Here the root of the instance tree, i.e. the "outermost" object will have to be recompiled but, applying the procedure described earlier, only the relevant part of the graph will need to be processed.
- **Removing a node:** This process involves removing each of the node's links, as above, and then the trivial step of removing the disconnected node.
- **Adding a node:** This is very simple as the new node initially has no links. No recompilation is required.
- **Adding/removing a reference link:** This will cause a change to the interface between the encapsulating object and the instance of the input node. A recompilation of the encapsulating class will occur but, applying IC, only the relevant part of the graph will need to be processed.
- **Changing the class specification:** The involves one or more of the above operations occurring within an object. Each change is handled as described earlier.

5 Implementation of the Algorithm

This project features an implementation of the incremental compilation algorithm for OOBNs. The major functions defined are as follows:

- **asia_test** sets up the Asia network discussed earlier for the testing of the other functions
- **person_test** creates an OOBN containing the attributes of a person. This network was first described in figure 1 and is given in complete detail in figure 6.
- **makeMPSD(junction tree, cliques, fillins)** returns a collection of maximal prime sub-graphs and the corresponding MPSD tree

- **markFragment(nodes, MPSs)** takes the maximal prime sub-graphs and a collection of nodes and returns the fragment of the overall graph that may be effected by a change to the nodes
- **mergeJTrees(old tree, old cliques, new tree, new cliques)** takes a larger junction tree and a smaller fragment and combines them, overwriting the old tree when necessary. This function is useful for finding the new junction tree as the last step of incremental compilation
- **OOBN_IC(OOBN)** takes an OOBN and carries out the incremental compilation process as changes are made

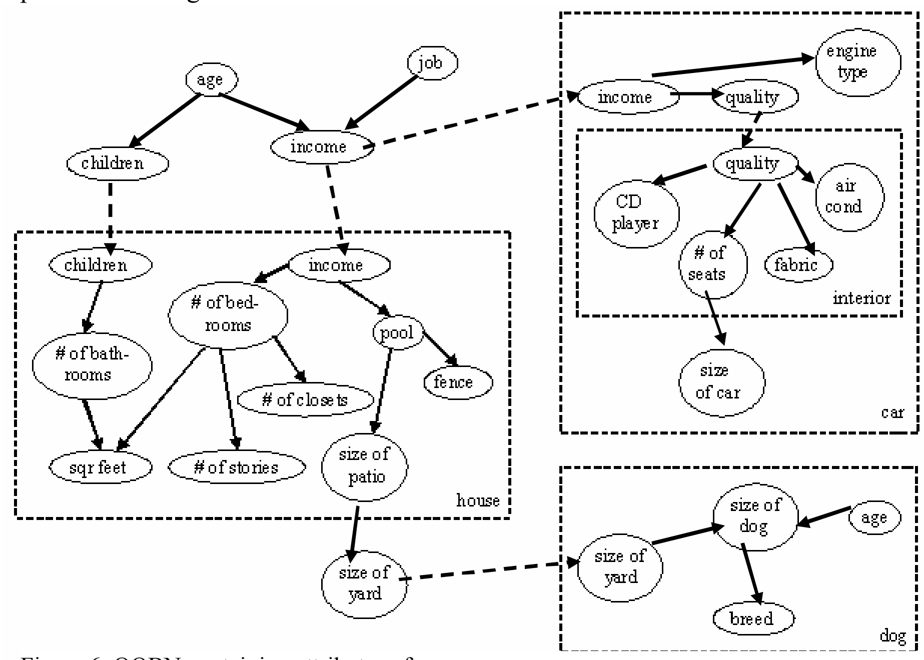

Figure 6. OOBN containing attributes of a person

Conclusion

Object-oriented Bayesian networks have a structure that increases their flexibility and robustness. It also allows their use over larger and more complex domains. As these networks increase in scale, however, even small revisions begin to require extensive recompilation work. Much of this work is unnecessary, as it involves the recreation of the junction tree over the unchanged part of the graph. Incremental compilation makes it possible for only the part of the junction tree which is effected by the change to be updated. This leads to much more efficient inference in large networks.

References

[1] O. Bangso, M. Flores, and F. Jensen. Plug and Play OOBNs. Technical report, Department of computer Science, Aalborg University, Denmark, 2003.

[2] M. Flores, J. Gamez, and K. Olesen. Incremental compilation of a Bayesian Network. In *Proceedings on the Nineteenth Conference on Uncertainty in Artificial Intelligence*. Pages 233-240. Morgan Kaufmann Publishers, San Francisco, 2003.

[3] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[4] D. Koller and A. Pfeffer. Object-Oriented Bayesian Networks. In Dan Geiger and Prakash P. Shenoy, editors, *Proceeding of the 13th Conference on Uncertainty in Artificial Intelligence*. Pages 302-313, San Francisco, 1997. Morgan Kaufmann Publishers, San Francisco.

[5] K. Olesen and A. Madsen. Maximal prime sub-graph decomposition of Bayesian Networks. IEEE Transactions on Systems, Man, and Cybernetics, Part B. bf 32:1:21-31, 2002.

[6] Y. Xiang. Optimization of inter-subnet belief updating in multiply sectioned Bayesian networks. In *Proceedings on the Eleventh Conference on Uncertainty in Artificial Intelligenc*. Pages 680-687. Morgan Kaufmann Publishers, San Francisco, 1995.

[7] Y.Xiang and F. Jensen. Inference in Multiply Sectioned Bayesian networks with extended Shafer-Shenoy and lazy propagation. In *Proceedings on the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Pages 680-687. Morgan Kaufmann Publishers, San Francisco, 1999.