Interactive Tree Decomposition Tool for Reducing System Analysis Complexity

Shahan Yang, Baobing Wang and John S. Baras
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20740, USA
{syang, briankw, baras}@umd.edu

Abstract. We present a graphical tool for the calculation of *treewidth*, a metric on the parametric structure of a system that is intimately tied to the complexity of system analysis. For many graphically describable systems, such as systems of parametric equations, as in a SysML Parametric diagram, or Bayesian networks or even mind maps and writing term papers, analysis of the system is exponential in treewidth and linear in system size. A tool facilitating comprehensive analysis can serve to bring competitive advantage to a systems engineering workflow by reducing costly unanticipated behaviors. Furthermore, a byproduct of computing treewidth is a framework for enumerating computationally compatible distributed algorithms.

Though there are classes for which treewidth computation is tractable (chordal graphs), it is generally NP-complete. For this reason, we pose the problem from the perspective of finding satisficing solutions, exposing choices that can influence the complexity of the resulting system to the designer. A designer can contribute two important things to the structure of the system: a visual intuition about the relationships between the underlying objects and the ability to *change* the relationships themselves at design time to reduce analysis complexity. Having a visual tool that provides instant feedback will help designers achieve an intuitive grasp of the relationship between design decisions and system complexity. As complexity is the root of almost every systems engineering problem, and also something not easily understood, incorporating complexity analysis into a design process should improve resulting system designs.

The tool uses a randomized, anytime algorithm for interactive optimization of treewidth. It presents a sequence of choices to a designer and incrementally lowers an upper bound on system treewidth over time. This algorithm is novel, as few algorithms are targeted at interactivity with a human user.

We present a number of simple examples for using the tool. We show how our tool helps to decompose some example systems, including a quadrotor optimization, a sensor network optimization, a Bayesian network, and a mind map.

Introduction

As systems engineers, we are intimately familiar with using graphical models to describe systems. However, these graphical models are non-unique and there is usually a wide range of behaviorally equivalent ways to model the same problem. One successful application of graphical models from a different community is Bayesian networks (see [Pearl 1988] for a review). In this work, we take some of the mathematical analysis of the graphical models of Bayesian networks and translate it into terms that are more familiar to systems engineers. In a systems oriented fashion, we may think of the Bayesian network as being a *subclass* of the more abstract class of commutative semirings, which has many other subclasses. See Figure 1.

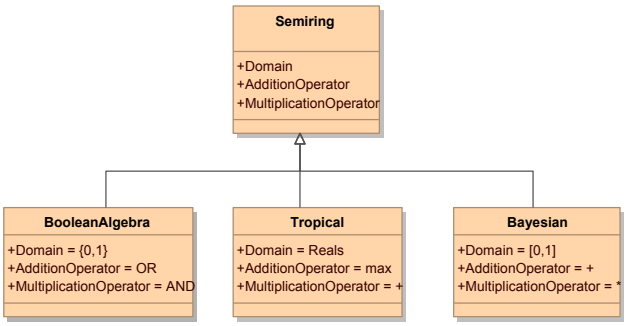

Figure 1. Interpretation of different commutative semirings by subclasses. One particularly interesting subclass from the perspective of systems engineering is the Tropical semiring. It encodes optimization over structures where the overall cost function is the sum of costs over individual components.

The basic essence of each of these cases is to solve a problem described over a network of components where decisions in one component may affect the choices available in another component and there is a global objective that can only be understood by examining the complete space of decisions. Examples of this class of problem include vertex cover, independent set, dominating set, graph k-colorability, hamiltonian circuit, network reliability [Arnborg et al. 1989], and dynamic programming [Bodlaender 1988]. This class of problems is computationally challenging in general and embodies the curse of dimensionality. Using structural decomposition techniques of systems engineering is one approach towards solving these problems. However, there are very few tools available for doing this systematically. We present a tool that achieves this.

It turns out that complexity is exponential in *treewidth* and *linear* in problem size. The intuition behind this result is that problems on graphs are difficult to solve due to the presence of loops. Removing the loops by multiplexing variables (aggregating them into objects) can lead to the tree decompositions of graph problems. Once the problem is in the form of a tree, then summary propagation is a viable technique for solving the problems. Multiplexing variables creates local complexity roughly in proportion to the number of variables tied together. More precisely, if we consider a discrete context, the space that needs to be explored is the product of the number of discretization bins, ie, if there are N variables with D quantization levels each in an aggregate object, then the complexity of analyzing that object in D^N. The complexity of the overall system is the summation of the complexity of analyzing each system independently. This sum is dominated by largest exponent in the system, which is precisely what the treewidth measures (see Appendix for details).

Our Contribution. In [Yang et al. 2011], we presented some theory of tree decomposition. This work describes a prototype that we have been working on to make the theory usable and many examples of problem solved using this tool. The main contribution of this work is an interactive tool for measuring treewidth of systems. A byproduct of this measurement is a system tree decomposition that can be used for analysis. We work out many examples using the tool and describe the algorithm used.

Tool Development and Case Studies

Tool Development. To facilitate the usage and enhance the understanding of the tree search algorithm, a user-friendly GUI is developed in Java, which enables users to control the execution of the algorithm interactively and view the result graphically. The GUI is shown in Figure 2, painting the relationship graph for the parameters in our case study, which will be explained later.

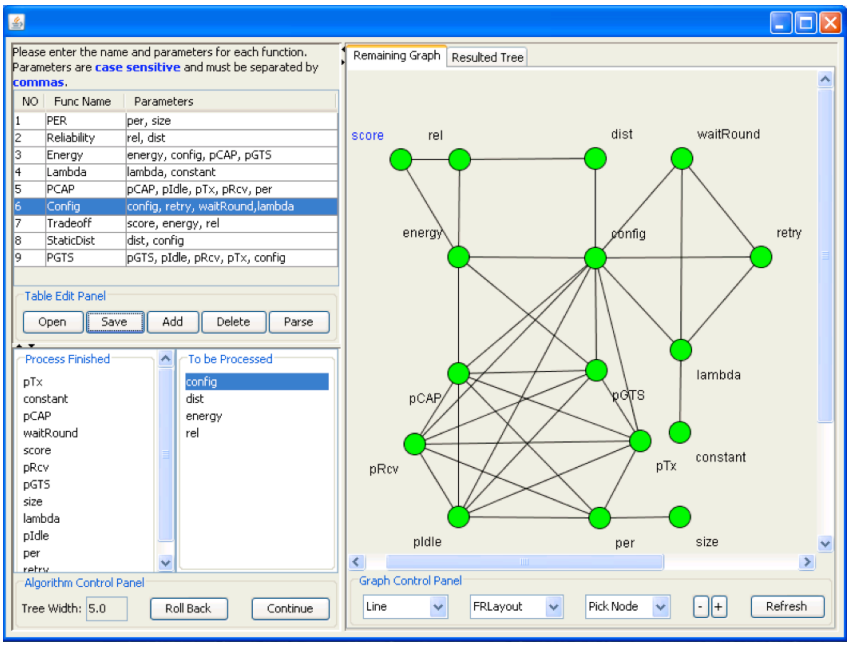

Figure 2. GUI and the generated relationship graph for the case study

Function definitions can be loaded from a pre-saved file, or input to the table in the upper left corner, by specifying their names and parameters. Then they can be checked and parsed to the data structures used in the tree search algorithm. If all functions are defined correctly, the tree search algorithm will process the chordal vertices automatically. The algorithm control area in the lower left corner will provide the list of unprocessed parameters and the parameters that have already been processed. Users can select an unprocessed parameter to continue the algorithm and the resulting treewidth will be calculated and updated incrementally. Users can also roll back the algorithm to its previous state and make a different choice, potentially with a smaller tree width.

An observer thread is running in the background to update the relationship graph of the parameters and the resulted tree of cliques periodically, which are shown in the right tabbed panel. Users can also update them instantly by clicking the "Refresh" button. Based on the characteristics of the graph and the tree, users can select different layout algorithms to place the vertices automatically to get a better view, or arrange them manually. The Java Universal Network/Graph (JUNG) Framework [JUNG 2011] is used for data visualization.

Wireless Sensor Networks. We consider the trade-off analysis between energy efficiency and transmission reliability in wireless sensor networks, where the IEEE 802.15.4 standard is applied as the media access control protocol. For simplicity, we only provide high-level abstract functions here, emphasizing the abstract relationships between the parameters in each function. More details are available in [Wang et al. 2011]. The following functions are used in this trade-off analysis, in which the blue parameters are their outputs:

Tradeoff(score, energy, rel) = 0. This function specifies the trade-off rules between energy efficiency and transmission reliability.

Reliability(rel, dist) = 0. This function calculates the reliability based on the static distribution of the Markov Chain model in [Wang et al., 2011], which models the peer-to-peer communications for time-critical applications in wireless sensor networks using the enhanced IEEE 802.15.4 protocol.

StaticDist(dist, config) = 0. This function computes the static distribution, based on the configuration information specified for the protocol.

Config(config, retry, waitRound, lambda) = 0. This function processes the protocol parameters, such as the maximum retransmission times and the maximum waiting rounds, to generate the configuration information.

Lambda(lambda, constant) = 0. This function is defined to simplify the *Config* function, by processing other protocol-specific constants and feeding the result to the *Config* function.

Energy(energy, config, pGTS, pCAP) = 0. This function calculates the expected energy consumption for each transmission, based on the configuration information, and the expected energy consumptions in the contention-based access period (CAP) and in the guranteed time-slot period (GTS).

PGTS(pGTS, config , pIdle, pRcv, pTx) = 0. This function computes the expected energy consumption in the GTS period, based on the transmission power, receiving power, the power in the idle state and the configuration information.

PCAP(pCAP, pIdle, pRcv, pTx, per) = 0. This function is very similar to the *PGTS* function, except that the packet error ratio (PER) is considered here.

PER(per, size) = 0. This function simply calculates the PERs based on packet sizes.

The generated tree of cliques is shown in Figure 3, in which each vertex stands for a clique in the relationship graph of parameters, and the edge direction represents the reverse order of information propagation. When a vertex has received the information from all its children, it begins to calculate the parameters in its clique locally and propagate the result back to its parent. Now suppose every parameter can have 10 different values (continuous parameters can be sampled discretely). Then the complexity can be reduced significantly to:

$10^2 \times 2 + 10^3 \times 3 + 10^4 \times 2 + 10^5 + 10^6 = 1123200$
compared to 10^{16} in the original computation.

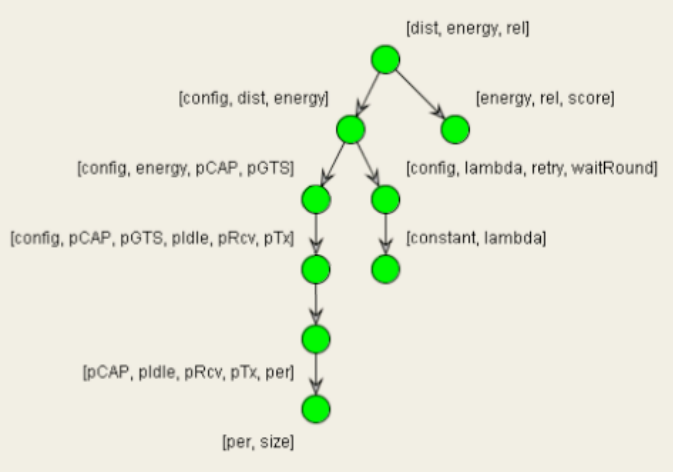

Figure 3. The generated tree of cliques.

Quadrotor Example. Figure 4 shows the relationships between variables in a quadrotor that is designed to fly out to a specified destination, land, perch and take observations. It uses a parametric diagram, which is exactly equivalent to a factor graph, in being a bipartite graph that has variable nodes in one partition and function nodes in the other partition. The fact that it is a factor graph means that summary propagation can be used as a solution algorithm with the correct interpretation of the summation and multiplication operations. This particular parametric diagram reflects a query on the tradeoff between range and cost. The constraint *Tradeoff* is a query in this case and modifies the structure of the parametric diagram, which in

turn has an impact on the resulting tree decomposition. In working with this system of tree decompositions, this dependency of structure on the query occurs often. If a query relates two variables that were previously unrelated, then a link must be added to the graph reflecting this added coupling.

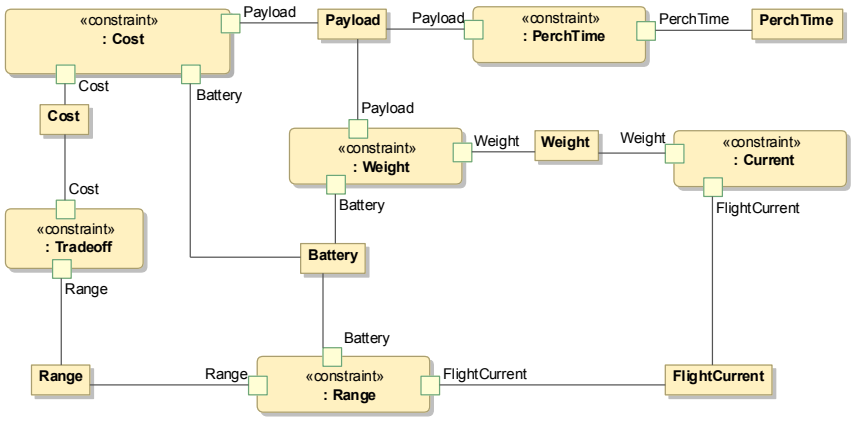

Figure 4: Parametric diagram for high level tradeoffs of a quadrotor. Consider the constraints shown in this diagram. The Tradeoff constraint reflects the fact that we are interested in the relationship between cost and range of the quadrotor. As indicated by the Cost constraint, the cost value is determined entirely, in this model, by choice of payload and battery. The weight is also determined by these two variables, as shown by the Weight constraint. The range of the quadrotor, as indicated by the Range constraint, is determined by the choice of battery and the power requirements expressed as current. The flight current needed is determined by the weight of the quadrotor, as indicated by the Current constraint. Finally, there is a perch time variable that is solely determined by the payload as shown in the PerchTime constraint.

The different functions specify feasible regions of values for the various parameters, but there is a locality structure to this specification because certain variables are not directly related. For example, the current needed to fly the quadrotor depends on the weight (as indicated in the *Current* constraint, but these variables are not directly connected to the cost). Weight depends on the battery and payload chosen, which then directly contribute to the cost.

We would like to determine all feasible configurations with respect to range and cost in our trade study. We shall assume that every parameter takes on a discrete set of values, which could come from discretization. Naively, there are 7 variables in this system. Evaluating over all of them simultaneously using brute force could involve D^7 evaluations, where D is the number of discretization levels.

Figure 5 shows the input to the tool. Compare this to Figure 4. The name column contains exactly entries corresponding to the constraints of the parametric diagram and the parameter column contains the arguments to those constraints.

NO	Func Name	
1	Cost	Cost,Battery,Payload
2	Tradeoff	Cost,Range
3	Range	Battery,Range,FlightCurrent
4	Weight	Weight,Battery,Payload
5	PerchTime	Payload,PerchTime
6	Current	FlightCurrent,Weight

Figure 5. Input to the tool representing the relationships between the variables.

Figure 6 shows the initial flattened topology of the quadrotor (the functional dependence graph in the language of [Yang 2011]).

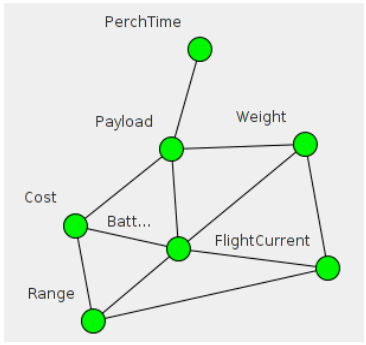

Figure 6. This shows the initial graph extracted from the input relationships of Figure 5. Note that this graph is not chordal which means that the designer will need to choose additional variable couplings for the system to decompose.

At this point, the designer has a decision to make because the only simplical node is PerchTime, which is eliminated by the algorithm. Elimination on the rest of the nodes creates fillins. To make this decision, the designer thinks about which variables most naturally fit together with respect to the fillins created. Since the relationship between weight and range is the most intuitive, the next elimination is FlightCurrent, which creates a fillin between weight and range. This is shown in Figure 7.

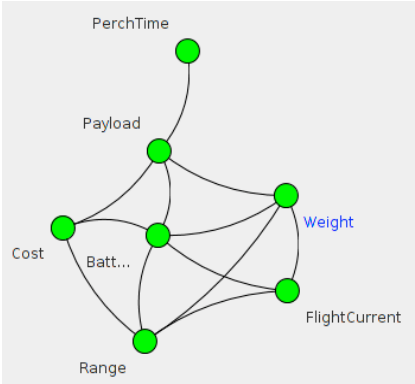

Figure 7. This shows the graph of Figure 6 under the condition of eliminating the variable FlightCurrent. This creates a fillin between weight and range, which are intuitively related, which is why FlightCurrent is chosen for elimination rather than another variable. One more fillin will be neeed to complete the system decomposition.

A link was added between weight and range, coupling these two variables within the analysis even though there is no immediate equation describing this relationship. This is an artifact of performing the tree decomposition of the system. Of the remaining variables, the next most intuitive relationship is the one between payload and range, so we eliminate cost next, which requires payload and range to be coupled. Figure 8 shows the result.

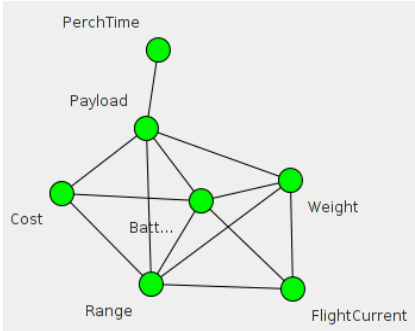

Figure 8. Shows the chordally decomposed system with the two fillins, weight-range and payload-range. This system is chordal and has a tree decomposition. The tree structure is

somewhat apparent in this diagram. It consists of three tetrahedrons that are stacked next to each other and a tail consisting of the PerchTime, which is only loosely coupled with the rest of the system.

The tool produces Figure 9 as the tree decomposition of the system using these hints from the designer.

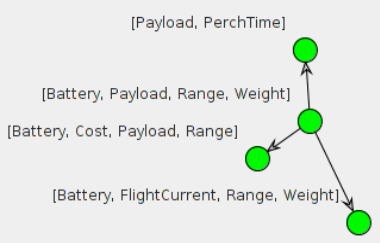

Figure 9. The resulting tree decomposition from the analysis performed in Figures 4-8.

The last step in the analysis described in [Yang et al. 2011] is to map the original constraints and functions back to the resulting join tree. The most natural language for expressing this is a block diagram, as shown in Figure 10. There is always a way to assign the constraints back to the aggregations in such a way that every constraint has all its parameters in its local block. Though mapping is not unique in general, it happens to be unique in this case.

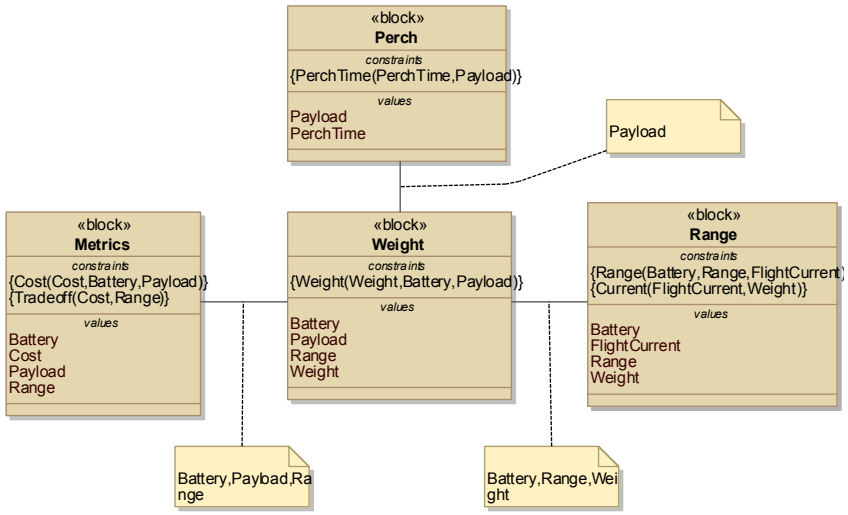

Figure 10. This shows the completed block diagram of the tree decomposition of the quadrotor. In this example, the assignment of the constraints back to the structure is unique, but that is not necessary in general. The constraints are the same as those constraints in the input parametric diagram and the variables are assigned to blocks exactly as the tree decomposition of Figure 7 would indicate. The associations between blocks are labeled according to the shared variables.

To analyze the system shown in Figure 10, we use a very simplistic algorithm using sets. Each block, Perch, Metrics, Weight, and Range, can be thought of as describing a set of feasible points based on the constraints. The overall space of the system can be described as the intersection of the spaces described in the blocks. To apply summary propagation, what we use set intersection as the multiplication operation and projection as the summation operation. Since this is a trade study, our goal is to evaluate the Metrics block. Figure 11 depicts the general strategy of evaluation. Using the decomposition of Figure 10 reduces the complexity of analyzing the system from D^7 down to $3D^4+D^2$. This is a significant reduction. Suppose, for example, we use a grid of 20 points. $20^7 = 1{,}280{,}000{,}000$ while $3 \cdot 20^4 + 20^2 = 480{,}400$ which is orders of magnitude fewer samples.

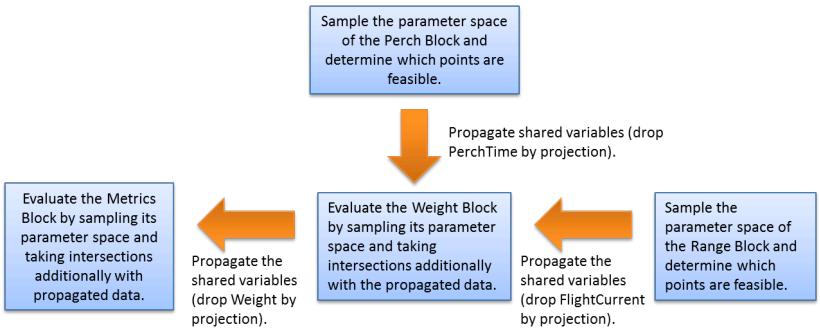

Figure 11: Summary propagation applied to the block diagram of Figure 10. We treat each of the blocks as sets. The overall system is understood as the intersection of all the sets. We can use a generalized version of summary propagation to efficiently run queries on this structure.

Traffic Intersection. Figure 10 shows the interference graph for a traffic intersection. Figure 11 shows a join tree computed by the tool. This is essentially a Boolean satisfaction problem where we are searching for all the satisfying instances.

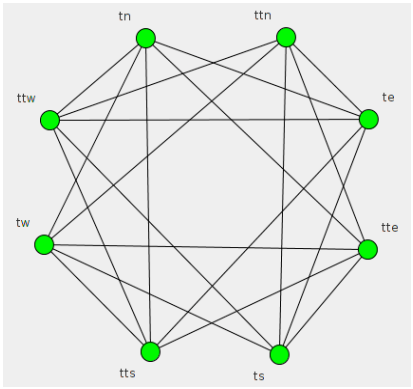

Figure 12. TTW denotes the traffic light controlling traffic from the west that is turning left (north) and TW denotes the traffic light controlling traffic coming from the west and going east. There is a link between two variables if they are not permitted to be simultaneously green.

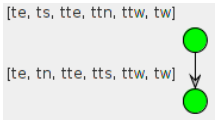

Figure 13. One of the possible join trees generated by the tool. Since Figure 12 is not chordal, this is not a unique decomposition. Note that the interface or shared variables are te, tte, ttw and tw.

It is not immediately clear how to analyze the system using the join tree given in Figure 13 since the variables are highly coupled. We take the nodes of Figure 12 and rearrange them so that the interface variables te, tte, ttw and tw are in the middle separating the rest of the variables. Figure 14 shows the result of this manipulation.

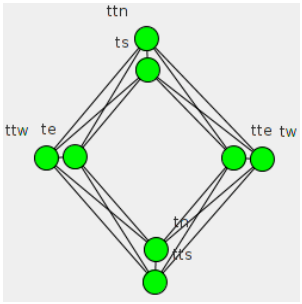

Figure 14. This graph, derived from the join tree of Figure 13, reveals the structure of the traffic intersection dependence graph intuitively. We can see that ttw | te and tte | tw disable the opposite pairs of lights. Among themselves, they also have some structure. They do not oppose each other at all, in fact, so the compatible configurations are ttw and (tw or tte). The other possible configurations are symmetrical to these.

Join Tree for a Bayesian Network. The traditional usage for junction trees is performing inference on Bayesian networks. Figure 15 shows a Bayesian network depicting a disease diagnosis inference. Figure 16 shows its corresponding loop-free join tree that is suitable for inference.

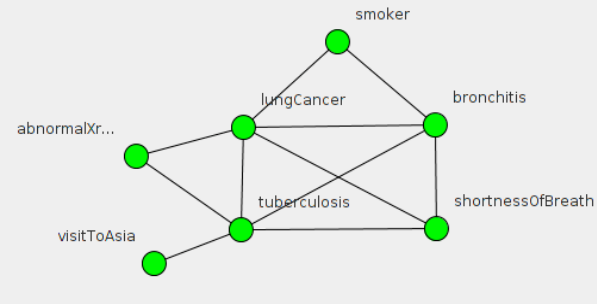

Figure 15. A graph showing a Bayesian network for diagnosing lung conditions. This graph is coincidentally chordal, so the algorithm converges immediately to a unique solution.

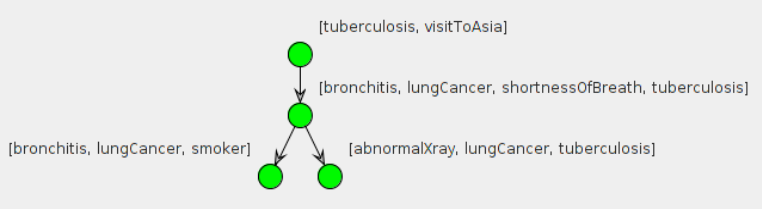

Figure 16. A graph depicting the join tree of the Bayesian network from Figure 15. The join tree is loop free so summary propagation is an exact inference algorithm.

Mind Mapping. One problem that occurs when writing a paper is taking a graph that represents the ideas in the paper and linearizing it into an outline. This tool can help in doing this. Figure 17 shows the graph structure of some concepts used in a different paper.

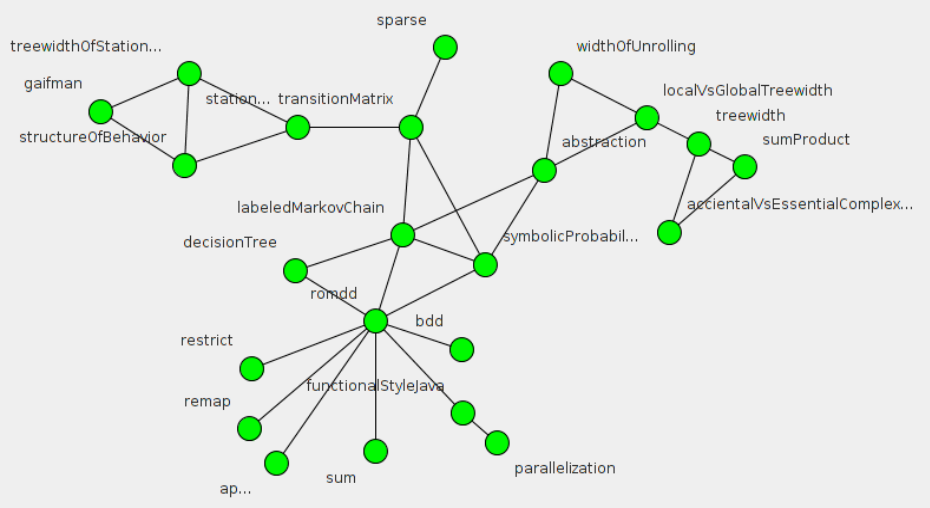

Figure 17. A map of concepts in a paper. This graph is coincidentally chordal.

Figure 18 depicts the tree representation of the graph in Figure 17. The tool helps convert the graph representation into a tree representation of the information. The tree can then be conveniently linearized into an outline.

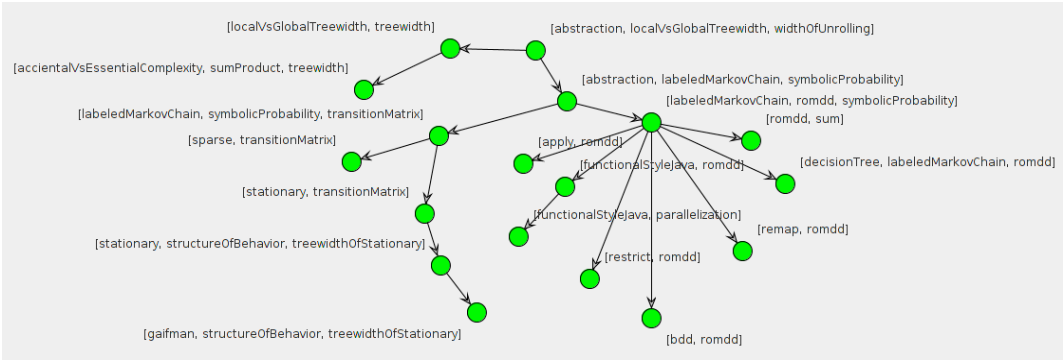

Figure 18. A tree view of the contents of Figure 17. This tree can be directly translated into an outline for a paper. In fact, many outlines can be produced from this tree. The first degree of freedom is the choice of a root node. Since a property of trees is that a unique path exists between any pair of nodes, identifying a root induces a partial order over the tree. The suborderings must also be determined to linearize the tree structure.

Discussion

One interesting property of the technique is how counterintuitive these join trees are from the perspective of creating block diagrams. However, looking at Figure 6 reveals an interesting relationship between the geometry of the chordal decomposition and the resulting block structure. The Battery and Range variables are shared by the three blocks. It is apparent in the geometry that these two variables form an axis which connect the three tetrahedrons and thus Battery and Range are shared variables over three of the blocks in the block diagram. We are not accustomed, as engineers to expressing decompositions using shared variables, although it is apparent that this is natural because the constraint structure has a both locality and a dependence structure. Having a tool for performing this analysis certainly helps in finding the tree decompositions.

As shown through the examples, this is a very general technique that can be applied to many domains. In the examples of this paper, the sets are static in nature. In [Yang et al. 2012], we show how the same technique of composition and projection can help in the formal analysis of *dynamic* Bayesian networks.

Future Work

The current tool only implements the basic join tree algorithm that can be found in [Jensen 2007]. This algorithm utilizes the fact that ordering the cliques in reverse of the elimination order that generates them provides a way to constructively attach the cliques into a clique. [Ibarra 2009] describes a means to generate all possible join trees. The cliques generated by the algorithm are unique and can be used to create the clique-separator graphs of [Ibarra et al. 2009]. The specific join tree could be a design decision for the system and it would be good to extend the tool to allow interaction over these structures. In [Yang et al. 2011], a further addition to the clique separator is to remap the original constraints back to the resulting cliques. This mapping is proven to exist for the clique decomposition but it is not unique, so the tool should also assist with this step.

It should also be possible to map parametric diagrams directly into inputs for this tool, given how similar they are structurally. Automated generation of block diagrams should then be possible as well from the output of the tool. The full version of this paper will likely include these features.

Conclusions

We have presented a tool that uses an interactive method to compute junction trees and show how the technique can be applied in structuring the analysis of broad range of systems. Theoretically, problems that can be encoded as commutative semirings are amenable to analysis by this technique, but it is not limited to this domain. We believe this tool and graphical decomposition technique could be of use to many systems engineers. It is complexity aware and generates decompositions that are amenable to localized computational analysis.

Appendix: Formal Description

Definition 1 *Define a* **system** *as the tuple*

$$P = < \mathcal{L}, P_1(\mathcal{X}_1), \ldots P_M(\mathcal{X}_M) >,$$

with $\mathcal{L} = \{\Sigma_1, \ldots, \Sigma_n\}$ and $\mathcal{X}_i \subseteq \mathcal{L}$ for $i = 1 \ldots M$. Each $\Sigma_i \in \mathcal{L}$ is a set corresponding to the domain of a system variable $\mathbf{x}_i$. Each P_i, for $i = 1 \ldots M$ is a general component that influences the variables with domains $\mathcal{X}_i$.

Observe that in general, the $\mathcal{X}$ values are not disjoint. In this model, the sharing of variables between components indicates communication between those components. We may draw in SysML, a parametric diagram to capture any *system* as defined above. (See Figure 2 for an example.) The constraint blocks are the components P_i of the system and the variables correspond to the variables Σ_i of the system. Each constraint block P has its own associated list of variables $\mathcal{X}$.

Definition 2 *Define the* **flattening** *of a system P as the graph $G = < \mathcal{L}, E >$ with*

$$E = \{(x, y): \exists i \in \{1, \ldots, M\} x, y \in \mathcal{X}_i \wedge x \neq y\}.$$

Every parameter set defined by $\mathcal{X}_i$, for $i = 1 \ldots M$, induces a clique of mutually connected nodes in the flattened graph, G.

Definition 3 *Define the* **elimination** *of a node* $\Sigma \in \mathcal{L}$ *from the graph* $G =< \mathcal{L}, E >$, *written* $\oplus_\Sigma G$, *as the graph* $G' =< L\backslash\{\Sigma\}, E' >$, *where* E' *is defined*

$$(E\backslash\{(x,y):\Sigma \in [x,y]\}) \cup F$$

with F being the set of links in the clique induced by the set of neighbors, $N(\Sigma)$ [1], of Σ.

This definition reflects the fact that in a semiring context, eliminating a variable (by summation) first entails collecting all the constraints that include that variable. The induced clique over the neighbors of Σ effects the necessary collection of constraints.

Let $O =< o_1, \ldots, o_N >$ be a permutation of $\mathcal{L}$. All such Os can be viewed as elimination orderings.

Definition 4 *The* **sequence of graphs induced by an elimination ordering** O, $< G_1(O), \ldots, G_{N+1}(O) >$, *is defined* $G_1(O) = G$ *and for* $i = 1 \ldots N$, $G_{i+1}(O) = \oplus_{o_i} G_i(O)$.

It is clear from the above definition that $G_{N+1}(O)$ must be an empty graph because all nodes are eliminated. This elimination induces also a sequence of cliques in the graph.

Definition 5 *The* **sequence of cliques induced by an elimination ordering** O, $< C_1(O), \ldots, C_N(O) >$, *is defined as* $C_i = N_{G_i(O)}(o_i) \cup \{o_i\}$, *for* $i = 1 \ldots N$. $N_{G_i(O)}(o_i)$ *is the set of neighbors of* o_i *in the graph* $G_i(O)$ *from the sequence of graphs induced by* O.

Note that in this definition, there may be cliques $C_i(O)$ that are contained in other cliques $C_i(O) \subset C_j(O)$ with $i \neq j$.

Definition 6 *The* **width** *of graph* G *with respect to ordering* O, $W_G(O)$ *is defined as the size of the maximum size clique in the sequence of induced cliques minus* 1*:*

$$W_G(O) = \max_i |C_i(O)| - 1.$$

The extra minus 1 ensures that the treewidth of a tree is 1.

Definition 7 *The* **treewidth** W *of a system* P *may be defined in terms of an optimization over elimination orderings* O *of the width of a system.*

$$W = \min_O W_G(O)$$

or expanding Definition 6

$$W = \min_O \max_i |C_i(O)| - 1. \qquad (1)$$

Theorem 1 *The treewidth generated by elimination orderings as in Definition 7 gives the minimal tree decomposition of the system.*

The optimization in equation (1) is NP-hard [Arnborg 1987]. Since such problems are not tractable in general, we seek heuristic or approximate solutions and give the user the ability to interactively solve the problem. Random search is a powerful solution technique for many NP-hard problems. We present a hierarchical algorithm utilizing random search that computes a sequence of upper bounds on treewidth.

---

[1] $N(\Sigma)$ is defined as the set $\{x \in \mathcal{L}: (x, \Sigma) \in E\}$. Since Definition 2 forbids self links, $\Sigma \notin N(\Sigma)$.

Definition 8 *A node is* **simplical** *if all of its neighbors are mutually connected.*

Since the optimization parameter is a permutation, the search space has a tree structure. There are subsequences of the permutation that are determined and do not need to be searched. Specifically, when there are *simplical* nodes in the network, they can be eliminated immediately, furthermore when there are multiple simplical nodes in the network, the order of their elimination does not affect the resulting treewidth.

Theorem 2 *Let x and y be two simplical nodes in the graph G. Then*

$$\ominus_{y}\ominus_{x} G = \ominus_{x}\ominus_{y} G.$$

Elimination of simplical nodes is commutative.

Eliminating a simplical node from a graph creates no fillins because all its neighbors are mutually connected. Eliminating x therefore removes x from the set of nodes and any links including x. The same is true of eliminating y. The resulting graph after eliminating both x and y from the graph $G = < \mathcal{L}, E >$ has the nodes $\mathcal{L}\backslash\{x, y\}$ and the set of edges not including x or y. It is clear that the order of x and y does not matter to the resulting graph.

Theorem 3 *Let O be an elimination order where o_i and o_{i+1} are both simplical in the graph $G_i(O)$. Then the elimination $O' = < o_1, \dots, o_{i-1}, o_{i+1}, o_{i-1}, o_{i+2}, \dots, o_N >$ induces the same width as O, $W_G(O) = W_G(O')$.*

O and O' differ only in swapping o_i and o_{i+1}, so $G_i(O) = G_i(O')$. Also, by Theorem 2,

$$G_{i+2}(O) \dots G_{N+1}(O) = G_{i+2}(O') \dots G_{N+1}(O').$$

STS

$$\max\{|C_{i+1}(O)|, |C_{i+2}(O)|\} = \max\{|C_{i+1}(O')|, |C_{i+2}(O')|\}.$$

There are two cases to consider, when o_i and o_{i+1} are neighbors and when they are not. In the case where o_i and o_{i+1} are neighbors, the fact that o_i is simplical in graph $G_i(O)$ implies that o_{i+1} is connected to all of its neighbors and vice versa. This implies that o_i and o_{i+1} are part of the same clique in $G_i(O)$. So o_i and o_{i+1} are symmetrical and indistinguishable WRT to the sizes of the cliques formed. In the case where o_i and o_{i+1} are not neighbors, it is clear that $C_{i+1}(O) = C_{i+2}(O')$ and $C_{i+2}(O) = C_{i+1}(O')$ because the eliminations are completely independent of one another. Theorem 3 states that elimination of simplical nodes is commutative WRT the width of the resulting graphs.

Corollary 1 *Simplical nodes can be eliminated in any order without impacting the width of the graph.*

The algorithm for finding treewidth can be sketched out as follows.

1. Eliminate all simplical nodes (in any order).
2. If any nodes remain, eliminate one randomly.
3. If any nodes remain, return to step (1).

This algorithm eventually finds all permissible elimination orders if run enough times as there are only finitely many elimination orders. The probability of not finding a particular order is roughly $(1 - \frac{1}{N})^K$ where N is the number of orderings and K is the number of iterations.

We would like to speed up convergence of the above algorithm and to do so, we collect statistics about the decisions made and use those statistics to improve future guesses. The algorithm that we implement here builds a tree of prespecified *bounded* size. See Figure 19. The system uses a predefined size for the search tree, bounded by memory constraints. At each node of the tree, a metric representing the sample mean under blind search from that node is maintained. The algorithm alternates between blind search and directed search. The blind search samples alternatives uniformly. The directed search samples branches in proportion to the expected value of the score function, which in this case is e^{-w} where w is the expected width of a branch.

In order to maintain a nominal tree size, the algorithm runs in two phases. An expansion phase and a pruning phase. In the expansion phase, nodes are added based on their weight. We start with N nodes and every time a branch occurs, we multiply N by the weight of that branch which is equal to $\frac{w_i}{\sum_j w_j} \leq 1$. This formula is applied recursively, so as the tree descends, the weights decrease. A branch is added if the remaining weight is ≥ 1. A pruning phase removes branches with weight less than 1. It is possible for branches to be pruned because weights change as more measurements become available. The depth of the tree that is kept in memory varies as a function of the the expected score. Paths in the tree that score better will naturally expand deeper in the exploration.

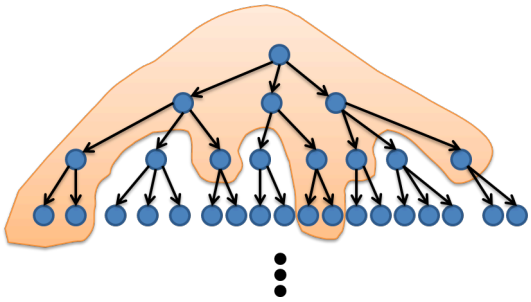

Figure 19. This figure shows the search tree through the permutation space. The nodes are the different eliminations possible and the graph extends downwards. The shaded region represents just those nodes held in memory for which statistics are collected.

References

Arnborg, S. and Proskurowski, A. 1989. "Linear time algorithms for NP-hard problems restricted to partial k-trees." *Discrete Applied Mathematics* 23(1): 11–24.

Arnborg, S., Corneil, D. G. and Proskurowski, A. 1987. "Complexity of finding embeddings in a k-tree." *SIAM J. Algebraic Discrete Methods* 8: 277–284.

Bodlaender, H. 1988. "Dynamic programming on graphs with bounded treewidth." *Automata, Languages and Programming*: 105–118.

Ibarra, L. 2009. "The clique-separator graph for chordal graphs." *Discrete Appl. Math.* 157: 1737–1749.

Jensen, F. V. and Nielsen, T. D. 2007. *Bayesian Networks and Decision Graphs*. Springer Publishing Company, Incorporated, 2nd edition.

JUNG Framework Development Team. 2011. "JUNG: the Java Universal Network/Graph Framework (Version 2.0.1)." Accessed 7 November. http://jung.sourceforge.net/index.html.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Wang, B. and Baras, J. S. 2011. “Performance Analysis of Time-Critical Peer-to-Peer Communications in IEEE 802.15.4 Networks.” Paper presented at the Proceedings of the IEEE Intl. Conference on Communications (ICC), Kyoto, Japan, 5–9 June.

Yang, S. and Baras, J. S. 2011. “Factor Join Trees for Systems Exploration.” Paper presented at 23rd International Conference on Software & Systems Engineering and their Applications, Paris, France.

Yang, S. and Zhou, Y. and Baras, J. S. 2012. “Compositional Analysis of Dynamic Bayesian Networks and Applications to CPS.” Submitted for publication.

Biography

Shahan Yang is a postdoctoral researcher with the Institute for Systems Engineering, at the University of Maryland. He completed his doctorate at the University of Maryland in 2007 on automated analysis security of ad hoc wireless routing protocols. Since then he has worked as a Principle Engineer at BAE Systems as part of several IRAD efforts in security and wireless networking. Prior to this, he was a consultant working for defense research agencies, telecommunications companies and in bioinformatics. His current research interest is the integration of systems engineering and formal methods and probabilistic model checking. He is also working on systems engineering based approaches to cyber physical systems.

Baobing Wang is a 3rd-year Ph.D student in the Department of Electrical and Computer Engineering at the University of Maryland, College Park. He is working with Prof. John S. Baras on Model-Based Systems Engineering for cyber-physical systems, especially the system modeling, design and synthesis for wireless sensor networks. Mr. Wang received his B.S. degree in Computer Science in 2006 from the Harbin Institute of Technology, China, and MPhil degree in Computer Science in 2009 from the City University of Hong Kong, Hong Kong SAR.

John S. Baras holds a permanent joint appointment as professor in the department of electrical and computer engineering and the Institute for Systems Research. He was the founding director of ISR, which is one of the first six National Science Foundation engineering research centers. Dr. Baras is the Lockheed Martin Chair in Systems Engineering and is the founding and current director of the Center for Hybrid and Satellite Communication Networks, a NASA commercial space center. He also serves as a faculty member of the university's Interdisciplinary Program in Applied Mathematics and an affiliate professor in the Computer Science Department.