Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 16 (2013) 138 – 147

Procedia Computer Science

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

Interactive Tree Decomposition Tool for Reducing System Analysis Complexity

Shahan Yang, Baobing Wang, and John S. Baras*

Department of Electrical and Computer Engineering, Institute for Systems Research, University of Maryland, College Park

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.

In this paper, we pose this NP-complete 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.

1. 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 [1]

*Corresponding author. Tel.: +1-301-405-6606; Fax: +1-301-314-9218; E-mail: baras@umd.edu.

1877-0509 © 2013 The Authors. Published by Elsevier B.V.
Selection and/or peer-review under responsibility of Georgia Institute of Technology
doi:10.1016/j.procs.2013.01.015

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 Fig. 1.

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 [2], and dynamic programming [3]. 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.

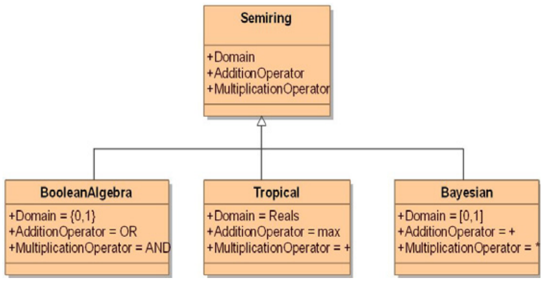


Fig. 1: Interpretation of commutative semirings by subclasses. One particularly interesting subclass from the perspective of systems engineering is the Tropical semiring. It encodes optimization over structures where the overall cost function is the sum of costs over individual components.

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 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, i.e., 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 the largest exponent in the system, which is precisely what the treewidth measures.

Our Contribution. In [4], we presented some theory of tree decomposition. This work describes a prototype that we have been working on to make the theory usable and several examples of problems 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 algorithm that can be used for analysis. We work out many examples using the tool and describe the algorithm used.

2. Related Work

Guenov [5] estimates the complexity to aid high-level designers in comparing alternatives during pre-competitive studies or during the architectural design process of composition systems. This approach is based on Boltzmann's entropy concept to measure the distribution of functional couplings in the system's decomposition. However, no mechanism is proposed to reduce the complexity.

Lu et al. [6] considered that the "overall difficulty" of an engineering system design consists of "inborn complication" due to custom requirements and external constraints as well as "acquired complexity" associated with uncertainty in satisfying the functional requirements caused by design decisions. They introduced the Axiomatic Design Theory and the Design-centric Complexity Theory to guide the creation and improvement of complex engineering systems. However, they cannot provide instant feedback on the impact of design decisions on the complexity.

Clarke [7] presented a framework to reduce the complexity of temporal logic model checking in systems composed of many parallel processes by using additional interface processes to model the environment for a component. These interface processes are typically much simpler than the full environment of the component. By composing a component with its interface processes and then checking properties of this composition, one can guarantee that these properties will be preserved at the global level. However, this partitioning is ad-hoc and depends heavily on rules of thumb and the expertise of systems engineers.

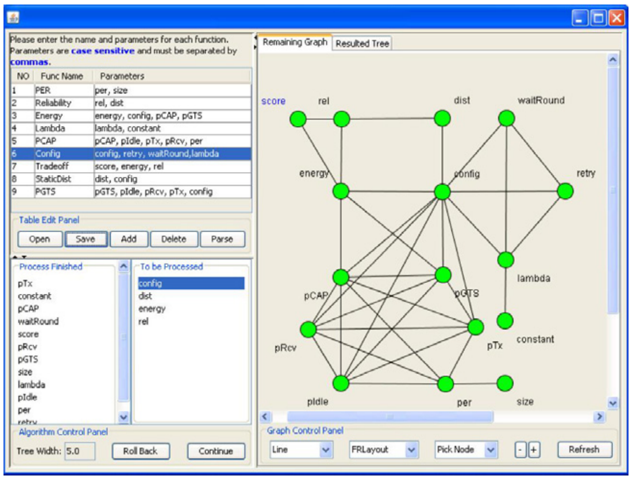
A large number of model checking algorithms are based on the symbolic model checking method, which was first proposed in [8]. This method avoids building a state graph by using Boolean formulas to represent sets and relations. A variety of properties characterized by least and greatest fixed points can be verified purely by manipulations of these formulas using Ordered Binary Decision Diagrams. Instead of enumerating reachable states one at a time, the state space is traversed much more efficiently by considering large numbers of states at a single step. Such state space traversal is based on representations of state sets and transition relations as formulas, binary decision diagrams or other related data structures as in [9]. Several tools that can reduce the complexity of formal verification based on symbolic model-checking and homomorphic reduction are discussed in [10]. While these tools can reduce the complexity of formal verification efficiently, they cannot provide guidance on how to improve the system designs to facilitate formal verification further.

3. Tool Development and Case Studies

3.1. Tool Development

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

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 [4] 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 treewidth.



Please enter the name and parameters for each function. Parameters are **case sensitive** and must be separated by **commas.**

NO	Func Name	Parameters
1	PER	per, size
2	Reliability	rel, dist
3	Energy	energy, config, pCAP, pGTS
4	Lambda	lambda, constant
5	PCAP	pCAP, pIdle, pTx, pRcv, per
6	Config	config, retry, waitRound,lambda
7	Tradeoff	score, energy, rel
8	StaticDist	dist, config
9	PGTS	pGTS, pIdle, pRcv, pTx, config

Process Finished	To be Processed
pTx	config
constant	dist
pCAP	energy
waitRound	rel
score	
pRcv	
pGTS	
size	
lambda	
pIdle	
per	
retry	

Tree Width: 5.0

Fig. 2: GUI and the generated relationship graph for the case study

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[1] is used for data visualization.

3.2. 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 [11]. The following functions are used in this trade-off analysis, in which the blue parameters are their outputs:

[1]`http://jung.sourceforge.net/index.html`

- **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 results 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 Fig. 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: $10^2 * 2 + 10^3 * 3 + 10^4 * 2 + 10^5 + 10^6 = 1123200$, compared to 10^{16} in the original computation.

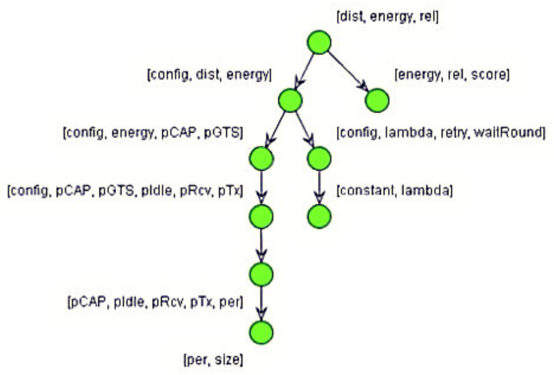

[dist, energy, rel]
[config, dist, energy]
[energy, rel, score]
[config, energy, pCAP, pGTS]
[config, lambda, retry, waitRound]
[config, pCAP, pGTS, pIdle, pRcv, pTx]
[constant, lambda]
[pCAP, pIdle, pRcv, pTx, per]
[per, size]

Fig. 3: The generated tree of cliques

3.3. Quadrotor Example

Fig. 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.

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.

Fig. 5a shows the input to the tool representing the relationships between the variables. Compare this to Fig. 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.

Fig. 5b shows the initial topology of the quadrotor (the functional dependence graph in the language of [4]), which is extracted from the relationships in Fig. 5a. Note that this graph is not chordal [4], which means that the designer will need to choose additional variable couplings for the system to decompose.

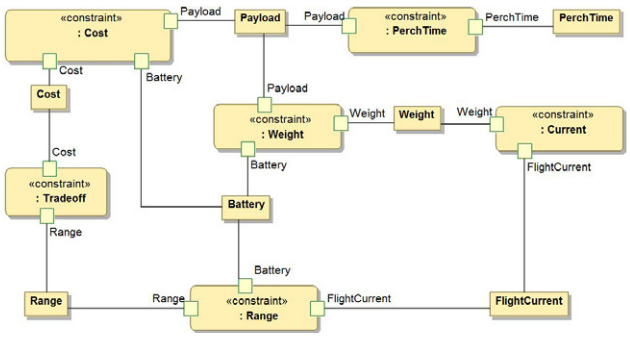


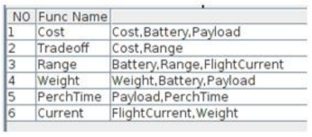
Fig. 4: Parametric diagram for high level tradeoffs of a quadrotor. Consider the constraints shown in this diagram. The Tradeoff constraint reflects the fact that we are interested in the relationship between cost and range of the quadrotor. As indicated by the Cost constraint, the cost value is determined entirely, in this model, by the 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.

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 Fig. 5c. One more fillin is 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. Fig. 5d shows the result. This system is chordal and has a tree decomposition. The tree structure 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 Fig. 5e as the tree decomposition of the system using these hints from the designer.

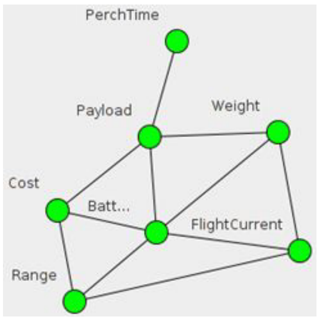
The last step in the analysis described in [4] is to map the original constraints and functions back to the resulting join tree. The most natural language for expressing this is a block diagram, as shown in Fig. 6. The associations between blocks are labeled according to the shared variables. There is 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 that the assignment of the constraints back to the structure is unique in this case.

To analyze the system shown in Fig. 6, 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, 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. Fig. 7 depicts the general strategy of evaluation. Using the decomposition of Fig. 6 reduces the complexity of analyzing the system from D^7 down to $3D^4 + D^2$. This is a

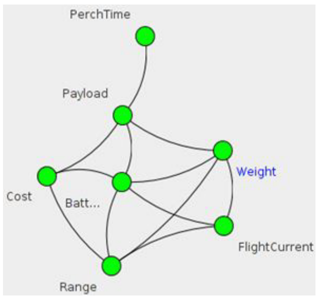


NO	Func Name	
1	Cost	Cost,Battery,Payload
2	Tradeoff	Cost,Range
3	Range	Battery,Range,FlightCurrent
4	Weight	Weight,Battery,Payload
5	PerchTime	Payload,PerchTime
6	Current	FlightCurrent,Weight

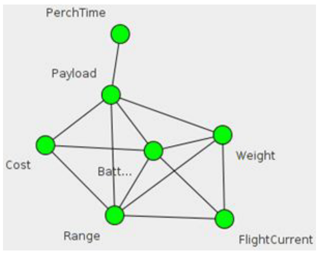
(a) Input to the tool



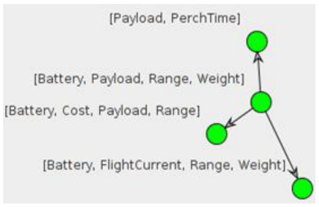
(b) Functional dependence graph (initial)



(c) Functional dependence graph (with additional fillins)



(d) Chordal transformation


[Payload, PerchTime]
[Battery, Payload, Range, Weight]
[Battery, Cost, Payload, Range]
[Battery, FlightCurrent, Range, Weight]

(e) Join tree

Fig. 5: Quadrotor example

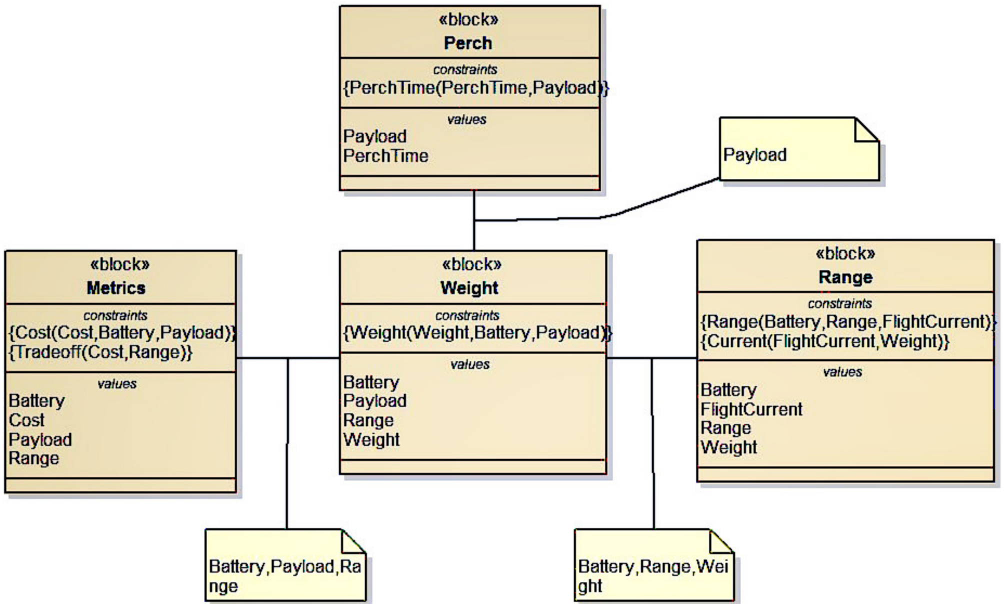


Fig. 6: The completed Block Diagram of the tree decomposition of the quadrotor

significant reduction. Suppose, for example, we use a grid of 20 points. $20^7 = 1.28*10^9$ while $3*20^4+20^2 = 480,400$, which is orders of magnitude fewer samples.

Now we summarize how systems engineers can use our tool to improve their designs. Firstly, a system engineer transforms the constraints (i.e., functions) in the SysML Parametric Diagrams and inputs them into our tool as in Fig. 5a. If the generated functional dependence graph is not chordal (e.g., Fig. 5b), the engineer needs to interact with our tool to help it transform the initial functional dependence graph into a chordal graph, which is essential for our algorithm. The basic operation is to add more fillins. However, the transformation is not unique. Different set of added fillins can result in different expected complexity of system analysis, which is dominated by the size of the maximum clique in the chordal graph. The engineer can try different options, and if the current selection induces a large complexity, the engineer can roll back the current graph to its previous status and continue investigation with other options. Therefore, our tool can expose a sequence of design choices to systems engineers to provide instant feedback about the influence of a design decision on the complexity of system analysis.

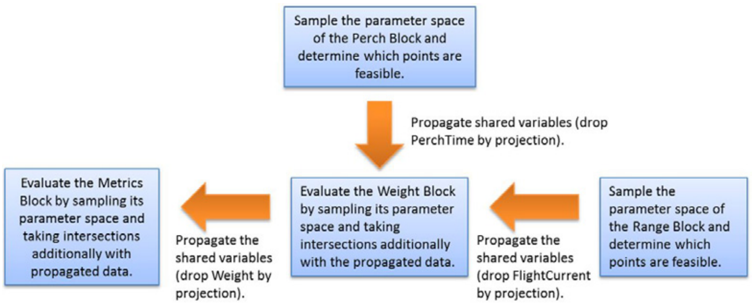

Sample the parameter space of the Perch Block and determine which points are feasible.
Propagate shared variables (drop PerchTime by projection).
Evaluate the Metrics Block by sampling its parameter space and taking intersections additionally with propagated data.
Propagate the shared variables (drop Weight by projection).
Evaluate the Weight Block by sampling its parameter space and taking intersections additionally with the propagated data.
Propagate the shared variables (drop FlightCurrent by projection).
Sample the parameter space of the Range Block and determine which points are feasible.

Fig. 7: Summary propagation applied to the block diagram of Fig. 6. 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

With a chordal graph, our tool will generate a join tree, based on which the engineer can create a SysML Block Diagram by assigning constraints to blocks. This Block Diagram is essentially a factor join tree, in which blocks are the factor nodes, and the intersection of parameters in two blocks is the variable node between them, as shown in Fig. 6. Based on this Block Diagram, the engineer can revise the original SysML Parametric Diagrams accordingly, such that constrains and parameters in the same block can be grouped together locally. Then the engineer can analyze the system using summary propagation as shown in Fig. 7. Therefore, our tool can provide guidance for systems engineers to improve their designs and analyze their systems.

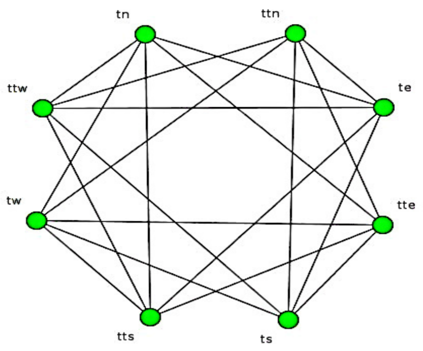
3.4. Other Examples

Traffic Intersection. Fig. 8a shows the interference graph for a traffic intersection. 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. Fig. 8b shows one of the possible join trees generated by the tool. Since Fig. 8a is not chordal, this is not a unique decomposition. Note that the interface or shared variables are *te*, *tte*, *ttw* and *tw*. This is essentially a Boolean satisfaction problem where we are searching for all the satisfying instances.

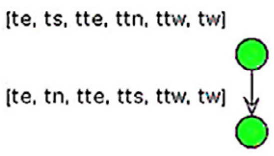
It is not immediately clear how to analyze the system using the join tree given in Fig. 8b since the variables are highly coupled. We take the nodes of Fig. 8a and rearrange them so that the interface variables *te*, *tte*, *ttw* and *tw* are in the middle separating the rest of the variables. Fig. 8c shows the result of this manipulation. This graph, derived from the join tree of Fig. 8b, 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. Fig. 9a shows a Bayesian network depicting a disease diagnosis inference. Fig. 9b shows its corresponding loop-free join tree that is suitable for inference.

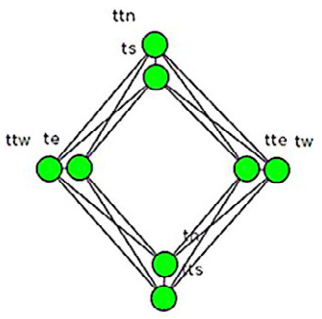
Mind Mapping. One problem that occurs when writing a paper is taking a graph that represents the ideas in the paper and linearizing it into an outline. This tool can help in doing this. Fig. 10 shows the graph structure of some concepts used in a different paper.



(a) Interference graph

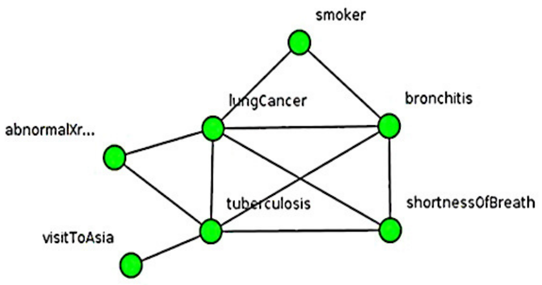


(b) Join tree

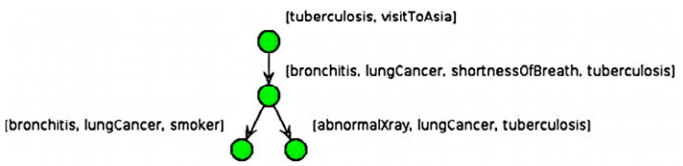


(c) Structure of traffic interference dependence graph

Fig. 8: Example of traffic intersection



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


[tuberculosis, visitToAsia]
[bronchitis, lungCancer, shortnessOfBreath, tuberculosis]
[bronchitis, lungCancer, smoker]
[abnormalXray, lungCancer, tuberculosis]

(b) A graph depicting the join tree of the Bayesian network from Fig. 9a. The join tree is loop free so summary propagation is an exact inference algorithm

Fig. 9: Example of Bayesian network

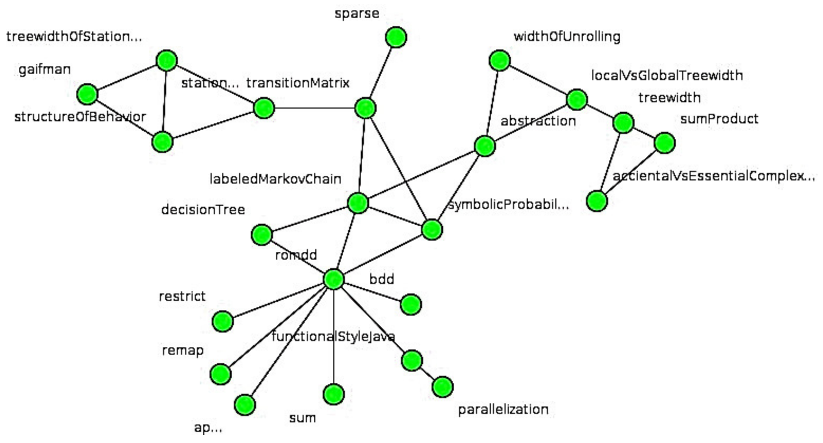


Fig. 10: A map of concepts in a paper. This graph is coincidentally chordal

Fig. 11 depicts the tree representation of the graph in Fig. 10. The tool helps convert the graph representation into a tree representation of the information. The tree can then be conveniently linearized into an outline.

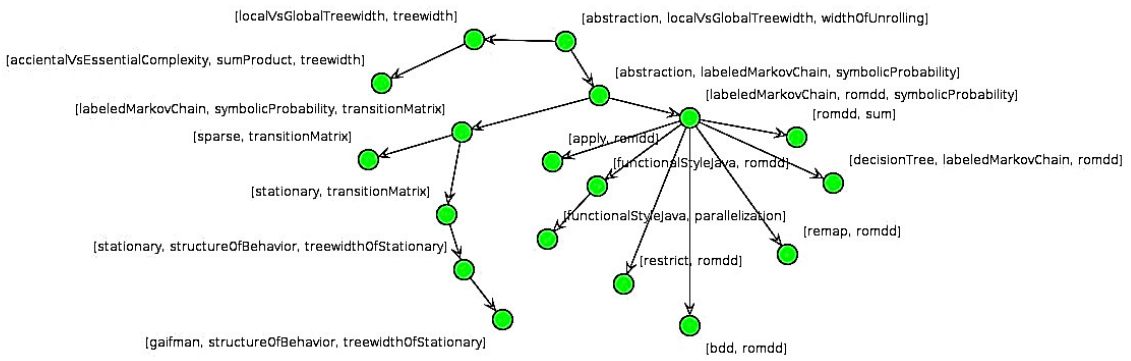

[localVsGlobalTreewidth, treewidth]
[abstraction, localVsGlobalTreewidth, widthOfUnrolling]
[accientalVsEssentialComplexity, sumProduct, treewidth]
[abstraction, labeledMarkovChain, symbolicProbability]
[labeledMarkovChain, romdd, symbolicProbability]
[labeledMarkovChain, symbolicProbability, transitionMatrix]
[romdd, sum]
[sparse, transitionMatrix]
[apply, romdd]
[functionalStyleJava, romdd]
[decisionTree, labeledMarkovChain, romdd]
[stationary, transitionMatrix]
[functionalStyleJava, parallelization]
[remap, romdd]
[stationary, structureOfBehavior, treewidthOfStationary]
[restrict, romdd]
[gaifman, structureOfBehavior, treewidthOfStationary]
[bdd, romdd]

Fig. 11: A tree view of the contents of Fig. 10. 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

4. Discussion

One interesting property of the technique is how counterintuitive these join trees are from the perspective of creating block diagrams. However, looking at Fig. 5d 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 connects 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 [12], we show how the same technique of composition and projection can help in the formal analysis of dynamic Bayesian networks.

5. Future Work

The current tool only implements the basic join tree algorithm that can be found in [13]. 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. [14] 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 [14]. 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 [4], 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.

6. Conclusion

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.

Acknowledgment

Research supported in part by the NSF under grant CNS-1035655, by the NIST under contract 70NANB11H148 and by a grant from the Lockheed Martin Corporation.

References

[1] J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference, 1988.
[2] S. Arnborg, A. Proskurowski, Linear time algorithms for np-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1) (1989) 11–24.
[3] H. Bodlaender, Dynamic programming on graphs with bounded treewidth, Automata, Languages and Programming (1988) 105–118.
[4] S. Yang, J. S. Baras, Factor join trees for systems exploration, in: International Conference on Software and Systems Engineering and their Applications, 2011, pp. 1–10.
[5] M. D. Guenov, Complexity and cost effectiveness measures for systems design, in: Manufacturing Complexity Network Conference, 2002, pp. 1–13.
[6] S. C. Y. Lu, N.-P. Suh, Complexity in design of technical systems, CIRP Annals - Manufacturing Technology 58 (1) (2009) 157 – 160.
[7] E. Clarke, Compositional model checking, in: Proceedings of Fourth Annual Symposium on Logic in Computer Science, 1989, pp. 353 – 362.
[8] K. L. McMillan, Symbolic model checking: an approach to the state explosion problem, Ph.D. thesis, uMI Order No. GAX92-24209 (1992).
[9] A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, Symbolic model checking without bdds, in: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems, TACAS '99, 1999, pp. 193–207.
[10] E. Clarke, Computer-aided verification, IEEE Spectrum 33 (6) (1996) 61 – 67.
[11] B. Wang, J. Baras, Performance analysis of time-critical peer-to-peer communications in ieee 802.15.4 networks, in: IEEE ICC, 2011, pp. 1–6.
[12] S. Yang, Y. Zhou, J. S. Baras, Compositional analysis of dynamic bayesian networks and applications to cps, in: Conference on Systems Engineering Research (CSER), 2013, pp. 1–10.
[13] F. V. Jensen, T. D. Nielsen, Bayesian Networks and Decision Graphs, 2nd Edition, 2007.
[14] L. Ibarra, The clique-separator graph for chordal graphs, Discrete Appl. Math. 157 (8) (2009) 1737–1749.

Appendix A. Theory Background

A *system* is defined as the tuple $\mathcal{P} = \langle \mathcal{L}, \mathcal{P}_1(\mathcal{X}_1), \dots, \mathcal{P}_\mathcal{M}(\mathcal{X}_\mathcal{M})\rangle$, with $\mathcal{L} = \{\Sigma_1, \dots, \Sigma_\mathcal{N}\}$ and $\mathcal{X}_i \subseteq \mathcal{L}$ for $i = 1, \dots, \mathcal{M}$. Each $\Sigma_i \in \mathcal{L}$ is a set corresponding to the domain of a system variable x_i. Each $\mathcal{P}_i$ $(i = 1, \dots, \mathcal{M})$ is a general component that influences the variables with domains $\mathcal{X}_i$. A *functional dependence graph* of a system $\mathcal{P}$ is defined as the graph $G = \langle \mathcal{L}, E\rangle$ with $E = \{(x, y) \mid \exists i \in [1, \mathcal{M}]\ s.t.\ (x, y \in \mathcal{X}_i) \wedge (x \neq y)\}$. Every parameter set defined by $\mathcal{X}_i$ induces a clique of mutually connected nodes in the flattened graph G. The *elimination* of a node $\Sigma \in \mathcal{L}$ from the graph G, denoted by $\bigoplus_\Sigma G$, is defined as the graph $G' = \langle \mathcal{L} \setminus \{\Sigma\}, E'\rangle$, where E' is defined as $(E \setminus \{(x, y) \mid \Sigma \in \{x, y\}\}) \bigcup F$. Here, F is the set of links in the clique induced by the set of neighbors $N(\Sigma)$ of Σ. The *sequence of graphs induced by an elimination ordering* $\mathcal{L}_p$, which is denoted by $\langle G_1(\mathcal{L}_p), \dots, G_{\mathcal{N}+1}(\mathcal{L}_p)\rangle$, is defined as $G_1(\mathcal{L}_p) = G$ and $G_{k+1}(\mathcal{L}_p) = \bigoplus_{\Sigma_{i_k}} G_k(\mathcal{L}_p)$ for $k = 1, \dots, \mathcal{N}$. The *sequence of cliques induced by an elimination ordering* $\mathcal{L}_p$, which is denoted by $\langle C_1(\mathcal{L}_p), \dots, C_\mathcal{N}(\mathcal{L}_p)\rangle$, is defined as $C_k = N_{G_k(\mathcal{L}_p)}(\Sigma_{i_k}) \bigcup \{\Sigma_{i_k}\}$ for $k = 1, \dots, \mathcal{N}$. Here, $N_{G_k(\mathcal{L}_p)}(\Sigma_{i_k})$ is the set of neighbors of Σ_{i_k} in $G_k(\mathcal{L}_p)$ from the sequence of graphs induced by $\mathcal{L}_p$.

The *width* of graph G with respect to ordering $\mathcal{L}_p$ is defined as the maximum size of the cliques in the sequence of induced cliques minus 1, i.e., $W_G(\mathcal{L}_p) = \max_k |C_k(\mathcal{L}_p)| - 1$. The extra minus 1 ensures that the treewidth of a tree is 1. The *treewidth* W of a system $\mathcal{P}$ is defined as:

$$W = \min_{\mathcal{L}_p} W_G(\mathcal{L}_p) = \min_{\mathcal{L}_p} \max_k |C_k(\mathcal{L}_p)| - 1$$

The value of W gives the minimal tree decomposition of the system.

A node is *simplical* if all of its neighbors are mutually connected. We have the following theorem:

Theorem 1. *Let $\mathcal{L}_p$ be an elimination order where Σ_{i_k} and Σ_{i_k+1} are both simplical in $G_k(\mathcal{L}_p)$, and $\mathcal{L}'_p$ be another elimination order by swapping Σ_{i_k} and Σ_{i_k+1} in $\mathcal{L}_p$. Then $W_G(\mathcal{L}'_p) = W_G(\mathcal{L}_p)$.*

Theorem 1 states that eliminations of simplical nodes are commutative with respect to the treewidth of the resulting graphs. Therefore, simplical nodes can be eliminated in any order without impacting the width of the graph. Our heuristic algorithm to find the treewidth of a graph 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.