Symbolic Bucket Elimination for Piecewise Continuous Constrained Optimization

Zhijiang Ye, Buser Say, and Scott Sanner

University of Toronto
tonyyezj@gmail.com,{bsay,ssanner}@mie.utoronto.ca

Abstract. Bucket elimination and its approximation extensions have proved to be effective techniques for discrete optimization. This paper addresses the extension of bucket elimination to continuous constrained optimization by leveraging the recent innovation of the extended algebraic decision diagram (XADD). XADDs support symbolic arithmetic and optimization operations on piecewise linear or univariate quadratic functions that permit the solution of continuous constrained optimization problems with a symbolic form of bucket elimination. The proposed framework is an efficient alternative for solving optimization problems with low tree-width constraint graphs *without* using a big-M formulation for piecewise, indicator, or conditional constraints. We apply this framework to difficult constrained optimization problems including XOR's of linear constraints and temporal constraint satisfaction problems with "repulsive" preferences, and show that this new approach significantly outperforms Gurobi. Our framework also enables symbolic parametric optimization whose closed-form solution cannot be computed with tools like Gurobi, where we demonstrate a final novel application to parametric optimization of learned Relu-based deep neural networks.

Keywords: bucket elimination, decision diagram, constrained optimization, symbolic dynamic programming

1 Introduction

Bucket elimination [2, 7] is a generalized dynamic programming framework that has been widely applied to probabilistic reasoning problems on graphical models [8], including cost networks, constraint satisfaction [6], and propositional satisfiability [5]. The application of this framework to combinatorial optimization problems has been shown to be highly competitive against alternative techniques [16, 14]. In this paper, we propose symbolic bucket elimination (SBE) as a novel method of solving mixed discrete and continuous constrained optimization problems (i.e., covering MILPs and a subclass of MIQPs). SBE critically leverages recent innovations in the extended algebraic decision diagram (XADD) that enable the exact representation and manipulation of piecewise linear and univariate quadratic functions [17]. We show that SBE can outperform Gurobi on low tree-width constrained optimization problems and that SBE can also perform symbolic *parameteric* optimization of learned Relu-based deep neural networks [15] — something tools like Gurobi cannot do exactly in closed-form.

2 Background

2.1 Case Representation and Operations

The case statements constitute the foundational symbolic mathematical representation that is used throughout this paper and are presented below.

Case Statement The *case* statement takes the following form:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases}$$

where ϕ_i is a logical formula over domain $(\boldsymbol{b}, \boldsymbol{x})$ with discrete[1] $\boldsymbol{b} \in \mathbb{B}^m$ and continuous variables $\boldsymbol{x} \in \mathbb{R}^n$, and is defined by arbitrary logical combinations $(\wedge, \vee, \neg)$ of (1) boolean variables in $\boldsymbol{b}$ and (2) linear inequality relations $(\geq, >, \leq, <)$ over continuous variables in $\boldsymbol{x}$. Each ϕ_i is disjoint from other ϕ_j $(j \neq i)$ and exhaustively covers the entire domain such that f is well defined. Each f_i is a linear or univariate quadratic function (LUQF) of $\boldsymbol{x}$, e.g. $f_1 = x_1 + 3x_2$ or $f_2 = 5x_3^2 - 2x_3 + 1$. Only one variable can be quadratic in a case statement and wherever it occurs it must be univariate, hence given the previous examples $f_3 = x_1 + 2x_3$ would be disallowed with f_1 and f_2 in the same case statement.

Binary Operations For binary operations, the cross-product of logical partitions of each case is taken. For example, the "cross-sum" $\oplus$ is defined as:

$$\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} \oplus \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{cases}$$

Note that the case representation is closed under general conditions for $\oplus$.

Case Maximization Maximization of two case statements is a piecewise operator that can be defined easily (e.g., consider the maximum of two hyperplanes):

$$\text{casemax}\left(\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases}, \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge f_1 > g_1 : & f_1 \\ \phi_1 \wedge \psi_1 \wedge f_1 \leq g_1 : & g_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 > g_2 : & f_1 \\ \phi_1 \wedge \psi_2 \wedge f_1 \leq g_2 : & g_2 \\ \vdots & \vdots \end{cases}$$

[1] For simplicity of exposition, we presume that non-binary discrete variables of cardinality k are encoded in binary with $\lceil \log_2(k) \rceil$ boolean variables.

The casemin operator is defined analogously. We remark that the case representation is closed for casemax and casemin for linear ϕ_i, ψ_j, f_i, and g_j. These operators are not necessarily closed for LUQF operands since the newly introduced constraints $f_i > g_j$ may become non-LUQF. However, we can often eliminate univariate quadratic variables before applying a casemax or casemin.

Case Substitution The case substitution operator defined as $\sigma = (y/g)$, triggers the substitution of a variable y with a case statement g. Similar to the $\oplus$ operation, $f\sigma$ results in a case with conditions as cross-products of case conditions between f and g, and value expressions in f with variable y replaced by the value corresponding to the case condition in g. As an illustrative example:

$$f\sigma = \begin{cases} x < y \wedge z \leq 0: & x + v \\ x < y \wedge z > 0: & x - 2w \\ y < z \wedge z \leq 0: & v + z \\ y < z \wedge z > 0: & -2w + z \end{cases}, \quad f = \begin{cases} x < y: & x + y \\ y < z: & y + z \end{cases}, \quad g = \begin{cases} z \leq 0: & v \\ z > 0: & -2w \end{cases}$$

Maximization/Minimization Over a Variable In symbolic optimization we will want to maximize over both boolean and continuous variables. For a boolean max over b_i, we simply take the casemax over both instantiations $\{0, 1\}$ of b_i: [2] $f(\boldsymbol{b}_{\setminus i}, \boldsymbol{x}) = \max_{b_i} g(\boldsymbol{b}, \boldsymbol{x}) = \text{casemax}(g(b_i = 0, \boldsymbol{b}_{\setminus i}, \boldsymbol{x}),\ g(b_i = 1, \boldsymbol{b}_{\setminus i}, \boldsymbol{x}))$. Symbolic maximization over a continuous variable x_i is a much more involved operation written as $f(\boldsymbol{b}, \boldsymbol{x}_{\setminus i}) = \max_{x_i}\ g(\boldsymbol{b}, \boldsymbol{x})$ and discussed in detail in [20]. This operation *is* closed-form for LUQF $g(\boldsymbol{b}, \boldsymbol{x})$ and results in a purely symbolic case statement for $f(\boldsymbol{b}, \boldsymbol{x}_{\setminus i})$. Minimization operators are defined analogously.

2.2 Extended Algebraic Decision Diagrams (XADDs)

Due to cross-product operations, a data structure such as decision diagrams are required to maintain a tractable case representation. Bryant [4] introduced the reduced ordered binary decision diagram (BDD) representing boolean functions; algebraic decision diagrams (ADD) [1] extended BDDs to non-boolean functions. The extended algebraic decision diagram (XADD) [18] shown in Figure 1 extends the ADD to allow continuous variables with inequalities for decsions and LUQF expressions for leaves. As for ADDs, XADDs have a fixed order of decisions from root to leaf. The standard ADD operations to build a canonical ADD (Reduce) and to perform a binary operation on two ADDs (Apply) also apply for XADDs. The XADD can be exponentially smaller than the case representation (each path from root to leaf is a case partition) and all previous case operations can be implemented to exploit the DAG stucture of XADDs [20].

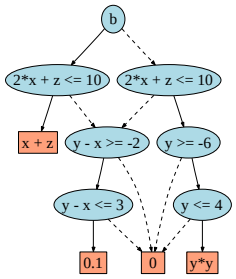


Fig. 1. Example XADD. The true branch is solid, the false branch is dashed.

[2] We use $\boldsymbol{b}_{\setminus i}$ to denote the set $\boldsymbol{b}$ with the variable b_i excluded. Similarly $\boldsymbol{x}_{\setminus i}$ denotes exclusion of x_i from $\boldsymbol{x}$

3 Symbolic Bucket Elimination

In this section, we introduce our novel framework: symbolic bucket elimination for continuous constrained optimization. Problems can be specified as follows:

$$\max_{\boldsymbol{b},\boldsymbol{x}} \sum_{i=1}^{n} R_i(\boldsymbol{b},\boldsymbol{x}) \text{ subject to } C_j \; \forall j \in \{1,\cdots,k\} \tag{1}$$

In our case, the R_i can be LUQF expressions and the C_j are linear constraints. We translate problems of this form into their symbolic equivalent:

$$\max_{\boldsymbol{b},\boldsymbol{x}} \bigoplus_{i=1}^{m} F_i(\boldsymbol{b},\boldsymbol{x}), \text{where } F_i = \begin{cases} C_i: & 0 \\ \neg C_i: & -\infty \end{cases}, \forall i \in \{1,\ldots,k\}. \tag{2}$$

where $m = k+n$ and for each linear constraint C_j, there is a corresponding linear case statement $F_i(\boldsymbol{b},\boldsymbol{x})$. A maximum value of $-\infty$ for the problem would indicate that the problem is overconstrained and infeasible. We also note that while a case representation would require a big-M formulation to handle piecewise, indicator, and conditional constraints possible in cases, this framework represents all of these logical constraints natively in the symbolic case form and thus as XADDs.

3.1 Symbolic Bucket Elimination Algorithm (SBE)

We can solve the general optimization problem of (2) by using a fully symbolic variant of bucket elimination [8] leveraging the case (XADD) representation and its efficient operations. In bucket elimination, each function F_i is placed into ordered, variable-specific buckets. Variable ordering is determined by heuristics that aim to minimize the induced tree-width of the underlying graph [8], with the restriction that variables appearing on the left hand side of equality constraints will be ordered such that their representative buckets will be eliminated first. The rule for bucket assignment is to identify the variable in each function that appears the latest in the ordering, and place the function in the bucket of the respective variable. The buckets are then eliminated sequentially in the forward step. In backtracking, optimal assignments are obtained with an arg max [20] on the summed function for each bucket. The SBE algorithm is presented in Algorithm 1. If the objective is minimization, (arg)max replaces (arg)min.

Computational Complexity For bucket elimination over discrete domains, complexity is bounded by an exponential function of the tree-width of the constraint graph [8]. When we extend bucket elimination to continuous domains using XADD, the complexity is not explicitly tree-width dependent. While a constraint with many decision variables may be represented compactly as a piecewise expression, one can generally only upper bound the number of pieces needed in a case statement as an exponential function of the number of primitive binary operations ($\oplus$, casemax) used by bucket elimination. Nevertheless, the XADD does maintain compact representations much smaller than the worst-case upper bound and proves to be particularly advantageous when the underlying constraint graph has low tree-width, as we show in the experimental results section.

Algorithm 1 SYMBOLIC BUCKET-ELIMINATION (SBE)

Input: Input XADD functions $\{F_1, \ldots, F_m\}$, a variable ordering d
Output: The optimal objective value and variable assignments

Initialization: bucket assignment
Assign each input function to an ordered set of buckets: $bucket_1, \ldots, bucket_n$. Let $f_{j,p}$ denote a function residing in $bucket_p$ that is either an input function, or a resultant function from a bucket already eliminated
Forward Elimination: eliminate each bucket sequentially
for $p \leftarrow n$ down to 1 in ordering d **do**
    $g_p \leftarrow \bigoplus_j f_{j,p}$
    if p is on left hand side of equality constraint: $p = cons$ **then**
        $h_p \leftarrow g_p\sigma$, where $\sigma = (p/cons)$ // substitute variable for constraint
    else $h_p \leftarrow \max_{x_p} g_p$
    if $p > 1$ **then** assign h_p to a bucket according to rule ($bucket_1$ if constant)
Backtracking: recover optimal variable assignments, $\boldsymbol{x}^*$
for $p \leftarrow 1$ up to n in ordering d **do**
    $x_p^* \leftarrow \arg\max_{x_p} g_p(x_1 = x_1^*, \ldots, x_{p-1} = x_{p-1}^*)$, $\boldsymbol{x}^* \leftarrow \boldsymbol{x}^* \cup \{x_p^*\}$
return h_1, $\boldsymbol{x}^*$

4 Experimental Results

In this section, we demonstrate the computational efficiency and the expressiveness of symbolic bucket elimination framework on three distinct problems. First, we present two problems in which the symbolic bucket elimination framework outperformed the state of art solver AMPL-Gurobi (7.5.0) with default settings on a 2.20GHz processor [10]. Second, we present a novel application of symbolic parametric optimization to Relu-based deep neural networks.

4.1 Problems with XOR Conditional Constraints

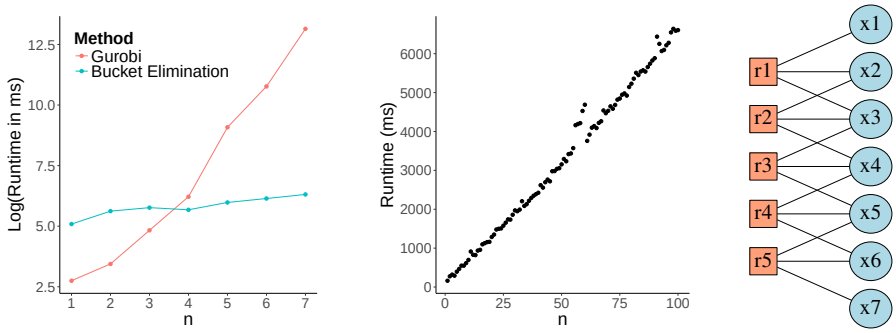
Following is a synthetic problem involving constraints combined with XOR ($\veebar$):

$$\max \sum_{i=1}^{n} r_i$$

$$\text{where } r_i = \text{if } (x_i \geq x_{i+1} \veebar x_{i+1} \geq x_{i+2}) \text{ then } \max(x_{i+1}, x_{i+2}) - \min(x_{i+1}, x_{i+2})$$
$$\text{else } \min(x_i, x_{i+1}) - \max(x_i, x_{i+1}),$$
$$\text{subject to } -10 \leq x_i \leq 10, \forall i \in \{1, \ldots, n+2\}, -20 \leq r_j \leq 20, \forall j \in \{1, \ldots, n\}$$

In this problem, the reward term in the objective r_i is determined by an XOR conditional expression involving the respective decision variables x_i, x_{i+1}, x_{i+2}.

This problem structure is particularly advantageous for the proposed framework due to the small size of the decision diagram for each constraint term, as well as the sparsity of the constraints. This is illustrated through evaluations of its runtime performance, with comparisons to that of Gurobi in Figure 2.



(a) (b) (c)

Fig. 2. (a) Comparison of log runtime of SBE vs. Gurobi, with SBE outperforming Gurobi for $n > 4$. (b) Non-log runtime for $1 \leq n \leq 100$ showing SBE is linear in n. (c) Constraint graph for $n = 5$, showing the low tree-width nature of this problem.

Symbolic bucket elimination outperforms Gurobi, even at a very small n. The performance gap becomes significant as n increases — while the runtime for Gurobi scales *exponentially* in n, the bucket elimination framework scales *linearly* in n as evidenced in the non-log plot Figure 2(b).

4.2 Temporal Constraint Satisfaction with Preferences

Temporal constraint problems with preferences deal with finding optimal assignments to time events based on preferences [12]. The objective is to optimize the total preference value, subject to a set of constraints such as ordering of certain time events, or time delays between events. This class of problems is a combination of Temporal Constraint Satisfaction Problems [9] with soft constraints [3]. The problem considered is:

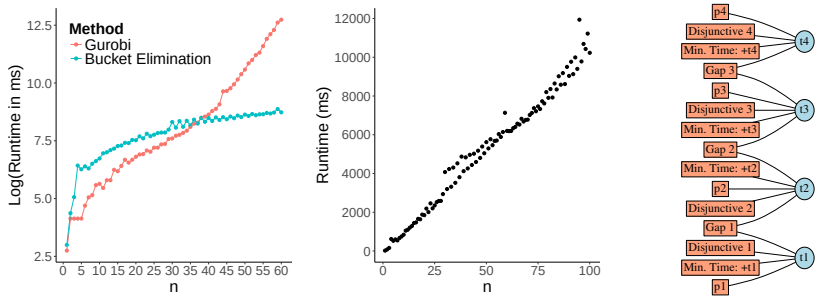
$$\min \sum_{i=1}^{n} t_i + p_i$$

$$\text{where } p_i = \text{if } (t_i \leq 10(i+1)) \text{ then } {t_i}^2 \text{ else } (t_i - 10n)^2, \forall i \in \{1, \ldots, n\}$$

$$\text{subject to } 10i \leq t_i \leq 10(i+1) \vee 10(i+2) \leq t_i \leq 10(i+3), \forall i \in \{1, \ldots, n\}$$

$$t_i + 10 \leq t_{i+1}, \forall i \in \{1, \ldots, n-1\}, \{t_i, p_i\} \in \mathbb{R}, \forall i \in \{1, \ldots, n\}$$

The definition of p_i is analogous to the mid-value preference constraint presented in [12]. The preference value p_i is dictated by whether the condition $t_i \leq 10(i+1)$ is true. If so, then it is preferred for time event t_i to occur as close to time 0 as possible, otherwise t_i should occur close to time $10n$ (i.e., the preferences are "repulsive" and prefer opposite ends of the timeline). The objective is to minimize the sum of time events t_i and preference values p_i. The first constraint is a disjunctive type constraint on time event t_i. The second constraint imposes a gap requirement between time events. We note that although quadratic terms



(a) (b) (c)

Fig. 3. (a) Comparison of log runtime of SBE vs. Gurobi, with SBE outperforming Gurobi for $n > 40$. (b) Non-log runtime for $1 \leq n \leq 100$ showing SBE is linear in n. (c) Constraint graph for $n = 4$, demonstrating the low tree-width structure.

appear in the leaves of the representative constraint XADD, decisions will remain linear throughout SBE as there are no explicit discrete variables to maximize and therefore no casemax operations to promote quadratic terms into decisions. The runtime evaluations and problem structure are visualized in Figure 3. As in the previous example, the runtime for Gurobi scales *exponentially* in problem size while SBE scales *linearly*. SBE outperforms Gurobi for $n > 40$.

4.3 Symbolic Parametric Optimization of Deep Neural Networks

We demonstrate a novel application of SBE to perform symbolic parametric optimization of deep neural networks with rectified linear units (Relu) that are piecewise linear. Previous work has shown Relu-based deep neural networks can be compiled into linear constraint optimization programs and solved non-parametrically with applications in automated planning [19] and verification [11]. In this section, we show promising results for symbolic parametric optimization on the learned output units, $y_j(\boldsymbol{x})$ of a Relu-based deep network trained on $h(\boldsymbol{x})$ by first compiling the constraints and then maximizing $y_j(\boldsymbol{x})$ w.r.t. a subset of the input variables $\boldsymbol{x}_s$ using SBE. This results in new *symbolic* piecewise functions that represent the *maximal* deep network output values that can be achieved for the best $\boldsymbol{x}_s$ as a function of the remaining inputs. Such symbolic parametric (partial) optimization is not possible with Gurobi. The Relu-based deep network is represented by the following objective, piecewise linear case statements and constraints implementing the connections and Relu functions:

$$\max_{\boldsymbol{x_s} \subset \{x_1,\ldots,x_n\}} y_j, \forall j \in \{1,\ldots,m\}$$

$$y_j = \sum_{i \in \{1,\ldots,p\}} w_{i,j,l} r_{i,l} + b_{j,l+1}, \forall j \in \{1,\ldots,m\}$$

$$r_{j,k} = \max(\sum_{i \in \{1,\ldots,p\}} (w_{i,j,k-1} r_{i,k-1}) + b_{j,k}, 0), \forall j \in \{1,\ldots,p\}, k \in \{2,\ldots,l\}$$

$$r_{j,1} = \max(\sum_{i \in \{1,\ldots,n\}} (w_{i,j,0} x_i) + b_{j,1}, 0), \forall j \in \{1,\ldots,p\}$$

$$-10 \leq x_i \leq 10, \forall i \in \{1,\ldots,n\}, y_j \in \mathbb{R}, \forall j \in \{1,\ldots,m\}$$

$$0 \leq r_{i,k} \in \mathbb{R}, \forall i \in \{1,\ldots,p\}, k \in \{1,\ldots,l\}$$

where parameters n, m, p and l denote the number of input units, number of output units, width (units in a hidden layer) and depth (hidden layers) of the network, $w_{i,j,k}$ denotes the weight between unit i at layer k and unit j, and $b_{j,k}$ denotes the bias of unit j at layer k.

In Figure 4, we show an example neural network structure and the runtime results of using SBE to parametrically optimize a network trained to learn $h(\boldsymbol{x}) = x_1^2 + x_2^2$ with $n = 2$, $m = 1$ for various width p and depth l. Runtimes are heavily width-dependent since tree-width grows with the width of a deep net, but not depth. The SBE eliminates the nodes in the hidden layers in a backward manner until it reaches the input layer, where the variable x_1 is maximized out. We note that for networks with more than one output, it is possible with SBE to parametrically optimize on different sets of $\boldsymbol{x}_s$ for the different outputs. Other types of activation functions (i.e., linear or step) are also possible, as long as each unit can be represented as piecewise functions. SBE applied to deep nets as done here has potential applications in planning and verification: *i.e.*, what is achievable as a function of an input?

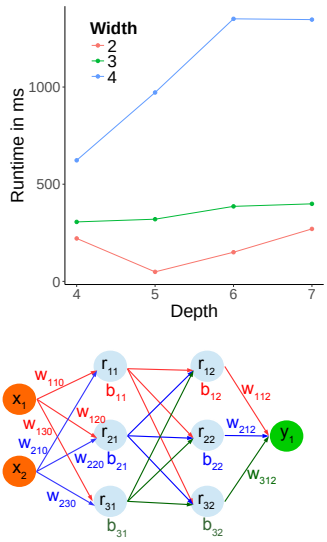


Fig. 4. Top: Runtime for SBE for network widths 2-4, depths 4-7. Bottom: Deep neural network structure with 2 input units, 1 output unit, width of 3, hidden layer depth of 2 ($n = 2, m = 1, p = 3, l = 2$).

5 Conclusion and Future Work

We introduced a novel symbolic bucket elimination (SBE) framework for representing and solving constrained optimization problems symbolically (and even parametrically), that can *exponentially* outperform Gurobi when the underlying constraint graph has low tree-width. In terms of future work, we remark that previous investigations in the discrete domain using mini-buckets [7] and heuristic search have demonstrated excellent improvement over exact bucket elimination [16, 13]. Hence, a promising direction for future work is mini-bucket extensions of SBE to allow it to scale to higher tree-width constrained optimization problems, vastly extending the scope of applicability of SBE.

References

1. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. In: Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design. pp. 188–191. ICCAD '93, IEEE Computer Society Press, Los Alamitos, CA, USA (1993), http://dl.acm.org/citation.cfm?id=259794.259826
2. Bertele, U., Brioschi, F.: Nonserial Dynamic Programming. Academic Press, Inc., Orlando, FL, USA (1972)
3. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM 44(2), 201–236 (Mar 1997), http://doi.acm.org/10.1145/256303.256306
4. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. 35(8), 677–691 (Aug 1986), http://dx.doi.org/10.1109/TC.1986.1676819
5. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM 7(3), 201–215 (Jul 1960), http://doi.acm.org/10.1145/321033.321034
6. Dechter, R., Pearl, J.: Search in artificial intelligence. chap. Network-based Heuristics for Constraint Satisfaction Problems, pp. 370–425. Springer-Verlag, London, UK, UK (1988), http://dl.acm.org/citation.cfm?id=60727.60738
7. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence 113(1), 41 – 85 (1999), http://www.sciencedirect.com/science/article/pii/S0004370299000594
8. Dechter, R.: Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms. Morgan & Claypool Publishers (2013)
9. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. 49(1-3), 61–95 (May 1991), http://dx.doi.org/10.1016/0004-3702(91)90006-6
10. Gurobi Optimization, I.: Gurobi optimizer reference manual (2016), http://www.gurobi.com
11. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: An efficient smt solver for verifying deep neural networks. In: Proceedings of the Twenty-Ninth International Conference on Computer Aided Verification. CAV (2017)
12. Khatib, L., Morris, P., Morris, R., Rossi, F.: Temporal constraint reasoning with preferences. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1. pp. 322–327. IJCAI'01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001), http://dl.acm.org/citation.cfm?id=1642090.1642135
13. Larrosa, J., Dechter, R.: Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. Constraints 8, 303–326 (2003)
14. Larrosa, J., Morancho, E., Niso, D.: On the practical use of variable elimination in constraint optimization problems: still-life as a case study. Constraints 23, 421–440 (2005)
15. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML. pp. 807–814 (2010), http://www.icml2010.org/papers/432.pdf
16. Rollón, E., Larrosa, J.: Bucket elimination for multiobjective optimization problems. Journal of Heuristics 12(4), 307–328 (Sep 2006), https://doi.org/10.1007/s10732-006-6726-y
17. Sanner, S., Abbasnejad, E.: Symbolic variable elimination for discrete and continuous graphical models. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence. pp. 1954–1960. AAAI'12, AAAI Press (2012), http://dl.acm.org/citation.cfm?id=2900929.2901004

18. Sanner, S., Delgado, K., Barros, L.: Symbolic dynamic programming for discrete and continuous state mdps. In: UAI. pp. 643–652 (01 2011)
19. Say, B., Wu, G., Zhou, Y.Q., Sanner, S.: Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. pp. 750–756 (2017), https://doi.org/10.24963/ijcai.2017/104
20. Zamani, Z., Sanner, S., Fang, C.: Symbolic dynamic programming for continuous state and action mdps. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence. pp. 1839–1845. AAAI'12, AAAI Press (2012), http://dl.acm.org/citation.cfm?id=2900929.2900988