FastMapSVM/FastMapSVR for Predictive Tasks on CSPs, SAT, and Weighted CSPs

Kexin Zheng and Ang Li and T. K. Satish Kumar
Department of Computer Science, University of Southern California
Los Angeles, USA
{kexinzhe, ali355}@usc.edu, tkskwork@gmail.com

***Abstract*—Predictive tasks on Constraint Satisfaction Problems (CSPs), Satisfiability (SAT) problems, and Weighted CSPs (WCSPs) are usually NP-hard but can also be modeled as classification or regression problems suitable for Machine Learning (ML) algorithms. While most existing ML algorithms have had only limited success on such tasks, a newly developed ML framework, called FastMapSVM, has been shown to be successful for predicting CSP satisfiability. FastMapSVM leverages a distance function between pairs of CSP instances instead of trying to characterize individual CSP instances. In this paper, we advance FastMapSVM in various ways. For predicting the satisfiability of CSP and SAT instances, we design a distance function that utilizes maxflow computations and strong path-consistency (or a truncated version of it). For predicting the optimal cost of WCSP instances, we design a distance function that also utilizes maxflow computations and replaces the Support Vector Machine (SVM) component of FastMapSVM by a Support Vector-based Regression (SVR) component. We demonstrate the success of our FastMapSVM/FastMapSVR approach over competing state-of-the-art ML algorithms in all three domains: In the CSP domain, we demonstrate our success on several CSP benchmark suites; in the SAT domain, we demonstrate our success on hard 3-SAT instances drawn from the phase transition region; and in the WCSP domain, we demonstrate our success on a wide range of randomly generated WCSP instances.**

***Index Terms*—FastMapSVM, FastMapSVR, Constraint Satisfaction Problems, Satisfiability, Weighted Constraint Satisfaction Problems**

I. Introduction

Constraints constitute a very natural and general means for formulating regularities in the real world. A fundamental combinatorial structure used for reasoning with constraints is that of the Constraint Satisfaction Problem (CSP). The CSP formally models a set of variables, their corresponding domains, and a collection of constraints between subsets of the variables. Each constraint restricts the set of allowed combinations of values of the participating variables. A solution of a given CSP instance is an assignment of values to all the variables from their respective domains such that all the constraints are satisfied. Technologies for efficiently solving CSPs bear immediate and important implications on how fast we can solve computational problems that arise in several other areas of research, including computer vision, spatial and temporal reasoning, model-based diagnosis, planning and scheduling, and language understanding.

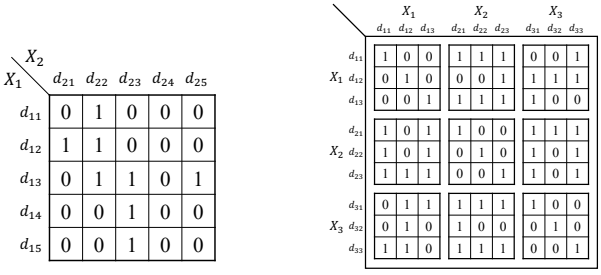
A special case of the CSP is the Satisfiability (SAT) problem, in which the variables are restricted to be Boolean. The SAT problem is foundational to Complexity Theory and AI. However, a more general case of the CSP is the Weighted CSP (WCSP), in which the constraints are generalized to weighted constraints [1]. Each weighted constraint specifies a weight (cost) associated with each combination of values of the participating variables. An optimal solution of a given WCSP instance is an assignment of values to all the variables from their respective domains such that the total sum of the weights is minimized. A Boolean WCSP is a WCSP in which all the variables are Boolean. WCSPs model a wide range of combinatorial optimization problems that arise in many real-world applications: combinatorial auctions, reasoning with user preferences [2], image restoration and panoramic image stitching [3], locating RNA motifs [4], and energy minimization on the Potts model or Markov random fields [5].

The SAT problem, the CSP, the WCSP, and the Boolean WCSP are all NP-hard to solve or solve optimally. Consequently, from a Machine Learning (ML) perspective, the tasks of predicting the satisfiability of the SAT problem and the CSP are NP-hard. Similarly, the tasks of predicting the optimal cost of the WCSP and the Boolean WCSP are also NP-hard. The first two tasks can be viewed as classification problems and the other two tasks can be viewed as regression problems.

While there have been many attempts to apply ML techniques for the above tasks, none of these attempts have yielded spectacular results: They do not consistently produce high-quality outcomes. In the CSP domain, some attempts include the application of Support Vector Machines (SVMs) [6], decision tree learning [7], clustering [8], and k-nearest neighbors [9]. A relevant survey can be found in [10]. In the SAT domain, Neural Networks (NNs) are used in state-of-the-art systems such as NeuroSAT [11] and End2EndSAT [12]. A relevant survey can be found in [13]. In the WCSP domain, there is no existing literature to the best of our knowledge.

Despite the limited success of previous ML approaches, a newly developed framework, called FastMapSVM, has been shown to be successful for predicting CSP satisfiability [14]. FastMapSVM leverages a distance function on *pairs* of CSP instances instead of trying to characterize *individual* CSP instances. A good distance function on a pair of CSP instances should be invariant to the permutations of the variables and their domain values within each CSP instance: In fact, one such maxflow-based distance function is presented in [14].

In this paper, we advance FastMapSVM in various ways.



X_1 \ X_2	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}
d_{11}	0	1	0	0	0
d_{12}	1	1	0	0	0
d_{13}	0	1	1	0	1
d_{14}	0	0	1	0	0
d_{15}	0	0	1	0	0

	X_1			X_2			X_3		
	d_{11}	d_{12}	d_{13}	d_{21}	d_{22}	d_{23}	d_{31}	d_{32}	d_{33}
d_{11}	1	0	0	1	1	1	0	0	1
X_1 d_{12}	0	1	0	0	0	1	1	1	1
d_{13}	0	0	1	1	1	1	1	0	0
d_{21}	1	0	1	1	0	0	1	1	1
X_2 d_{22}	1	0	1	0	1	0	1	0	1
d_{23}	1	1	1	0	0	1	1	0	1
d_{31}	0	1	1	1	1	1	1	0	0
X_3 d_{32}	0	1	0	1	0	0	0	1	0
d_{33}	1	1	0	1	1	1	0	0	1

Fig. 1. The left panel shows the $(0,1)$-matrix representation of a single constraint $C(X_1, X_2)$. The right panel shows the $(0,1)$-matrix representation of an entire binary CSP instance on three variables.

For predicting the satisfiability of CSP and SAT instances, we enhance the maxflow-based distance function presented in [14] with strong path-consistency. We show that doing so improves the accuracy in many cases. However, establishing strong path-consistency can be very expensive. In such cases, we propose a truncated version of it to simultaneously achieve both good accuracy and efficiency. For predicting the optimal cost of WCSP instances, we design a distance function that also utilizes maxflow computations and replaces the SVM component of FastMapSVM by a Support Vector-based Regression (SVR) component. We demonstrate the success of our FastMapSVM and FastMapSVR approaches over competing state-of-the-art ML algorithms in all three domains: In the CSP domain, we demonstrate our success on several CSP benchmark suites; in the SAT domain, we demonstrate our success on hard 3-SAT instances drawn from the phase transition region; and in the WCSP domain, we demonstrate our success on a wide range of randomly generated WCSP instances.

II. Preliminaries and Background

In this section, we describe the background material pertinent to CSPs, path-consistency, SAT, and WCSPs.

A. Constraint Satisfaction Problems

A CSP instance is defined by a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X_1, X_2 \ldots X_N\}$ is a set of variables and $\mathcal{C} = \{C_1, C_2 \ldots C_M\}$ is a set of constraints on subsets of them. Each variable X_i is associated with a finite discrete-valued domain $D_i \in \mathcal{D}$, and each constraint C_i is a pair $\langle S_i, R_i \rangle$ defined on a subset of variables $S_i \subseteq \mathcal{X}$, called the *scope* of C_i. $|S_i|$ is referred to as the *arity* of the constraint. $R_i \subseteq D_{S_i}$ ($D_{S_i} = \times_{X_j \in S_i} D_j$) denotes all compatible tuples of D_{S_i} allowed by the constraint. The absence of a constraint on a certain subset of the variables is equivalent to a constraint on the same subset of the variables that allows all combinations of values to them. A *solution* of a CSP instance is an assignment of values to all the variables from their respective domains such that all the constraints are satisfied.

A binary CSP instance has at most two variables per constraint. However, binary CSPs are representationally as powerful as general CSPs with bounded arity of the constraints [15]. Given a binary CSP instance, we can build a matrix representation for it using a simple mechanism. First, we assume that the domain values of each variable are ordered in some way. (We can simply use the order in which the domain values of each of the variables are specified.) Under such an ordering, we can represent each binary constraint as a 2-dimensional matrix with all its entries set to either '1' or '0' based on whether the corresponding combination of values to the participating variables is allowed or not by that constraint. The left panel of Figure 1 shows the $(0,1)$-matrix representation of a binary constraint between two variables X_1 and X_2 with domain sizes of five each. The combination of values $(X_1 \leftarrow d_{12}, X_2 \leftarrow d_{21})$ is an allowed combination, and the corresponding entry in the matrix is therefore set to '1'. However, the combination of values $(X_1 \leftarrow d_{14}, X_2 \leftarrow d_{22})$ is a disallowed combination, and the corresponding entry is therefore set to '0'. In general, d_{ip} denotes the p-th domain value of X_i in an index ordering on the domain values of X_i.

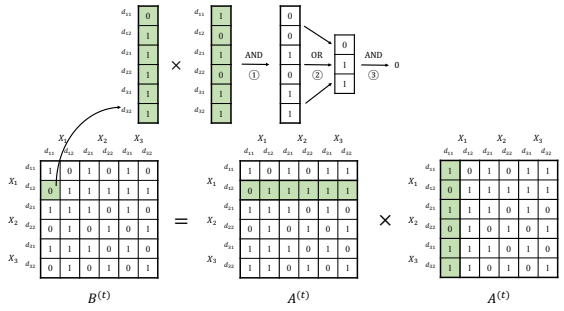
The $(0,1)$-matrix representation of an entire binary CSP instance can be constructed simply by stacking up the matrix representations for the individual constraints into a bigger "block" matrix. The right panel of Figure 1 illustrates how a binary CSP instance on three variables X_1, X_2, and X_3 can be represented as a "mega-matrix" with three sets of rows and three sets of columns. Each block-entry inside this mega-matrix is the matrix representation of the direct constraint between the corresponding row and column variables. Therefore, the matrix representation of an entire binary CSP instance has $\sum_{i=1}^{N} |D_i|$ rows and $\sum_{i=1}^{N} |D_i|$ columns.

B. Path-Consistency

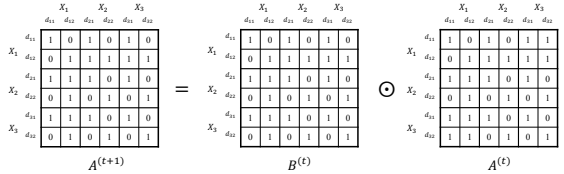
A binary CSP instance is *arc-consistent* if and only if for all distinct variables X_i and X_j, and for every instantiation of X_i, there exists an instantiation of X_j such that the direct constraint between them is satisfied. Similarly, a binary CSP instance is *path-consistent* if and only if for all distinct variables X_i, X_j, and X_k, and for every instantiation of X_i and X_j that satisfies the direct constraint between them, there exists an instantiation of X_k such that the direct constraints between X_i and X_k and between X_j and X_k are also satisfied.

Establishing path-consistency on a binary CSP instance typically takes $O(N^3 \tilde{D}^3)$ time, where $\tilde{D}$ is the size of the largest domain [15]. In practice, a good path-consistency algorithm iteratively tightens the constraints of the binary CSP instance until convergence [16]. If A is the $(0,1)$-matrix representation of the binary CSP instance, it is iteratively modified by converting some of its '1's to '0's in each iteration. Let $A^{(t)}$ be the matrix after the t-th iteration, with $A^{(0)} = A$. In iteration $t+1$, $A^{(t+1)}$ is set to $B^{(t)} \odot A^{(t)}$, where $B^{(t)}$ is defined as $A^{(t)} \times A^{(t)}$. All matrices have the same dimensions.

Figure 2 illustrates the two operations $\times$ and $\odot$. The operation $B^{(t)} = A^{(t)} \times A^{(t)}$ derives each $B_{ij}^{(t)}$ from the block vectors $A_{i*}^{(t)}$ and $A_{*j}^{(t)}$ sectioned by N different variables. The two block vectors are combined via a sequence of logical steps. In the first step, the corresponding "bits" of the two vectors are logically AND-ed. In the second step, the bits within each block (section) of the resulting vector are logically OR-ed to yield an N-bit vector. In the third step, all these N bits are



(a) The $\times$ operation



(b) The $\odot$ operation

Fig. 2. Illustrates the $(t+1)$-th iteration of a practical path-consistency algorithm using the two operations $\times$ and $\odot$ on $(0,1)$-matrices.

logically AND-ed to produce $B_{ij}^{(t)}$. The three steps execute the combinatorial principle of looking for a common support in every variable's domain. The operation $A^{(t+1)} = B^{(t)} \odot A^{(t)}$ performs an element-wise logical AND of the two matrices $B^{(t)}$ and $A^{(t)}$ to yield $A^{(t+1)}$. It is used to monotonically tighten the constraints until convergence.

In principle, path-consistency requires $A^{(t)}$ to converge. However, this can be expensive in practice. Hence, a truncated version of path-consistency, in which only a user-specified number of iterations are carried out, is sometimes more viable.

C. 3-*Satisfiability*

The SAT problem is a special case of the CSP in which all variables are Boolean. While the 2-SAT problem can be solved in polynomial time, the 3-SAT problem not only is NP-hard but is also used as a popular canonical form of the general SAT problem. Figure 3 shows how to interpret a 3-SAT instance as a binary CSP instance. We associate a CSP variable with each 3-SAT clause. Its domain is the set of seven satisfying assignments to the SAT variables participating in that clause. The binary CSP constraints enforce that the shared SAT variables are assigned consistent values.

D. Weighted Constraint Satisfaction Problems

The WCSP is a generalization of the CSP in which the constraints are no longer "hard" but are weighted instead [1]. Formally, a WCSP instance is defined by a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X_1, X_2 \ldots X_N\}$ is a set of N variables, $\mathcal{D} = \{D_1, D_2 \ldots D_N\}$ is a set of N finite discrete-valued domains, and $\mathcal{C} = \{C_1, C_2 \ldots C_M\}$ is a set of M weighted constraints. Each variable $X_i \in \mathcal{X}$ can be assigned a value from its associated domain $D_i \in \mathcal{D}$. Each weighted constraint $C_i \in \mathcal{C}$ is defined over a certain subset of the variables $S_i \subseteq \mathcal{X}$, called the *scope* of C_i. $|S_i|$ is referred to as the *arity* of the weighted constraint. C_i associates a non-negative weight

$(x_1 \vee \overline{x_2} \vee x_3)$

x_1	x_2	x_3	Value
1	1	1	true
1	1	0	true
1	0	1	true
1	0	0	true
0	1	1	true
0	1	0	false
0	0	1	true
0	0	0	true

$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_4 \vee \overline{x_5}) \wedge (x_1 \vee x_3 \vee \overline{x_4})$

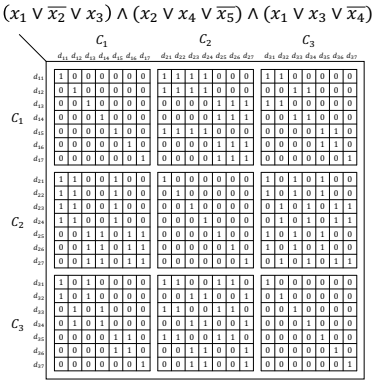


Fig. 3. The left panel shows the logic of a single 3-SAT clause. The right panel shows a 3-SAT instance with five variables and three clauses along with the interpretation of it as a binary CSP instance. The seven domain values of each CSP variable follow the same order of enumerating the satisfying assignments of its corresponding 3-SAT clause, as shown in the left panel.

X_1 \ X_2	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}
d_{11}	10	79	20	1	3
d_{12}	41	5	33	8	6
d_{13}	12	6	1	10	31
d_{14}	19	15	9	2	36
d_{15}	16	3	5	10	40

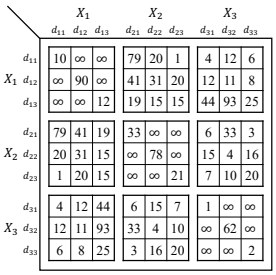


Fig. 4. The left panel shows the matrix representation of a single weighted constraint $C(X_1, X_2)$. The right panel shows the matrix representation of an entire binary WCSP instance on three variables.

with each possible assignment of values to the variables in S_i. An optimal solution of a WCSP instance is an assignment of values to all the variables from their respective domains such that the sum of the weights specified by each weighted constraint in $\mathcal{C}$ is minimized.

A binary WCSP instance has at most two variables per weighted constraint. However, binary WCSPs are representationally as powerful as general WCSPs [17]. Given a binary WCSP instance, we can build a matrix representation of it using a simple extension of the method used for binary CSP instances. The left panel of Figure 4 shows the matrix representation of a binary weighted constraint between two variables X_1 and X_2 with domain sizes of five each. Each combination of values assigned to X_1 and X_2 is associated with a non-negative weight. As in the case of binary CSP instances, the matrix representation of an entire binary WCSP instance can be constructed by stacking up the matrix representations of the individual weighted constraints into a bigger block matrix. The right panel of Figure 4 illustrates this for a binary WCSP instance on three variables X_1, X_2, and X_3.

III. FastMapSVM and FastMapSVR

FastMapSVM is a recently developed ML framework that is based on comparing *pairs* of objects instead of characterizing *individual* objects. It has been demonstrated to be particu-

larly useful for classification tasks on complex combinatorial objects [14], [18]. In this section, we first describe it and then extend it to FastMapSVR for regression tasks on similar complex combinatorial objects.

Both FastMapSVM and FastMapSVR rely on FastMap [19], a Data Mining algorithm that embeds complex objects—such as audio signals, seismograms, DNA sequences, electrocardiograms, or magnetic-resonance images—into a K-dimensional Euclidean space, for a user-specified value of K. FastMap does this embedding by leveraging a domain-specific distance function $D(\cdot, \cdot)$ on pairs of objects. In the Euclidean embedding that it produces, the Euclidean distances between every pair of the point representations of objects approximate the domain-specific distances between them. While specific qualities of the approximation depend on K, similar objects, as quantified by $D(\cdot, \cdot)$, generally map to nearby points in the Euclidean embedding and dissimilar objects map to distant points. Moreover, although FastMap preserves $O(N^2)$ pairwise distances between N objects, it generates the Euclidean embedding in only $O(KN)$ time. Because of its efficiency, FastMap has already found numerous real-world applications, including in Data Mining [19], shortest-path computations [20], solving combinatorial optimization problems on graphs [21], and community detection and block modeling [22].

A. FastMapSVM

FastMapSVM [18] combines the strengths of FastMap and SVMs. In the first phase, it invokes FastMap to convert complex objects to their point representations in Euclidean space. In the second phase, it invokes SVMs and kernel methods for learning to classify the points in this Euclidean space. FastMapSVM has several advantages over other methods.

First, FastMapSVM leverages domain-specific knowledge via a distance function instead of relying on complex ML models to infer the underlying structure in the data entirely. In many real-world domains with complex objects, a distance function on *pairs* of objects is well defined and easy to compute. In such domains, FastMapSVM is more easily applicable than other ML algorithms that focus on learning the features of *individual* objects.

Second, FastMapSVM facilitates interpretability, explainability, and visualization. While the objects in the problem domain may themselves be complex, FastMapSVM embeds them in a Euclidean space by considering only the domain-specific distance function. In effect, it simplifies the descriptions of the objects by assigning Euclidean coordinates to them. Moreover, since the distance function is itself user-supplied and encapsulates domain knowledge, FastMapSVM naturally facilitates interpretability and explainability. In fact, it even provides a perspicuous visualization of the objects and the classification boundaries between them.

Third, FastMapSVM uses significantly smaller amounts of time and data for model training compared to other ML algorithms. It stores explicit references to some of the original objects, referred to as *pivots* [18]. While making predictions, objects in the test instances are compared directly to the pivots using the user-supplied distance function. Thereby, FastMapSVM obviates the need to learn a complex transformation of the input data and thus significantly reduces the amounts of time and data required for model training.

Fourth, FastMapSVM extends the applicability of SVMs and kernel methods to complex objects. Generally speaking, these are particularly good for classification tasks since they recognize and represent complex nonlinear classification boundaries very elegantly. However, the SVM machinery requires the objects to be represented as points in a Euclidean space. Often, it is very difficult to represent complex objects as precise geometric points without introducing inaccuracy or losing domain-specific representational features. FastMapSVM revives the SVM approach by leveraging a distance function and creating a low-dimensional Euclidean embedding of the complex objects in the problem domain.

B. FastMapSVR

We extend FastMapSVM to a new ML framework, called FastMapSVR, to be able to address regression problems (in the WCSP domain). Similar to FastMapSVM, FastMapSVR combines FastMap with the machinery of support vectors but uses a regression module instead of a classification module. Such a regression module based on support vectors is presented in [23] and is popularly referred to as SVR. Hence, FastMapSVR first invokes FastMap to embed complex objects in a Euclidean space and then invokes SVR to solve regression problems on them using their simplified point representations.

IV. Distance Functions

The FastMap component of both the FastMapSVM and the FastMapSVR algorithms relies on a domain-specific distance function, which plays a critical rule in their success. In this section, we present the distance functions necessary for FastMapSVM in the CSP and the SAT domains and FastMapSVR in the WCSP domain. In each case, it is easy to observe that we satisfy the requirements of the FastMap algorithm on the distance function: It should be non-negative, symmetric, and evaluate to 0 on identical objects.

A. Distance Function on Binary CSPs

Figure 5 illustrates a maxflow-based distance function on pairs of binary CSP instances $\mathcal{I}_1$ and $\mathcal{I}_2$ [14]. $\mathcal{I}_1$ and $\mathcal{I}_2$ can be of different sizes. The maxflow computations are utilized in: (a) a single high-level 'maximum matching of minimum cost' problem posed on the variables of $\mathcal{I}_1$ and $\mathcal{I}_2$, and (b) multiple low-level 'maximum matching of minimum cost' problems posed on the domain values of pairs of variables, one from each of $\mathcal{I}_1$ and $\mathcal{I}_2$.

The high-level 'maximum matching of minimum cost' problem is posed on a complete bipartite graph, in which the two partitions of the bipartite graph correspond to the variables of $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. If the numbers of variables in $\mathcal{I}_1$ and $\mathcal{I}_2$ do not match, dummy variables are added to the CSP instance with fewer variables. Figure 5 (top panel) illustrates this for $\mathcal{I}_1$ and $\mathcal{I}_2$ with variables $\{X_1, X_2, X_3, X_4\}$

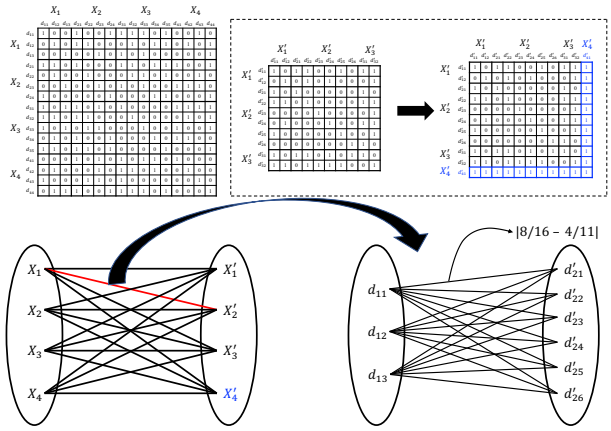


Fig. 5. The top panel shows two CSP instances with variables $\{X_1, X_2, X_3, X_4\}$ (left) and $\{X'_1, X'_2, X'_3\}$ (middle), respectively. A dummy variable X'_4 with a singleton domain is added to the CSP instance with fewer variables (right). The bottom panel (left) shows how a 'maximum matching of minimum cost' problem is posed on a complete bipartite graph with the variables of the two CSP instances in each partition. The cost annotating the edge between X_i and X'_j is itself derived from a 'maximum matching of minimum cost' problem posed on the domain values of X_i and X'_j. The bottom panel (right) shows this 'maximum matching of minimum cost' problem for the variables X_1 and X'_2. It is posed on a complete bipartite graph with the domain values $\{d_{11}, d_{12}, d_{13}\}$ and $\{d'_{21}, d'_{22}, d'_{23}, d'_{24}, d'_{25}, d'_{26}\}$ in each partition. The cost annotating the edge between d_{11} and d'_{21} is the absolute value of the difference between the average compatibility of d_{11} and the average compatibility of d'_{21}.

and $\{X'_1, X'_2, X'_3\}$, respectively. A dummy variable X'_4 is added to $\mathcal{I}_2$. The dummy variable has a single domain value that is designed to be consistent with all domain values of all other variables, since this does not change the CSP instance.

The distance between $\mathcal{I}_1$ and $\mathcal{I}_2$ is defined to be the cost of the 'maximum matching of minimum cost' on the high-level bipartite graph. This bipartite graph has an edge between every X_i in $\mathcal{I}_1$ and every X'_j in $\mathcal{I}_2$. The cost annotating an edge between X_i and X'_j is itself set to be the cost of the 'maximum matching of minimum cost' posed at the low level on the domain values of X_i and X'_j. Figure 5 (bottom-left panel) shows the high-level bipartite graph and highlights an edge between X_1 and X'_2 for explanation of the low-level 'maximum matching of minimum cost'.

The low-level 'maximum matching of minimum cost' problem posed on the domain values of X_i and X'_j also uses a complete bipartite graph. The two partitions consist of the domain values of X_i and X'_j, respectively. The cost annotating the edge between d_{ip} and d'_{jq} is the absolute value of the difference between the average compatibility of d_{ip} and the average compatibility of d'_{jq}. Figure 5 (bottom-right panel) shows the low-level 'maximum matching of minimum cost' problem posed on the domain values of X_1 and X'_2. The domains of X_1 and X'_2 are $\{d_{11}, d_{12}, d_{13}\}$ and $\{d'_{21}, d'_{22}, d'_{23}, d'_{24}, d'_{25}, d'_{26}\}$, respectively. Consider the edge between d_{11} and d'_{21}. The average compatibility of d_{11} is the fraction of '1's in the column 'd_{11}' in the matrix representation of $\mathcal{I}_1$. This fraction is equal to $8/16$. The average compatibility of d'_{21} is the fraction of '1's in the column 'd'_{21}' in the matrix representation of $\mathcal{I}_2$ after adding the dummy variable X'_4. This fraction is equal to $4/11$. Therefore, the cost annotating the edge between d_{11} and d'_{21} is equal to $|8/16 - 4/11|$.

The distance function is invariant to both variable-orderings and domain value-orderings. This property allows us to avoid data augmentation methods and address the drawbacks of traditional ML algorithms that are otherwise forced to learn exponentially large equivalence classes of CSP instances via permutations of variable-orderings and domain value-orderings.

The distance function can also be invoked after establishing various levels of local-consistency on the two CSP instances $\mathcal{I}_1$ and $\mathcal{I}_2$. Doing so leverages the power of constraint propagation methods in FastMapSVM and FastMapSVR seamlessly. However, establishing l-consistency for $l > 3$ is not only expensive but also creates non-binary constraints [15]. Hence, $l = 3$, that is, establishing path-consistency is an appropriate choice. Sometimes, path-consistency can also be fairly expensive, in which case we establish only a truncated version of it.

B. Distance Function on 3-SAT Problems

As mentioned before, a 3-SAT instance can be interpreted as a binary CSP instance. Hence, the same distance function discussed above, with or without establishing path-consistency, is applicable on pairs of 3-SAT instances as well.

C. Distance Function on Binary WCSPs

An appropriate distance function on binary WCSPs can be derived from the distance function discussed above for binary CSPs. Only two modifications are required. The first is in the way that dummy variables are added. A dummy variable is assigned a singleton domain and all weights associated with this unique domain value are set to 0. The second is in the way that the low-level 'maximum matching of minimum cost' problems are posed. The cost annotating the edge between d_{ip} and d'_{jq} is set to be the absolute value of the difference between the average paired-cost of d_{ip} and the average paired-cost of d'_{jq}. The average *paired-cost* of a domain value d_{ip} is the average of all weights specified in the WCSP instance that involve $X_i \leftarrow d_{ip}$.

V. Experimental Results

In this section, we experimentally evaluate FastMapSVM on predicting CSP and 3-SAT satisfiability and FastMapSVR on predicting the optimal cost of WCSP instances against other state-of-the-art competing approaches.

A. Experimental Setup

In the CSP domain, we evaluate FastMapSVM against three competing approaches. The first is a state-of-the-art deep graph convolutional neural network (DGCNN) [24]. The second is a state-of-the-art graph isomorphism network (GIN) [25]. Both these networks ingest a CSP instance in the form of a graph and do not require the CSP training and test instances to be of the same size. The third is the truncated path-consistency (TPC) algorithm: TPC first establishes truncated path-consistency and then checks whether any variable's domain is annihilated. If so, it declares the CSP instance to be

'unsatisfiable'; else, 'satisfiable'. FastMapSVM can also be invoked after establishing TPC, in which case it is referred to as TPC-FastMapSVM.

In the SAT domain, we evaluate FastMapSVM against two more state-of-the-art competing approaches: NeuroSAT [11] and End2EndSAT [12]. Both are based on training NNs to learn the characteristics of satisfiable and unsatisfiable SAT instances. While all other approaches from the CSP domain can be included in the evaluation by interpreting the SAT instances as binary CSP instances, GIN has to be excluded since it runs out of memory.

In the WCSP domain, since there are no known competing approaches, we evaluate FastMapSVR against a baseline approach that randomly generates assignments.

We implemented FastMapSVM, FastMapSVR, and TPC in Python3 and ran them on a laptop with an Apple M2 chip with 16 GB memory. However, we ran DGCNN, GIN, NeuroSAT, and End2EndSAT on a Linux system with a P100 GPU, since they need more computational resources. The different platforms are inconsequential to the comparative performances of various approaches with respect to effectiveness.

B. Instance Generation

In the CSP domain, we generated four benchmark suites: Blocked n-Queens, Graph Coloring, Sudoku, and Zebra puzzles. For Blocked n-Queens, we used the CSPLib [26] generator, in which n is an integer chosen uniformly at random from the interval $[2, 50]$. For Graph Coloring, we generated Erdős-Rényi graphs [27], in which the number of vertices is chosen uniformly at random from the interval $[4, 100]$ and the probability parameter for edge creation, fixed for each instance, is chosen independently and uniformly at random from the interval $[0.1, 0.9]$. For Sudoku, we chose the mask rate uniformly at random from the interval $[0.2, 0.8]$. For Zebra puzzles, we used five houses and five attributes based on [28]. For the Sudoku and Zebra puzzle instances, we used 'python-constraint' to ensure that they have unique solutions: We did this to make the prediction tasks more challenging. In each of the four categories, we generated $1,000$ instances for training and $1,000$ instances for testing.

In the SAT domain, we randomly generated hard 3-SAT instances from the phase transition region with 50 variables and 218 clauses. Each clause contains three distinct randomly chosen variables; and each chosen variable in it appears as a positive or a negative literal with probability 0.5. We generated $1,000$ instances for training and $1,000$ instances for testing.

In the WCSP domain, we generated the binary WCSP instances as follows. First, we chose N, the number of variables, uniformly at random from the interval $[2, 20]$. Second, we chose the domain size of each variable independently and uniformly at random from the interval $[1, 10]$. Third, for each unary and binary weighted constraint, we chose each weight in it independently and uniformly at random from the interval $[1, 100]$. We used Toulbar2 [29] to generate the ground-truth optimal cost solution of each instance. We generated $1,000$ instances for training and $1,000$ instances for testing.

TABLE I
SHOWS THE COMPARATIVE PERFORMANCES OF VARIOUS APPROACHES ON FOUR BENCHMARK SUITES IN THE CSP DOMAIN. TPC IMPLEMENTS TWO ITERATIONS OF PATH-CONSISTENCY OPERATIONS.

Benchmark	Approach	Accuracy	Recall	Precision	F1
Blocked n-Queens	FastMapSVM	**100.0**	**100.0**	**100.0**	**100.0**
	DGCNN	88.3	92.2	85.5	88.7
	GIN	96.2	92.4	**100.0**	96.0
Graph Coloring	FastMapSVM	**93.0**	**99.2**	88.3	**93.4**
	DGCNN	86.3	93.4	81.8	87.2
	GIN	83.0	73.6	**90.6**	81.2
Sudoku	TPC-FastMapSVM	**92.2**	93.0	**91.5**	**92.3**
	TPC	79.4	**100.0**	70.8	82.9
	DGCNN	79.5	77.6	80.7	79.1
	GIN	82.0	89.0	78.1	83.2
Zebra	TPC-FastMapSVM	**94.6**	99.6	**90.5**	**94.9**
	TPC	51.9	**100.0**	51.0	67.5
	DGCNN	51.5	85.4	50.9	63.8
	GIN	63.1	68.4	61.8	65.0

C. Results

We present three sets of results: one each for the CSP domain, the SAT domain, and the WCSP domain.

In the CSP domain, Table I and Figure 6 show representative results. From Table I, it is easy to observe that FastMapSVM or its TPC variant, that is, TPC-FastMapSVM, significantly outperforms all other approaches.[1] Among the other approaches, GIN is the second best on the Blocked n-Queens, Sudoku, and Zebra puzzle instances. DGCNN is the second best on the Graph Coloring instances.

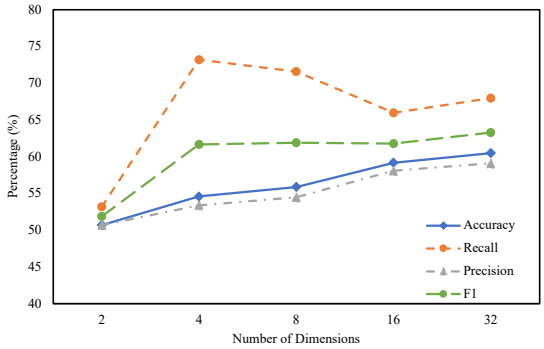
On the Sudoku and Zebra puzzle instances, FastMapSVM's performance is not very impressive. However, with a little help from TPC, it performs significantly better than not only FastMapSVM but all other approaches, including TPC itself. This demonstrates FastMapSVM's unique ability to effectively leverage the power of constraint propagation methods in an ML framework. From Figure 6, it is easy to validate these arguments and observe the general benefits of TPC in the FastMapSVM framework with only two iterations of path-consistency operations. It is also easy to observe the benefits of increasing the number of dimensions of the Euclidean embedding generated by the FastMap component of FastMapSVM.[2]

In the SAT domain, Table II shows representative results. It compares DGCNN, NeuroSAT, End2EndSAT, TPC, and TPC-FastMapSVM: the last two with a varying number of TPC iterations, that is, the number of iterations of path-consistency operations. TPC-FastMapSVM is also evaluated by varying the number of dimensions. It is easy to observe that DGCNN, NeuroSAT, and End2EndSAT perform very poorly.[3] This is not surprising despite all three of these approaches being considered "state of the art". This is primarily because 3-SAT instances drawn from the phase transition region are notoriously hard to analyze. TPC achieves a reasonable performance when the number of TPC iterations increases, that is, when it leverages an increasing amount of constraint propagation.

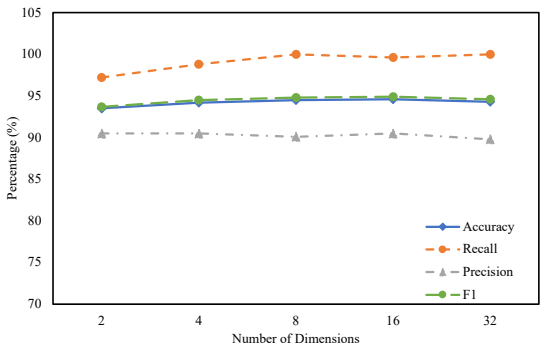
[1]For each benchmark suite, we trained DGCNN and GIN for 100 epochs with a learning rate of 0.0001 and a minibatch size of 100.

[2]Increasing the number of dimension yields diminishing returns for TPC-FastMapSVM since it achieves good performance even in low dimensions.

[3]We trained DGCNN, NeuroSAT, and End2EndSAT for 100, 50, and 1000 epochs, respectively, with a learning rate of 0.0001, for "good" performance.



(a) Influence of the number of dimensions on the performance of FastMapSVM on Zebra puzzle instances



(b) Influence of the number of dimensions on the performance of TPC-FastMapSVM on Zebra puzzle instances

Fig. 6. Shows the behavior of FastMapSVM and TPC-FastMapSVM with respect to the number of dimensions on Zebra puzzle instances. The performance metrics include the accuracy, recall, precision, and the F1 score. TPC implements two iterations of path-consistency operations.

TABLE II
SHOWS THE COMPARATIVE PERFORMANCES OF VARIOUS APPROACHES IN THE SAT DOMAIN.

Approach			Accuracy	Recall	Precision	F1
DGCNN			50.0	0.0	0.0/0.0	0.0
NeuroSAT			57.0	56.0	57.1	56.6
End2EndSAT			50.0	100.0	50.0	66.7
	TPC Iterations					
TPC	3		50.0	100.0	50.0	66.7
	5		50.0	100.0	50.0	66.7
	7		50.0	100.0	50.0	66.7
	9		56.9	100.0	53.7	69.9
	11		79.0	100.0	70.4	82.6
	12		88.7	100.0	81.6	89.8
	TPC Iterations	Dimensions				
TPC-FastMapSVM	3	2	55.7	61.6	55.1	58.2
		4	55.5	53.8	55.7	54.7
		8	55.2	54.4	55.3	54.8
		16	55.5	58.6	55.2	56.8
	5	2	57.4	63.0	56.7	59.7
		4	57.5	63.2	56.7	59.8
		8	57.7	64.8	56.7	60.5
		16	57.5	64.2	56.6	60.2
	7	2	61.7	70.6	59.9	64.8
		4	61.5	70.4	59.8	64.6
		8	61.4	71.8	59.4	65.0
		16	61.9	73.4	59.7	65.8
	9	2	66.7	77.4	63.8	69.9
		4	66.7	77.6	63.7	70.0
		8	67.0	78.2	63.9	70.3
		16	66.9	78.0	63.8	70.2
	11	2	80.5	87.0	77.0	81.7
		4	80.4	86.6	77.0	81.5
		8	80.2	86.2	77.0	81.3
		16	82.3	89.2	77.2	82.7
	12	2	89.0	93.6	85.7	89.5
		4	90.3	96.2	86.0	90.8
		8	92.0	100.0	86.2	92.6
		16	92.4	100.0	86.8	92.9

However, it is also easy to observe that TPC-FastMapSVM outperforms TPC for the same number of TPC iterations and any number of dimensions. Moreover, the performance of TPC-FastMapSVM increases with an increasing number of dimensions, especially when the number of TPC iterations is relatively large. In fact, TPC-FastMapSVM with 12 TPC iterations and 16 dimensions achieves extraordinarily good performance on this exceptionally hard task.

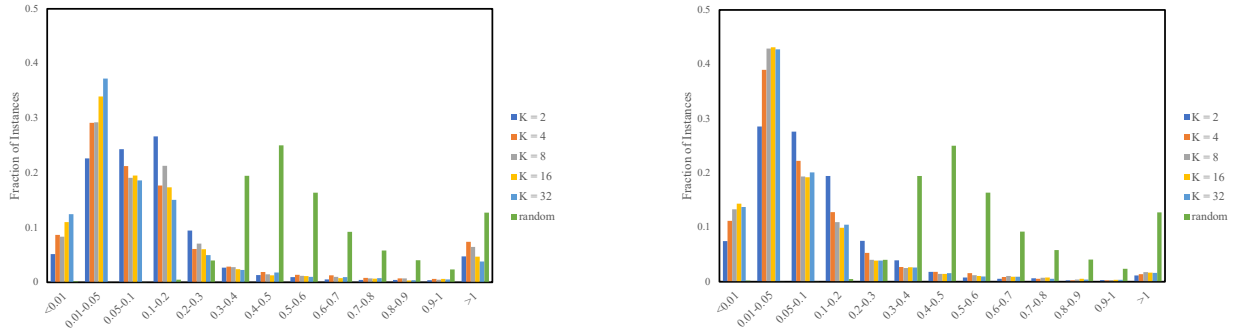
In the WCSP domain, Figure 7 shows representative results. It compares the performance of FastMapSVR, for different values of K and C, against that of a baseline procedure. Here, K is the number of dimensions of the Euclidean embedding generated by the FastMap component of FastMapSVR and C is the regularization parameter used in the SVR component of FastMapSVR. The baseline procedure assigns random values to all the WCSP variables from their respective domains. The performance metric is the distribution of relative error. For each instance, the relative error is given by the absolute value of the difference between the predicted cost and the optimal cost divided by the optimal cost. Figure 7 shows the distribution of relative error on the $1,000$ test instances, averaged over five trials, for each competing approach.[4]

From Figure 7, it is easy to observe that FastMapSVR produces smaller relative errors on a larger fraction of the instances. For example, in Figures 7(a) and 7(b), FastMapSVR produces relative errors greater than 0.3 on a much smaller fraction of the instances compared to the baseline procedure. Hence, even the average Root Mean Squared Error (RMSE) of FastMapSVR is lower than that of the baseline procedure. Moreover, the quality of the results produced by FastMapSVR depends on K and C. It generally improves with an increasing value of K and an increasing value of C. For example, the average RMSE value of FastMapSVR with $K = 16$ decreases as follows: 24.66, 0.56, 0.45, and 0.38 for $C = 8$, 32, 128, and 512, respectively.

VI. CONCLUSIONS

In this paper, we proposed novel enhancements to FastMapSVM and demonstrated their success in the CSP domain, the SAT domain, and the WCSP domain. In the CSP domain, we enhanced FastMapSVM with TPC to be able to harness the power of constraint propagation methods. Consequently, we demonstrated the superiority of TPC-FastMapSVM over competing approaches on benchmark CSP instances. In the SAT domain, we demonstrated the superiority of TPC-FastMapSVM over competing approaches on hard 3-SAT instances drawn from the phase transition region by interpreting them as binary CSP instances. In the WCSP domain, we modified FastMapSVM and proposed

[4]FastMapSVR can produce different results in different trials on the same instance because it uses randomness.



(a) Distribution of relative error for $C = 32$

(b) Distribution of relative error for $C = 512$

Fig. 7. Shows the behavior of FastMapSVR with respect to the number of dimensions K and the regularization parameter C on randomly generated binary WCSP instances.

FastMapSVR along with a novel distance function on binary WCSPs. We demonstrated the superiority of FastMapSVR over a baseline approach on randomly generated binary WCSP instances. Overall, our FastMapSVM, TPC-FastMapSVM, and FastMapSVR frameworks have broader applicability and various representational and combinatorial advantages compared to other ML approaches. They also generally facilitate the integration of constraint reasoning and ML methods.

REFERENCES

[1] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier, "Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison," *Constraints*, 1999.

[2] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole, "CP-Nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements," *Journal of Artifical Intelligence Research*, 2004.

[3] V. Kolmogorov, "Primal-Dual Algorithm for Convex Markov Random Fields," Microsoft Research, Tech. Rep. MSR-TR-2005-117, 2005.

[4] M. Zytnicki, C. Gaspin, and T. Schiex, "DARN! A Weighted Constraint Solver for RNA Motif Localization," *Constraints*, 2008.

[5] J. S. Yedidia, W. T. Freeman, and Y. Weiss, *Understanding Belief Propagation and its Generalizations*. Morgan Kaufmann Publishers Inc., 2003.

[6] A. Arbelaez, Y. Hamadi, and M. Sebag, "Continuous Search in Constraint Programming," in *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, 2010.

[7] I. P. Gent, C. A. Jefferson, L. Kotthoff, I. J. Miguel, N. C. A. Moore, P. Nightingale, and K. Petrie, "Learning When to Use Lazy Learning in Constraint Solving," in *Proceedings of the European Conference on Artificial Intelligence*, 2010.

[8] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "ISAC–Instance-Specific Algorithm Configuration," in *Proceedings of the European Conference on Artificial Intelligence*, 2010.

[9] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, "Using Case-Based Reasoning in an Algorithm Portfolio for Constraint Solving," in *Proceedings of the Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[10] L. Kotthoff, "Algorithm Selection for Combinatorial Search Problems: A Survey," *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, 2016.

[11] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT Solver from Single-Bit Supervision," *CoRR*, 2018.

[12] C. Cameron, R. Chen, J. Hartford, and K. Leyton-Brown, "Predicting Satisfiability via End-to-End Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[13] W. Guo, H.-L. Zhen, X. Li, W. Luo, M. Yuan, Y. Jin, and J. Yan, "Machine Learning Methods in Solving the Boolean Satisfiability Problem," *Machine Intelligence Research*, 2023.

[14] K. Zheng, A. Li, H. Zhang, and T. K. S. Kumar, "FastMapSVM for Predicting CSP Satisfiability," in *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 2023.

[15] R. Dechter, *Constraint Processing*. Morgan Kaufmann, 2003.

[16] T. K. S. Kumar, L. Cohen, and S. Koenig, "Incorrect Lower Bounds for Path Consistency and More," in *Symposium on Abstraction, Reformulation, and Approximation*, 2013.

[17] J. Larrosa and R. Dechter, "On the Dual Representation of Non-Binary Semiring-Based CSPs," in *Proceedings of the Workshop on Soft Constraints: International Conference on Principles and Practice of Constraint Programming*, 2000.

[18] M. White, K. Sharma, A. Li, T. K. S. Kumar, and N. Nakata, "Classifying Seismograms Using the FastMap Algorithm and Support-Vector Machines," *Communications Engineering*, 2023.

[19] C. Faloutsos and K.-I. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.

[20] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, and T. K. S. Kumar, "The FastMap Algorithm for Shortest Path Computations," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018.

[21] J. Li, A. Felner, S. Koenig, and T. K. S. Kumar, "Using FastMap to Solve Graph Problems in a Euclidean Space," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2019.

[22] A. Li, P. Stuckey, S. Koenig, and T. K. S. Kumar, "A FastMap-Based Algorithm for Block Modeling," in *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2022.

[23] A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," *Statistics and Computing*, 2004.

[24] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An End-to-End Deep Learning Architecture for Graph Classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[25] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *Proceedings of the International Conference on Learning Representations*, 2019.

[26] CSPLib, "CSPLib: A Problem Library for Constraints," 2022. [Online]. Available: https://www.csplib.org

[27] P. Erdős and A. Rényi, "On Random Graphs. I," *Publicationes Mathematicae*, 1959.

[28] T. Chandra, "Zebra Puzzles," 2024. [Online]. Available: https://github.com/tuchandra/zebra

[29] B. Hurley, B. O'Sullivan, D. Allouche, G. Katsirelos, T. Schiex, M. Zytnicki, and S. D. Givry, "Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization," *Constraints*, 2016.