FastMapSVM for Predicting CSP Satisfiability

Kexin Zheng[1]
University of Southern California, Los Angeles, CA, USA

Ang Li[1]
University of Southern California, Los Angeles, CA, USA

Han Zhang
University of Southern California, Los Angeles, CA, USA

T. K. Satish Kumar
University of Southern California, Los Angeles, CA, USA

Abstract

Recognizing the satisfiability of Constraint Satisfaction Problems (CSPs) is NP-hard. Although several Machine Learning (ML) approaches have attempted this task by casting it as a binary classification problem, they have had only limited success for a variety of challenging reasons. First, the NP-hardness of the task does not make it amenable to straightforward approaches. Second, CSPs come in various forms and sizes while many ML algorithms impose the same form and size on their training and test instances. Third, the representation of a CSP instance is not unique since the variables and their domain values are unordered. In this paper, we propose FastMapSVM, a recently developed ML framework that leverages a distance function between pairs of objects. We define a novel distance function between two CSP instances using maxflow computations. This distance function is well defined for CSPs of different sizes. It is also invariant to the ordering on the variables and their domain values. Therefore, our framework has broader applicability compared to other approaches. We discuss various representational and combinatorial advantages of FastMapSVM. Through experiments, we also show that it outperforms other state-of-the-art ML approaches.

2012 ACM Subject Classification Computing methodologies → Machine learning

Keywords and phrases Constraint Satisfaction Problems, Machine Learning, FastMapSVM

Digital Object Identifier 10.4230/LIPIcs.CP.2023.40

Funding This work at the University of Southern California is supported by DARPA under grant number HR001120C0157 and by NSF under grant number 2112533. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the sponsoring organizations, agencies, or the U.S. Government.

1 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.

[1] Corresponding Author

Unfortunately, solving CSPs is NP-hard since they generalize the Satisfiability (SAT) problem. Although many technologies have been developed for solving CSPs in practice [4], they do not sufficiently harness the power of Machine Learning (ML) techniques. While there have been a lot of attempts to apply ML techniques to CSPs, none of these attempts have yielded spectacular results: They do not consistently produce high-quality outcomes. Examples of ML approaches used in the CSP domain include the application of Support Vector Machines (SVMs) [2], linear regression [22], decision tree learning [7, 6], clustering [15, 8], k-nearest neighbors [13], and others [9]. However, most of these approaches have had limited success for a variety of challenging reasons.

First, from a complexity theory perspective, the NP-hardness of the task does not make it amenable to straightforward ML approaches. For example, since a neural network (NN) is essentially a continuous differentiable form of a circuit, it is not straightforward to make NNs effective in the CSP domain. Second, CSPs come in various forms and sizes while many ML algorithms use an architectural framework that imposes the same form and size on their training and test instances. For example, an NN may have a fixed input layer that it uses for the training and test instances alike. Third, the representation of a CSP instance is not unique since the variables and their domain values are unordered. This poses a significant combinatorial challenge for ML algorithms since they have to learn the permutation invariance with respect to orderings on the variables and their domain values. For example, an NN may pose the overhead of having to be trained on all permutations of the same CSP instance to become effective.

In this paper, we consider the problem of predicting the satisfiability of CSP instances using ML. In ML terminology, this is essentially a binary classification problem defined on CSPs with the two possible classification labels "satisfiable" and "unsatisfiable". This classification problem is a cornerstone task for addressing the combinatorics of CSPs. It also serves as a stepping stone for the task of solving CSPs. In fact, any ML framework expected to be viable for solving CSPs should likely first demonstrate its success on solving the aforementioned classification problem on CSPs.

We propose to solve the above classification problem on binary CSPs[2] using a recently developed ML framework called FastMapSVM [19]. While most ML algorithms learn to identify characteristic features of *individual* objects in a class, FastMapSVM leverages a domain-specific distance function on *pairs* of objects. It does this by combining the strengths of FastMap [5] and SVMs. In its first stage, FastMapSVM invokes FastMap, an efficient linear-time algorithm that maps complex objects to points in a Euclidean space, while preserving pairwise distances between them. In its second stage, it invokes SVMs and kernel methods for learning to classify the points in this Euclidean space.

FastMapSVM has demonstrated success on classifying complex objects such as seismograms in Earthquake Science [19]. It offers several advantages over ML algorithms that reason about individual objects instead of pairs of objects. First, FastMapSVM enables domain experts to incorporate their domain knowledge using a distance function. This avoids relying on complex ML models to infer the underlying structure in the data entirely. Second, because the distance function 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. Fourth, it extends the applicability of SVMs and kernel methods to domains with complex objects.

[2] Binary CSPs have at most two variables per constraint but are allowed to have non-Boolean variables. Binary CSPs are representationally as powerful as general CSPs with bounded arity of the constraints.

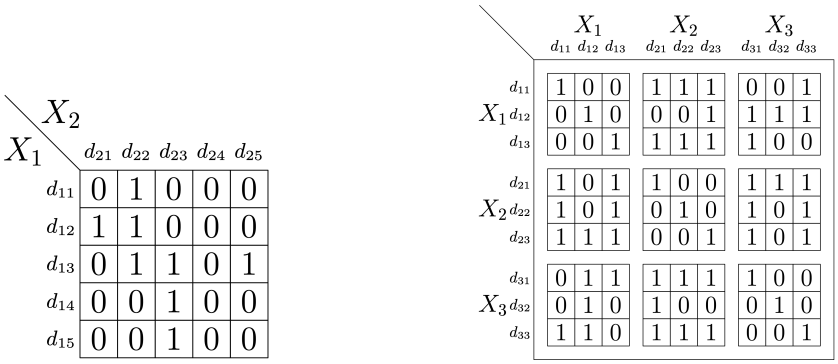


Figure 1 The left panel shows the $(0, 1)$-matrix representation of a single constraint $C(X_1, X_2)$. The right panel shows the $(0, 1)$-matrix representation of an entire binary CSP instance.

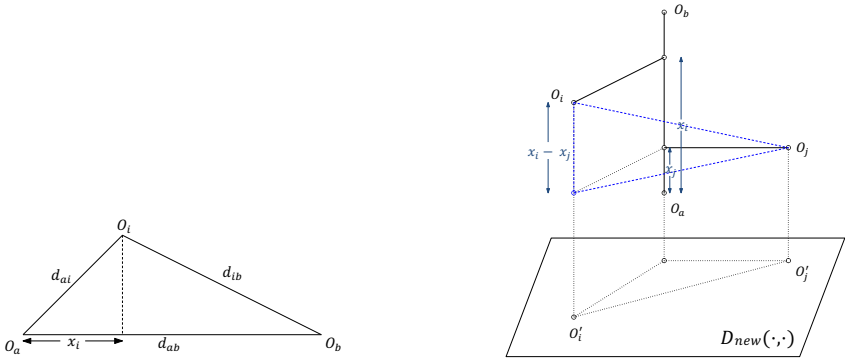
In applying FastMapSVM to the CSP domain, we define a novel distance function between two CSP instances. This distance function uses maxflow computations and is well defined for CSP instances of different sizes. It is also invariant to the ordering on the variables and their domain values in the CSP instances. Therefore, FastMapSVM has broader applicability compared to other ML approaches in the CSP domain. Moreover, since it uses the intelligence of SVMs, kernel methods, and maxflow computations, it is able to significantly outperform competing ML approaches. It is also able to outperform procedures that invest polynomial time in establishing local consistency—such as arc-consistency—to discover unsatisfiable CSP instances. This demonstrates that a trained FastMapSVM model acquires an intelligence beyond that of prominent polynomial-time procedures.[3] We discuss various other representational and combinatorial advantages of FastMapSVM and, through experiments, we also demonstrate its superior performance.

2 Preliminaries and Definitions

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. Binary CSPs are representationally as powerful as general CSPs with bounded arity of the constraints [4].

A binary CSP is *arc-consistent* if and only if for all 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 is *path-consistent* if and only if for all variables

[3] This is an important hallmark of an ML algorithm. In [20], a deep NN model is presented to recognize the satisfiability of CSP instances with Boolean variables and binary constraints. However, this class of CSP instances is equivalent to 2-SAT and can be solved in polynomial time, diminishing the advantages of an ML framework over polynomial-time reasoning.



(a) the "cosine law" projection in a triangle

(b) projection onto a hyperplane that is perpendicular to $\overline{O_aO_b}$

Figure 2 The figure, borrowed from [3], illustrates how coordinates are computed and recursion is carried out in FastMap.

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.

For a given 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 5 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 assuming 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 3 variables X_1, X_2, and X_3 can be represented as a "mega-matrix" with 3 sets of rows and 3 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.

3 FastMap and FastMapSVM

In this section, we describe FastMap and FastMapSVM to set up the groundwork for our approach. Both these rely on a domain-specific distance function $D(\cdot,\cdot)$ on pairs of objects. $D(\cdot,\cdot)$ is required to be a non-negative symmetric function.

3.1 FastMap

FastMap [5] is a Data Mining algorithm that embeds complex objects—like audio signals, seismograms, DNA sequences, electrocardiograms, or magnetic-resonance images—into a K-dimensional Euclidean space, for a user-specified value of K and a user-supplied distance function $D(\cdot, \cdot)$ on pairs of objects. The Euclidean distance between any two objects in the embedding approximates the domain-specific distance between them. Therefore, similar objects, as quantified by $D(\cdot, \cdot)$, map to nearby points in Euclidean space while dissimilar objects map to distant points. Although FastMap preserves $O(N^2)$ pairwise distances between N objects, it generates the embedding in only $O(KN)$ time. Because of its efficiency, FastMap has already found numerous real-world applications, including in Data Mining [5], shortest-path computations [3], solving combinatorial optimization problems on graphs [12], and community detection and block modeling [11].

FastMap embeds a collection of complex objects in an artificially created Euclidean space that enables geometric interpretations, algebraic manipulations, and downstream ML algorithms. It gets as input a collection of complex objects $\mathcal{O}$, where $D(O_i, O_j)$ represents the domain-specific distance between objects $O_i, O_j \in \mathcal{O}$. It generates a Euclidean embedding that assigns a K-dimensional point $\mathbf{p}_i \in \mathbb{R}^K$ to each object O_i. A good Euclidean embedding is one in which the Euclidean distance $\|\mathbf{p}_j - \mathbf{p}_i\|_2$ between any two points $\mathbf{p}_i$ and $\mathbf{p}_j$ closely approximates $D(O_i, O_j)$.

In the first iteration, FastMap heuristically identifies the farthest pair of objects O_a and O_b in linear time. Once O_a and O_b are determined, every other object O_i defines a triangle with sides of lengths $d_{ai} = D(O_a, O_i)$, $d_{ab} = D(O_a, O_b)$, and $d_{ib} = D(O_i, O_b)$, as illustrated in Figure 2a. The lengths of the sides of the triangle define its entire geometry, and the projection of O_i onto the line $\overline{O_a O_b}$ is given by

$$x_i = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2)/(2d_{ab}). \tag{1}$$

FastMap sets the first coordinate of $\mathbf{p}_i$, the embedding of O_i, equal to x_i. In the subsequent $K - 1$ iterations, the same procedure is followed for computing the remaining $K - 1$ coordinates of each object; however, the distance function is adapted for each iteration. For example, for the first iteration, the coordinates of O_a and O_b are 0 and d_{ab}, respectively. Because these coordinates fully explain the true distance between these two objects, from the second iteration onward, the rest of $\mathbf{p}_a$ and $\mathbf{p}_b$'s coordinates should be identical. Intuitively, this means that the second iteration should mimic the first one on a hyperplane that is perpendicular to the line $\overline{O_a O_b}$. Figure 2b illustrates this. Although the hyperplane is never explicitly constructed, it conceptually implies that the distance function for the second iteration should be changed for all i and j in the following way:

$$D_{new}(O_i', O_j')^2 = D(O_i, O_j)^2 - (x_i - x_j)^2, \tag{2}$$

in which O_i' and O_j' are the projections of O_i and O_j, respectively, onto this hyperplane, and $D_{new}(\cdot, \cdot)$ is the new distance function. The distance function is recursively updated according to Equation 2 at the beginning of each of the $K - 1$ iterations that follow the first.

In each of the K iterations, FastMap heuristically finds the farthest pair of objects according to the distance function defined for that iteration. These objects are called pivots and are stored as reference objects. There are very few, that is, $\leq 2K$, reference objects. Technically, finding the farthest pair of objects in any iteration takes $O(N^2)$ time. However, FastMap uses a linear-time "pivot changing" heuristic [5] to efficiently and effectively identify a pair of objects O_a and O_b that is very often the farthest pair. It does this by initially

choosing a random object O_b and then choosing O_a to be the farthest object away from O_b. It then reassigns O_b to be the farthest object away from O_a, reassigns O_a to be the farthest object away from O_b, and so on, until convergence or a maximum of C iterations, for a small constant $C \leq 10$.

3.2 FastMapSVM

FastMapSVM [19] elegantly combines the strengths of FastMap and SVMs. In the first phase, it invokes FastMap for efficiently mapping complex objects to points in a Euclidean space, while preserving pairwise distances between them. 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. There are many real-world domains in which feature selection for *individual* objects is hard. While a domain expert can occasionally identify and incorporate domain-specific features of the objects to be classified, doing so becomes increasingly hard with increasing complexity of the objects. Therefore, many existing ML algorithms for classification find it hard to leverage domain knowledge when used off the shelf. However, 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 the features of individual objects. FastMapSVM also enables domain experts to incorporate their domain knowledge via a distance function instead of relying on complex ML models to infer the underlying structure in the data entirely. Examples of such real-world objects include audio signals, seismograms, DNA sequences, electrocardiograms, and magnetic-resonance images. While these objects are complex and may have many subtle features that are hard to recognize, there exists a well-defined distance function on pairs of objects that is easy to compute. For instance, individual DNA sequences have many complex and subtle features but the *edit distance*[4] between two DNA sequences is well defined and easy to compute. Similarly, the Minkowski distance [1] is well defined for images and the cosine similarity [16] is well defined for text documents.

Second, FastMapSVM facilitates interpretability, explainability, and visualization. Many existing ML algorithms produce results that are hard to interpret or explain. For example, in NNs, a large number of interactions between neurons with nonlinearities makes a meaningful interpretation or explanation of the results very hard. In fact, the very complexity of the objects in the domain can hinder interpretability and explainability. FastMapSVM mitigates these challenges and thereby supports interpretability and explainability. While the objects themselves may be complex, FastMapSVM embeds them in a Euclidean space by considering only the distance function defined on pairs of objects. In effect, it simplifies the description 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. This aids human interpretation of the data and results. It also enables a human-in-the-loop framework for refining the processes of learning and decision making. As a hallmark, FastMapSVM produces the visualization very efficiently since it invests only linear time in generating the Euclidean embedding.

[4] The edit distance between two strings is the minimum number of insertions, deletions, or substitutions that are needed to transform one to the other.

Third, FastMapSVM uses significantly smaller amounts of time and data for model training compared to other ML algorithms. While NNs and other ML algorithms store abstract representations of the training data in their model parameters, FastMapSVM stores explicit references to some of the original objects, referred to as pivots. 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. Moreover, given N training instances, that is, N objects and their classification labels, FastMapSVM leverages $O(N^2)$ pieces of information via the distance function that is defined on every pair of objects. In contrast, ML algorithms that focus on individual objects leverage only $O(N)$ pieces of information.

Fourth, FastMapSVM extends the applicability of SVMs and kernel methods to complex objects. Generally speaking, SVMs are particularly good for classification tasks. When combined with kernel methods, they recognize and represent complex nonlinear classification boundaries very elegantly [17]. Moreover, soft-margin SVMs with kernel methods [14] can be used to recognize both outliers and inherent nonlinearities in the data. While the SVM machinery is very effective, it requires the objects in the classification task 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. In such cases, deep NNs have gained more popularity compared to SVMs for the reason that it is unwieldy for SVMs to represent all the features of complex objects in Euclidean space. However, FastMapSVM revives the SVM approach by leveraging a distance function and creating a low-dimensional Euclidean embedding of the complex objects.

4 Distance Function on CSPs

In this section, we describe a distance function on binary CSPs. This distance function is based on maxflow computations and is illustrated in Figure 3. It is well defined for CSP instances $\mathcal{I}_1$ and $\mathcal{I}_2$ that may have 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 number of variables in $\mathcal{I}_1$ does not match the number of variables in $\mathcal{I}_2$, dummy variables are added to the CSP instance with fewer variables. Figure 3 (top panel) illustrates this for $\mathcal{I}_1$ and $\mathcal{I}_2$ with variables $\{X_1, X_2, X_3, X_4\}$ 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 3 (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".

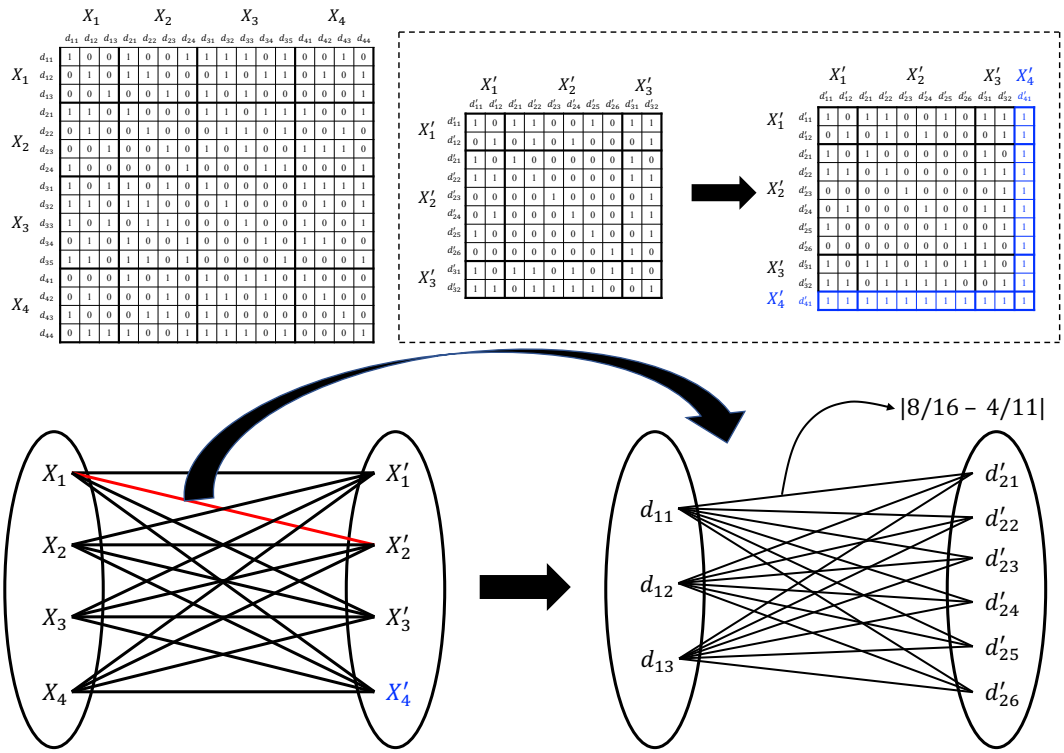


Figure 3 The top panel shows two CSP instances with variables $\{X_1, X_2, X_3, X_4\}$ (left) and $\{X'_1, X'_2, X'_3\}$ (middle), respectively. A dummy variable X'_4 with a singleton domain is added to the CSP instance with fewer variables (right). The bottom panel (left) shows how a "maximum matching 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}.

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 3 (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 "maximum matching of minimum cost" problems in the high level and the low level are posed on bipartite graphs. Since the costs annotating the edges of the bipartite graphs in the high level and the low level are non-negative, the distance function is also non-negative.

Moreover, since the two partitions of any bipartite graph can be viewed interchangeably without affecting the "maximum matching of minimum cost", the overall distance function is symmetric. It is also easy to observe that the distance between two identical CSP instances is always 0. These properties of the distance function satisfy all the requirements imposed on it by the FastMap component of FastMapSVM.

The high-level bipartite graph is invariant to the orderings on the elements within each partition. That is, it is invariant to the orderings on the variables of $\mathcal{I}_1$ and $\mathcal{I}_2$. Similarly, all low-level bipartite graphs are invariant to the orderings on the domain values of the participating variables. Therefore, the overall distance function is invariant to variable-orderings as well as domain value-orderings. This allows us to bypass data augmentation methods typically required for training other ML models.[5] In the context of CSPs, a CSP instance is typically augmented by changing the ordering on its variables or the ordering on the domain values of individual variables. However, doing so generates an exponential number of CSP training instances within the same equivalence class. This drawback of traditional ML algorithms of having to learn equivalence classes is now intelligently addressed within the framework of FastMapSVM by utilizing a distance function that is invariant to both variable-orderings and domain value-orderings.

We note that the above distance function could have been defined in many other ways. For example, we could have introduced dummy domain values in the low-level "maximum matching of minimum cost" problems to equalize the domain sizes of the participating variables. We could have also chosen not to use dummy variables in the high-level "maximum matching of minimum cost" problem. In addition, we could have defined the costs annotating the edges of the bipartite graphs using many other characteristics of the CSPs. These variations of the distance function are not of fundamental importance to this paper. Instead, in this paper, we focus on the advantages of the FastMapSVM framework as a whole. The study of more refined distance functions is delegated to future work.

5 Experimental Results

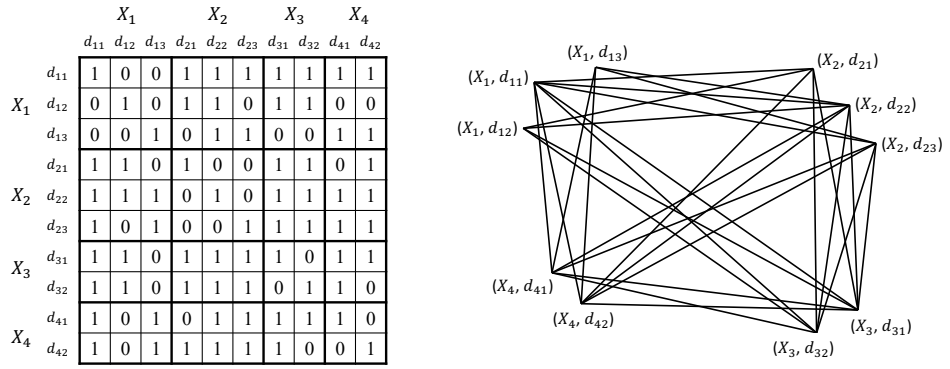
In this section, we describe the comparative performance of FastMapSVM against other state-of-the-art ML approaches on predicting CSP satisfiability.

5.1 Experimental Setup

We evaluate FastMapSVM against three competing approaches. The first is a state-of-the-art deep graph convolutional neural network (DGCNN) [23]. The second is a state-of-the-art graph isomorphism network (GIN) [21]. Both these networks ingest a CSP instance in the form of a graph, as shown in Figure 4. In the graphical representation of a binary CSP instance, a vertex represents a domain value and is tagged with the name of the variable that it belongs to. Information in these tags is utilized by the DGCNN and the GIN. An edge between two vertices v_1 and v_2 with tags X_i and X_j, respectively, represents the compatible combination $(X_i \leftarrow v_1, X_j \leftarrow v_2)$ allowed by the direct constraint between X_i and X_j.[6] DGCNN and GIN do not require the CSP training and test instances to be of the same size.

The third is a polynomial-time algorithm based on establishing arc-consistency. This algorithm first establishes arc-consistency and then checks whether any variable's domain is annihilated. If so, it declares the CSP instance to be "unsatisfiable". Otherwise, it declares

[5] Data augmentation refers to transformations of data without changing their labels, known as label-preserving transformations. For example, to generate more training data serving object recognition tasks in computer vision applications, an image can be augmented by translating it or reflecting it



		X_1			X_2			X_3		X_4	
		d_{11}	d_{12}	d_{13}	d_{21}	d_{22}	d_{23}	d_{31}	d_{32}	d_{41}	d_{42}
X_1	d_{11}	1	0	0	1	1	1	1	1	1	1
	d_{12}	0	1	0	1	1	0	1	1	0	0
	d_{13}	0	0	1	0	1	1	0	0	1	1
X_2	d_{21}	1	1	0	1	0	0	1	1	0	1
	d_{22}	1	1	1	0	1	0	1	1	1	1
	d_{23}	1	0	1	0	0	1	1	1	1	1
X_3	d_{31}	1	1	0	1	1	1	1	0	1	1
	d_{32}	1	1	0	1	1	1	0	1	1	0
X_4	d_{41}	1	0	1	0	1	1	1	1	1	0
	d_{42}	1	0	1	1	1	1	1	0	0	1

Figure 4 The left panel shows the matrix representation of a binary CSP instance. The right panel shows its graphical representation. The vertices represent domain values and are clustered into four groups, corresponding to the four variables $\{X_1, X_2, X_3, X_4\}$.

the CSP instance to be "satisfiable". This algorithm is used in our evaluation to demonstrate that FastMapSVM's capabilities go beyond that of a polynomial-time algorithm.[7] Of course, a polynomial-time algorithm based on establishing path-consistency could also have been used. But we excluded this algorithm since arc-consistency already provides the required proof of concept and establishing path-consistency is prohibitively expensive.

In our experiments, we do not include results from the CSP-cNN framework of [20] for the following reasons. First, this framework has the drawback that it is applicable only to CSP training and test instances of the same size. (FastMapSVM does not have this drawback since the distance function does not require the CSP instances to be of the same size.) Second, the success of CSP-cNN critically depends on data augmentation methods: For each CSP training instance, a very large number of other training instances that permute the variables and their domain values also have to be generated. (This requirement is completely obviated by FastMapSVM since the distance function is invariant to such permutations.) Third, CSP-cNN has been shown to be successful only on Boolean binary CSP instances, that is, the polynomial-time solvable 2-SAT problems. (FastMapSVM does not have this limitation; below, we demonstrate its success on general binary CSP instances.)

We implemented FastMapSVM and arc-consistency in Python3 and ran them on a laptop with an Apple M2 chip with 16 GB memory. We ran DGCNN and GIN on a Linux system with an Intel(R) Xeon(R) Silver 4216 CPU at 2.10 GHz. The different platforms are inconsequential to the comparative performances of these algorithms with respect to effectiveness. For each dataset, we trained DGCNN and GIN for 100 epochs with a learning rate of 0.0001 and a minibatch size of 100 to obtain representative results.

5.2 Instance Generation

We generate the binary CSP instances for both training and testing using the Model A method in [18, 20]. We generate a CSP instance by first picking the number of variables N uniformly at random to be an integer within the range $[1, 100]$. Then, we pick the domain

horizontally without changing its label [10].

[6] The graphical representation of a binary CSP instance is obtained by parsing its matrix representation. Thus, we correctly represent the compatible tuples of domain values between every pair of variables, even if there does not exist a direct constraint between those variables.

[7] This is done to avoid the pitfalls of [20], as mentioned before.

size of each variable independently and uniformly at random to be an integer within the range $[1, 10]$. We use a probability parameter P_1 to independently determine the existence of a direct constraint between each pair of distinct variables. That is, for each pair of distinct variables X_i and X_j, we introduce a direct constraint between them with probability P_1. Moreover, we use a probability parameter P_2 to determine the compatible tuples of a direct constraint. For a pair of variables X_i and X_j with a direct constraint between them, each tuple $(X_i \leftarrow d_{ip}, X_j \leftarrow d_{jq})$ is independently deemed to be compatible with probability $1 - P_2$. We set $P_1 = 1$ and $P_2 = 0.4$ to obtain representative results for all approaches.

Model A has a tendency to produce mostly unsatisfiable CSP instances with increasing N [18, 20]. Therefore, we use a "hidden solution" method to generate satisfiable CSP instances whenever required. In this method, a set of hidden solutions of the CSP instance is chosen a priori.[8] A hidden solution $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$ is utilized as follows: While generating the direct constraints using Model A, a direct constraint between variables X_i and X_j reserves the tuple $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jp_j})$ as being compatible before the other tuples are set using the probability parameter P_2. Therefore, $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$ satisfies all the direct constraints and, consequently, qualifies as a solution. Similarly, multiple hidden solutions can be utilized with the following modification in the generation procedure: A direct constraint between variables X_i and X_j reserves multiple tuples as being compatible. For generating satisfiable CSP instances, we pick the number of hidden solutions uniformly at random to be an integer within the range $[1, 10]$. We pick a hidden solution itself by assigning a domain value chosen independently and uniformly at random for each variable from its domain.

We generate three datasets: Dataset-1, Dataset-2, and Dataset-3. For each dataset, we generate 1000 training instances and 1000 test instances. Each training and test set has an equal number of satisfiable and unsatisfiable instances.

In Dataset-1, we generate the instances using Model A. Since Model A frequently generates unsatisfiable instances, we use a complete CSP solver to identify and collect such instances. We generate the satisfiable instances using the hidden solution method, as described above.

In Dataset-2, we generate the satisfiable instances as in Dataset-1. However, we design and generate the unsatisfiable instances to be more challenging. We do this by hiding two complementary pseudo-solutions $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$ and $(X_1 \leftarrow d_{1q_1}, X_2 \leftarrow d_{2q_2} \dots X_N \leftarrow d_{Nq_N})$. We identify a pair of distinct variables X_i and X_j such that $d_{ip_i} \neq d_{iq_i}$ and $d_{jp_j} \neq d_{jq_j}$. All direct constraints between distinct variables X_s and X_t such that $\{X_s, X_t\} \neq \{X_i, X_j\}$ are generated as before by reserving the tuples $(X_s \leftarrow d_{sp_s}, X_t \leftarrow d_{tp_t})$ and $(X_s \leftarrow d_{sq_s}, X_t \leftarrow d_{tq_t})$ as being compatible. However, the direct constraint between X_i and X_j reserves the tuples $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jq_j})$ and $(X_i \leftarrow d_{iq_i}, X_j \leftarrow d_{jp_j})$ as being compatible and reserves the tuples $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jp_j})$ and $(X_i \leftarrow d_{iq_i}, X_j \leftarrow d_{jq_j})$ as being not compatible. We finally use a complete CSP solver to verify that the CSP instance is indeed unsatisfiable.[9]

In Dataset-3, we generate the satisfiable instances as in Dataset-1. However, we design and generate the unsatisfiable instances differently from in Dataset-2. We do this by first hiding two complementary pseudo-solutions $(X_1 \leftarrow d_{1p_1}, X_2 \leftarrow d_{2p_2} \dots X_N \leftarrow d_{Np_N})$ and $(X_1 \leftarrow d_{1q_1}, X_2 \leftarrow d_{2q_2} \dots X_N \leftarrow d_{Nq_N})$, as in Dataset-2. However, we gather all variables $X_{r_1}, X_{r_2} \dots X_{r_{\tilde{M}}}$ for which the two pseudo-solutions have different assignments

[8] The CSP instance can have other solutions as well.

[9] This procedure frequently generates unsatisfiable instances, as required. However, satisfiable instances that are generated occasionally are filtered out by the CSP solver.

of domain values, that is, $d_{r_m p_{r_m}} \neq d_{r_m q_{r_m}}$, for all $1 \leq m \leq \bar{M}$. For any two distinct variables X_i and X_j in $\{X_{r_1}, X_{r_2} \ldots X_{r_{\bar{M}}}\}$, we reserve the tuples $(X_i \leftarrow d_{ip_i}, X_j \leftarrow d_{jq_j})$ and $(X_i \leftarrow d_{iq_i}, X_j \leftarrow d_{jp_j})$ as being not compatible. Finally, we pick two distinct variables X_s and X_t from $\{X_{r_1}, X_{r_2} \ldots X_{r_{\bar{M}}}\}$ and overwrite the tuples $(X_s \leftarrow d_{sp_s}, X_t \leftarrow d_{tq_t})$ and $(X_s \leftarrow d_{sq_s}, X_t \leftarrow d_{tp_t})$ as being compatible and reserve the tuples $(X_s \leftarrow d_{sp_s}, X_t \leftarrow d_{tp_t})$ and $(X_s \leftarrow d_{sq_s}, X_t \leftarrow d_{tq_t})$ as being not compatible. As before, we use a complete CSP solver to verify that the CSP instance is indeed unsatisfiable.

5.3 Results

We show three sets of results pertaining to FastMapSVM. First, we show the 2-dimensional and the 3-dimensional embeddings that FastMapSVM produces to aid visualization. Second, we show the behavior of FastMapSVM with respect to the hyperparameter K, that is, the number of dimensions and with respect to the size of the training data. Third, we show the comparative performance of FastMapSVM against DGCNN, GIN, and arc-consistency.

FastMapSVM used the SVM classifier from the scikit-learn library. Its hyperparameter settings were determined by grid search. For Dataset-1 and Dataset-3, the hyperparameters were regularization parameter = 8, kernel = "rbf", and kernel coefficient = "scale". For Dataset-2, the hyperparameters were regularization parameter = 8, kernel = "poly", and kernel coefficient = "scale".

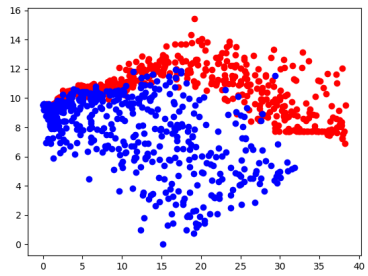
Figure 5 shows a perspicuous visualization of the CSP test instances for all three datasets. This visualization capability is unique to FastMapSVM. We note that while the accuracy, recall, precision, and the F1 score of FastMapSVM typically increase with increasing K, $K = 2$ and $K = 3$ are the only two values that support visualization. Still, in most cases, Figure 5 shows a clear separation between the satisfiable and unsatisfiable instances. Moreover, the separation is clearer in the 3-dimensional embeddings compared to their 2-dimensional counterparts.

Figure 6a shows the behavior of FastMapSVM with respect to the number of dimensions K on Dataset-1. Its behavior on the other datasets is similar. The performance metrics, that is, the accuracy, recall, precision, and the F1 score, improve with increasing K. This is intuitively expected since the distances between the CSP instances can be embedded with lower distortion in higher dimensions. However, Figure 6a also shows that a point of diminishing returns is attained rather quickly at around $K = 8$. This shows that $K = 8$, 9, or 10 is good enough for the CSP domain. Finally, Figure 6a also shows that the improvements in the performance metrics are significant between $K = 2$ and $K = 8$.

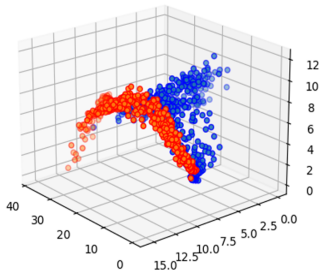
Figure 6b shows the behavior of FastMapSVM with respect to the size of the training data on Dataset-1. Its behavior on the other datasets is similar. The performance metrics improve with increasing size of the training data. Figure 6b also shows that the improvements in the performance metrics are significant between 128 and 256 training data instances. Further improvements are gradual between 256 and 1000 training data instances. This shows that FastMapSVM has the capability to achieve good performance from relatively small amounts of training data and training time.

Table 1 shows a comparison of all the competing methods on all three datasets with respect to all of the performance metrics. It uses $K = 8$ for FastMapSVM. It also shows two versions of DGCNN and GIN: the "labeled" version and the "unlabeled" version.

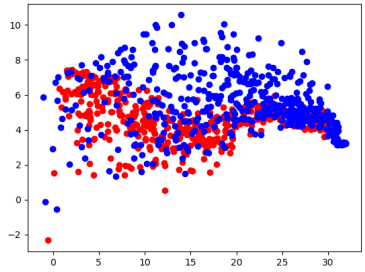
We recollect that in the graphical representation of a binary CSP instance, a vertex represents a domain value and is tagged with the name of the variable that it belongs to. Information in these tags is available to be utilized by the DGCNN and the GIN. The labeled versions of DGCNN and GIN utilize this information while the unlabeled versions ignore this



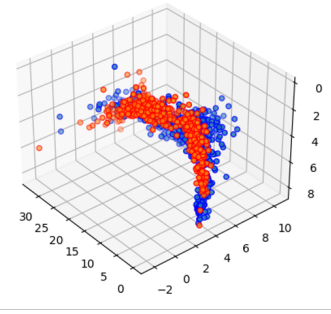
(a) Dataset-1 CSP instances embedded in a 2-dimensional Euclidean space by FastMapSVM



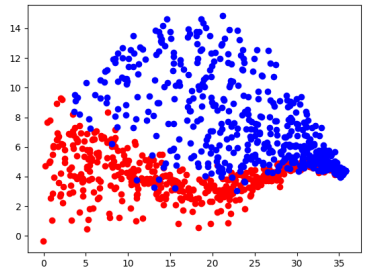
(b) Dataset-1 CSP instances embedded in a 3-dimensional Euclidean space by FastMapSVM



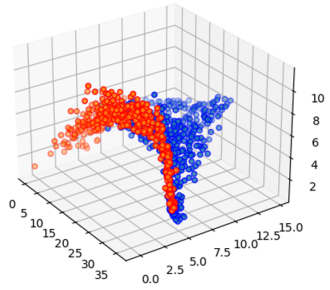
(c) Dataset-2 CSP instances embedded in a 2-dimensional Euclidean space by FastMapSVM



(d) Dataset-2 CSP instances embedded in a 3-dimensional Euclidean space by FastMapSVM



(e) Dataset-3 CSP instances embedded in a 2-dimensional Euclidean space by FastMapSVM



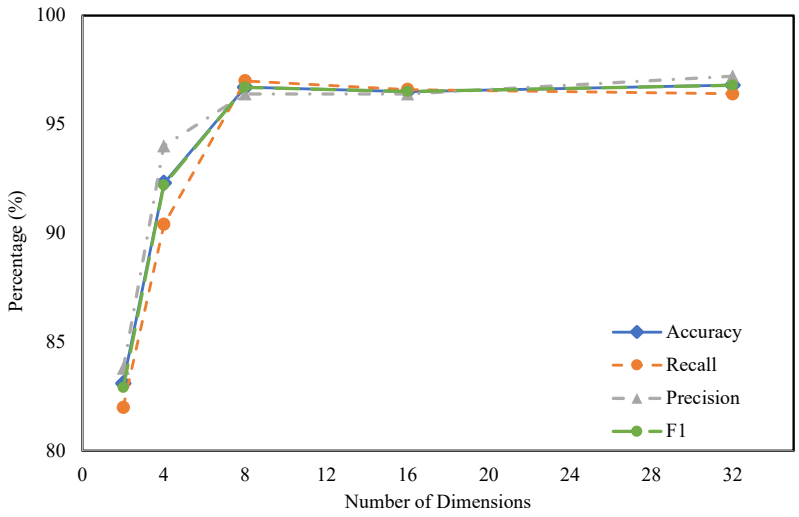
(f) Dataset-3 CSP instances embedded in a 3-dimensional Euclidean space by FastMapSVM

Figure 5 The figure shows the low-dimensional Euclidean embeddings produced by FastMapSVM for classifying CSP instances. Mostly, there is a clear separation of satisfiable instances (blue) and unsatisfiable instances (red).

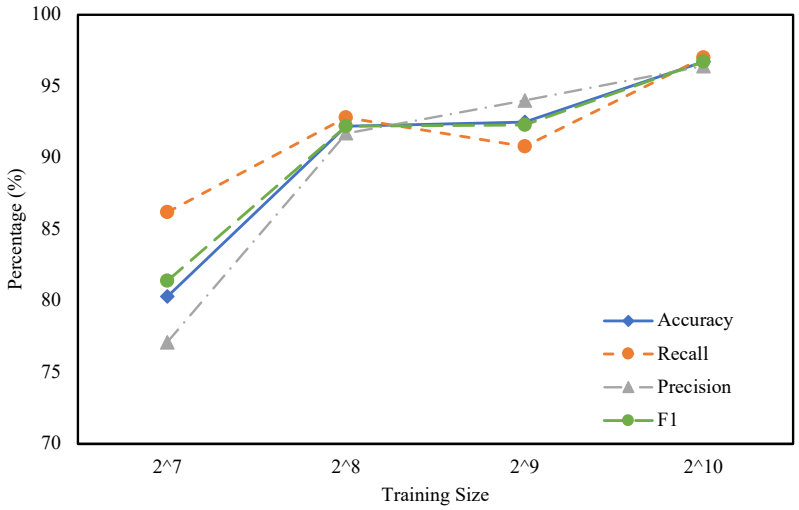
information. Table 1 shows that the unlabeled versions perform better than their labeled counterparts. While this is a little surprising, it is likely that the unlabeled versions indirectly perform permutation reasoning on the tags (names of variables) much more efficiently.

Table 1 shows that FastMapSVM generally outperforms all other competing methods by a significant margin. Even on a particular dataset where it is not the top performer with respect to a particular performance metric, it is a close second. Overall, Dataset-1 seems to be the easiest for all methods and Dataset-2 seems to be the hardest for all methods.

In comparison to arc-consistency, FastMapSVM is significantly better on Dataset-2 and Dataset-3. On these datasets, arc-consistency declares all test instances as being "satisfiable", leading to a perfect recall score but very poor precision, accuracy, and F1 scores. On the one



(a) Influence of the number of dimensions on the performance of FastMapSVM



(b) Influence of the size of the training data on the performance of FastMapSVM

Figure 6 The figure shows the behavior of FastMapSVM with respect to the number of dimensions and with respect to the size of the training data. The performance metrics include the accuracy, recall, precision, and the F1 score.

hand, this shows that arc-consistency is ineffective in recognizing unsatisfiable CSP instances. On the other hand, it also shows that CSP instances generated as in Dataset-1 are insufficient to conclusively evaluate competing ML methods. In contrast, FastMapSVM performs well on all three datasets.

In comparison to DGCNN and GIN, FastMapSVM is significantly better on all three datasets. On the accuracy, recall, and F1 scores, FastMapSVM is better than DGCNN, which in turn is better than GIN. GIN generally has high precision scores but very poor recall scores. This shows that it is poor in identifying satisfiable instances but is mostly correct

Table 1 The table shows all performance metrics for all competing methods on all datasets. "AC" represents arc-consistency.

Dataset	Model	Accuracy	Recall	Precision	F1
Dataset-1	FastMapSVM	96.7%	97.0%	96.4%	96.7%
	AC	**99.3%**	**100.0%**	96.7%	**98.3%**
	DGCNN (unlabeled)	94.2%	92.2%	96.0%	94.1%
	DGCNN (labeled)	82.3%	82.0%	82.5%	82.2%
	GIN (unlabeled)	84.1%	67.0%	**99.4%**	80.0%
	GIN (labeled)	56.4%	52.6%	56.9%	54.7%
Dataset-2	FastMapSVM	**82.9%**	72.8%	**91.2%**	**81.0%**
	AC	50.1%	**100.0%**	50.1%	66.8%
	DGCNN (unlabeled)	73.4%	61.4%	80.8%	69.8%
	DGCNN (labeled)	53.9%	51.2%	54.1%	52.6%
	GIN (unlabeled)	71.9%	49.0%	90.4%	63.6%
	GIN (labeled)	54.4%	51.6%	54.7%	53.1%
Dataset-3	FastMapSVM	**95.4%**	94.4%	96.3%	**95.3%**
	AC	50.0%	**100.0%**	50.0%	66.7%
	DGCNN (unlabeled)	90.3%	86.6%	93.5%	89.9%
	DGCNN (labeled)	74.7%	71.8%	76.2%	73.9%
	GIN (unlabeled)	78.4%	53.6%	**98.5%**	69.4%
	GIN (labeled)	57.4%	53.8%	58.0%	55.8%

when it does so. FastMapSVM does not have this drawback. Moreover, on the accuracy and F1 scores, FastMapSVM outperforms the closest competitor (DGCNN) by larger margins with increasing hardness of the CSP instances, that is, in the order of Dataset-1, Dataset-3, and Dataset-2. Even on the metric of efficiency, FastMapSVM outperforms DGCNN and GIN.[10]

6 Conclusions and Future Work

In this paper, we introduced a novel ML framework, called FastMapSVM, for the task of predicting CSP satisfiability. FastMapSVM overcomes the hurdles faced by other ML approaches in the CSP domain. It leverages a distance function on CSPs that is defined via maxflow computations. FastMapSVM is applicable to CSP training and test instances of different sizes and is invariant to both variable-orderings and domain value-orderings. This allows it to bypass the onus of having to learn equivalence classes of CSP instances and, therefore, requires significantly smaller amounts of time and data for model training compared to other ML algorithms. FastMapSVM also uses the intelligence of SVMs, kernel methods, and maxflow computations, accounting for its superior empirical performance, even over state-of-the-art graph neural networks. Moreover, it facilitates a perspicuous visualization of the CSP instances, their distribution, and the classification boundaries between them. Overall, the FastMapSVM framework for CSPs has broader applicability and various representational and combinatorial advantages compared to other ML approaches.

There are many avenues for future work. These include the design of better distance functions on CSPs, the application of FastMapSVM to optimization variants of CSPs, and the general facilitation of integrating constraint reasoning and ML methods via FastMapSVM.

[10] DGCNN and GIN ran on a different platform compared to FastMapSVM. However, the ballpark results are still conclusive.

References

1 Abder-Rahman Ali, Micael S Couceiro, Aboul Ella Hassanien, and D Jude Hemanth. Fuzzy c-means based on Minkowski distance for liver CT image segmentation. *Intelligent Decision Technologies*, 2016.

2 Alejandro Arbelaez, Youssef Hamadi, and Michele Sebag. Continuous search in constraint programming. In *Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence*, 2010.

3 Liron Cohen, Tansel Uras, Shiva Jahangiri, Aliyah Arunasalam, Sven Koenig, and T. K. Satish Kumar. The FastMap algorithm for shortest path computations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018.

4 Rina Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

5 Christos Faloutsos and King-Ip 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.

6 Ian Philip Gent, Christopher Anthony Jefferson, Lars Kotthoff, Ian James Miguel, Neil Charles Armour Moore, Peter Nightingale, and Karen Petrie. Learning when to use lazy learning in constraint solving. In *Proceedings of the 19th European Conference on Artificial Intelligence*, 2010.

7 Alessio Guerri and Michela Milano. Learning techniques for automatic algorithm portfolio selection. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 2004.

8 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC–instance-specific algorithm configuration. In *Proceedings of the 19th European Conference on Artificial Intelligence*, 2010.

9 Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, 2016.

10 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 2017.

11 Ang Li, Peter Stuckey, Sven Koenig, and T. K. Satish 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.

12 Jiaoyang Li, Ariel Felner, Sven Koenig, and T. K. Satish Kumar. Using FastMap to solve graph problems in a Euclidean space. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2019.

13 Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry 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.

14 Arti Patle and Deepak Singh Chouhan. SVM kernel functions for classification. In *Proceedings of the International Conference on Advances in Technology and Engineering*, 2013.

15 Luca Pulina and Armando Tacchella. A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, 2007.

16 Faisal Rahutomo, Teruaki Kitasuka, and Masayoshi Aritsugi. Semantic cosine similarity. In *Proceedings of the International Student Conference on Advanced Science and Technology*, 2012.

17 V David Sánchez A. Advanced support vector machines and kernel methods. *Neurocomputing*, 2003.

18 Barbara M Smith and Martin E Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 1996.

19 Malcolm White, Kushal Sharma, Ang Li, T. K. Satish Kumar, and Nori Nakata. FastMapSVM: Classifying complex objects using the FastMap algorithm and support-vector machines. *arXiv preprint arXiv:2204.05112*, 2022.

20 Hong Xu, Sven Koenig, and T. K. Satish Kumar. Towards effective deep learning for constraint satisfaction problems. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming*, 2018.

21 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

22 Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 2008.

23 Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.