Edit Distance Kernelization of NP Theorem Proving For Polynomial-Time Machine Learning of Proof Heuristics

David Windridge[1] and Florian Kammüller[1]

Middlesex University London, UK
{d.windridge|f.kammueller}@mdx.ac.uk

Abstract. We outline a general strategy for the application of edit-distance based kernels to NP Theorem Proving in order to allow for polynomial-time machine learning of proof heuristics without the loss of sequential structural information associated with conventional feature-based machine learning. We provide a general short introduction to logic and proof considering a few important complexity results to set the scene and highlight the relevance of our findings.

1 Logic and Proof – Syntax, Semantics, and Proof Systems

Theorem proving is an attempt to provide machine support for logic and proof. We need logic for problem solving; if we have very small sets of states, we can potentially solve problems just by enumerating all possibilities. Logic, however, provides a convenient way of dealing with larger (potentially infinite) sets of states via the manipulation of compact sentential descriptions instead of large sets of states. These manipulations are made possible by providing *syntax*, *semantics* and a *proof system* for a logic. The syntax defines what constitute legal sentences, the semantics says what they mean, and the proof system allows to syntactically change logic expressions to provide new insights. The proofs allow us to make conclusions about the world in a given state from given percepts and also to conclude properties of the next state given additional state operators (formulas usually contain variables that can be generalized to more complex states). The properties of such a logic description then depends on their interpretation, i.e., the values these variables have. The semantics of a logic formula is usually one of the truth values *true* or *false*, but it may depend on the interpretation of variables contained in the formula. We say a formula is *valid* if it evaluates to *true* for all possible interpretations; we say it is *satisfiable* if there is an interpretation such that it becomes true for inserting these values into the variables, and it is called *unsatisfiable* if there is no interpretation that makes the formula *true*.

Decision Problems Satisfiability (SAT) solver and Satisfiability Modulo Theory (SMT) solvers look at satisfiability of logic formulas; the latter one with

respect to an additional axiom system (also called *theory*). Satisfiability problems occur frequently everywhere in the form of constraint systems, for example, scheduling problems. In general, we are often interested in *validity*. That is, we want to know the correspondence of entailment between a set of formulas Γ and a formula ϕ. Entailment is written as

$$\Gamma \models \phi\,.$$

This statement means that for every interpretation of the variables in Γ the formula ϕ is also *true*. In other words, there is a subset relation between interpretations between the formula ϕ and the set of formulas Γ: $I_\phi \subseteq I_\Gamma$. This, however, is equivalent to the statement that the logical implication between them is *valid*:

$$\models \Gamma \longrightarrow \phi$$

i.e., this implication is true for all interpretations of variables.

Proof Systems Proof is a way of determining validity without examining all interpretations which satisfy a formula. Proofs are commonly written in a notation similar to entailment. The formula

$$\Gamma \vdash \phi$$

means ϕ can be proved from Γ for a given proof system associated with $\vdash$.

A proof system is a set of so called *inference rules* and a way to apply those to sets of formulas Γ that allows to transform statements, that is, *infer* a statement from a previous one. We say that a proof system is *correct* (or sound) iff

$$\Gamma \vdash \phi \implies \Gamma \models \phi$$

and we say that it is *complete* iff

$$\Gamma \models \phi \implies \Gamma \vdash \phi\,.$$

Natural Deduction There are different styles of proof systems. The proof system of *Natural Deduction* uses a style of proof that suits human understanding. For example, the rule of *modus ponens* allows to infer Q from the set of premises $\{P, P \longrightarrow Q\}$ written as

$$P \qquad P \longrightarrow Q \vdash Q\,.$$

This could also be written in a two-dimensional style thus creating a tree like structure for proofs that visualises the proof structure, i.e., chaining up of rules. For example, consider the proof of $P \wedge Q \longrightarrow Q \wedge P$.

$$\dfrac{\dfrac{\dfrac{[A \wedge B]}{B}[conjE_{\text{left}}] \qquad \dfrac{[A \wedge B]}{A}[conjE_{\text{right}}]}{B \wedge A}[conjI]}{A \wedge B \longrightarrow B \wedge A}[impI]$$

In each step, the items above the lines are the premises specified by the inference rule. The rules name is given in square brackets at the side, for example, $conjE_{\text{left}}$ for the rule $P \wedge Q \vdash Q$. The last step of the proof uses the rule $impI$: $[P] \vdash Q \vdash P \longrightarrow Q$ to eliminate the assumption $A \wedge B$ by adding it as a premise to the current formula. This elimination of premises is called cancellation syntactically indicated by the square brackets around the cancelled assumption. Note that it is possible to cancel various occurrences of the premise at once and that the cancellation may reach across several proof steps as shown in the example.

Resolution Refutation A proof system that is different in style but more suitable for the implementation of automated proving on computers is that of resolution refutation [12]. It uses the resolution rule

$$P \vee Q \quad Q \longrightarrow R \vdash P \vee R$$

which can also be written as

$$P \vee Q \quad \neg Q \vee R \vdash P \vee R\,.$$

Resolution refutation works as follows. Given a set of assumptions Γ and a conclusion ϕ, we prove $\vdash \Gamma \longrightarrow \phi$ by the following procedure.

1. Transform each premise $\gamma \in \Gamma$ into Conjunctive Normal Form (CNF), (i.e., a conjunction of disjunctions, e.g. $(P \vee Q \vee \neg R) \wedge (\neg P \vee R)$), obtaining Γ' as the set of conjuncts.
2. Conjoin the negated conclusion $\neg\phi$ to Γ' transformed into CNF.
3. Apply repeatedly the resolution rule to Γ' extending Γ' by adding the conclusion.

This procedure either terminates by reaching a contradiction in Γ' by having P and $\neg P$ in the set, or by reaching a Γ' such that the resolution rule cannot be applied any more to extend Γ'. In the former case, we have proved

$$\Gamma \quad \neg\phi \vdash \mathit{false}$$

which is equivalent to having proved $\Gamma \vdash \phi$ in classical logics. In the latter case, we have shown that we could not prove ϕ using the resolution refutation procedure. For propositional logic, refutation resolution is a complete procedure. Hence, we know that in the latter case if ϕ cannot be proved from Γ using refutation resolution, then it is also not entailed in Γ.

Since transformation of a set of formulas Γ into CNF is a simple algorithm and applying just the one resolution refutation rule defines the whole process of resolution, it seems intuitively clear how it could be implemented as a decision procedure on a computer.

We illustrate the resolution refutation procedure on the example that we have used for natural deduction. To prove

$$\vdash A \wedge B \longrightarrow B \wedge A$$

we apply Steps 1) and 2) negating this conclusion ϕ and adding it to the empty set of premises.

$$\neg(A \wedge B \longrightarrow B \wedge A)\,.$$

Next, we transform this (set of) premises into CNF by translation the implication and applying de Morgan's laws.

$$\neg(\neg(A \wedge B) \vee (B \wedge A)) \equiv$$

$$(\neg\neg(A \wedge B)) \wedge \neg(B \wedge A) \equiv$$

$$A \wedge B \wedge (\neg B \vee \neg A).$$

We then apply Step 3), i.e., the resolution rule once and add the conclusion.

$$A \wedge B \wedge (\neg B \vee \neg A) \wedge B \wedge \neg B.$$

Finally, we have the contradiction $B \wedge \neg B$ in the set of conjuncts and have thus proved ϕ.

Decidability of Proof Systems As we have seen above, resolution refutation is a complete and sound procedure for propositional logic. Gödel's Completeness Theorem proves that there exists a complete proof system for First Order Logic (FOL) but it is only later that Robinson showed a more constructive version proving that resolution refutation is a complete proof system for FOL. This resolution refutation for FOL is more complicated than the one for propositional logic as it has to deal with quantifiers. As a consequence, provability, i.e. $\Gamma \vdash \phi$, is only semi-decidable for FOL. That is, if we apply resolution refutation to a formula ϕ and a set of premises Γ such that ϕ is actually entailed by ϕ, then it will arrive at a contradiction for Γ and $\neg\phi$, i.e. find the proof. But if ϕ is not entailed in Γ, the resolution refutation might not terminate. By contrast, resolution refutation terminates for propositional logic in both cases, i.e., provability for propositional logic is decidable.

If we consider more expressive logics than FOL, we have to consider Gödel's **In**completeness Theorem: there is no complete and consistent (sound) proof system for FOL if Arithmetic is added. So for any potential proof system there would be either a true statement it cannot prove (incomplete) or it would be possible to prove a false statement (inconsistent). Intuitively, the proof works by assuming an arbitrary proof system for FOL with Arithmetic, then showing that one of the two is the case. Arithmetic provides the means to construct code names for sentences in the logic (using a procedure called "Gödel-numbering") and therefore construct sentences that are self-referential. The sentence used in the proof is sometimes called the Gödel-sentence $G =$ "G is not provable". Now, if G would be true then it is not provable by definition and therefore the proof system would be incomplete. If G, however, would be false, then G is provable which means the proof system is not sound (inconsistent).

Having no complete and sound proof system for FOL with Arithmetic implies that there is also no complete and sound procedure to implement automated

proofs on computers. Consequently, any logics that can formalise Arithmetic cannot be adequately automated. This includes all Higher Order Logics. Yet, theorem provers for Higher Order Logics exist, e.g., Coq or Isabelle, but they are interactive, that is, need human intervention. And even with human intervention, the proof systems they implement must be incomplete (if not inconsistent as one would hope). Surprisingly, Gödel's Incompleteness Theorem itself could be proved in Isabelle [10]. So, these logics are not so incomplete as one might think although the proof document containing the transcript of the interactions with Isabelle in the Archive of Formal proof has more than 200 pages [9]. Also, incompleteness is still a great loss since it says that the proof system will never reveal all truth. However, the main motivation for interactive proof in HOL is to provide a means to have sound proofs. Although traditionally mathematical proof is a social process performed by humans in a community of scientists there are cases where the human capacity for guaranteeing the consistency of proofs is reached and where formalisation and proof in interactive theorem provers is necessary to provide acceptable proofs. Examples are the proof of the Four Colour Theorem in Coq [6] and the proof of Kepler's conjecture in HOL-light [7].

Computational Complexity of Propositional Logic Proof Systems If we want to investigate the feasibility of theorem proving, we need to describe the computational complexity of decision procedures for (provability of) logical statements precisely. This only makes sense for decidable questions since for undecidable or semi-decidable proof procedures we could only ever compare efficiency relative to some varying portion of the problem. Therefore, complexity theory focuses only on propositional logic because even for FOL we already have no decision procedure: as we have seen above, the complete and sound procedure of resolution for FOL is only semi-decidable.

It appears from the literature that the complexity of propositional proofs is a well studied subject and still many questions remain open. There are some very remarkable relations to general complexity problems.

Let Σ be a finite alphabet and Σ^* denote the set of all finite strings over it. A language L is defined as a subset of Σ^*, that is, a set of strings. The length of a string s is written as $|s|$. Let us first recall the basic definition of computational complexity theory. We adopt the definitions of [13, 4] adapting them slightly. A set of strings L is in the class P (NP) if it is recognized in time polynomial in the length of the input by a deterministic (non-deterministic) Turing machine. A set of strings L is in the class co-NP if L is the complement $\Sigma^* \setminus \hat{L}$ of a language $\hat{L}$ that is in NP. The following set L defines characteristic functions for the elements of class P and thus makes the notion of "polynomial-time function" more precise.

Definition 1 (Polynomial function class $\mathcal{L}$). *A function $f : \Sigma_1^* \to \Sigma_2^*$ for finite alphabets Σ_1, Σ_2 is in $\mathcal{L}$, if it can be computed by a deterministic Turing machine in time bounded by a polynomial in the length of the input.*

To compare the efficiency of propositional proof systems, we need a general yet abstract definition of proof system to encompass the existing variety of existing systems.

Definition 2 (Abstract Proof System). *Let $L \subseteq \Sigma^*$ be a language. A proof system for L is a function $\mathcal{P} : \hat{\Sigma}^* \to L$ for some alphabet $\hat{\Sigma}$ such that $\mathcal{P}$ is in L and onto. A proof system $\mathcal{P}$ is polynomially bounded if there is a polynomial p such that for all $A \in L$ there is a $\pi \in \hat{\Sigma}^*$ such that $\mathcal{P}(\pi) = A$ and $|A| \leq |p(\pi)|$.*

The language L will be the set of all valid propositional formulas. The idea of this definition is that $\mathcal{P}(\pi) = A$ if π is a proof for A in the proof system $\mathcal{P}$. The special property of a polynomially-bounded proof system as defined above is that there is a feasible function P that checks whether a potential proof π of a property A is a proof in that proof system and that it is a proof of property A.

To illustrate this on our running example, we encode the resolution proof of $B \wedge A \longrightarrow A \wedge B$ in one string

$$[A \wedge B \wedge (\neg B \vee \neg A), B \wedge \neg B, false] .$$

The function $\mathcal{P}_{\text{res}}$ implementing a proof system for resolution refutation checks that this string represents a sequence of resolution refutation steps according to the algorithm sketched above. In addition, it re-transforms $A \wedge B \wedge (\neg B \vee \neg A)$ into $\neg(A \wedge B \longrightarrow B \wedge A)$ and by deleting again the negation to check that it is a proof of the property $A \wedge B \longrightarrow B \wedge A$. These steps can be clearly done in time polynomial to the length of the input.

A problem p is called NP-complete, if it is at least as hard (complex) than any other NP problem. That is, if every other problem in NP can be transformed (or reduced) into p in polynomial time. If a polynomial solution to any NP-complete problem would be found, all other NP problems would also follow to be in P thereby proving P = NP. The idea of completeness transfers also to the class coNP. Simply, every coNP-complete problem is the complement of an NP-complete problem.

A celebrated result by Cook and Reckhow [2] shows that *SAT solving for propositional logic is NP complete.*

From this theorem follows a result on propositional logic [4].

Corollary 1. *Validity for propositional logic is coNP-complete.*

This corollary immediately implies another one [4].

Corollary 2. *Validity for propositional logic is in P if and only if P = NP.*

This is relevant for the implications of the work we are going to present next.

2 Machine Learning Theorem Proving

Theorem proving/formal verification is, for the most part (i.e. the majority of axiom systems deployed in the field), NP complete. Any polynomial-time heuristic system for reducing the search space involved in proving a given theorem that

performs better than chance can therefore be effectively utilized for increasing the efficiency of theorem-proving/formal-verification.

Such heuristics are generally provided manually. However, an alternative is to learn these via the techniques of *machine learning*. The typical machine learning paradigm is *supervised classification* in which discretely-labeled (i.e. class-delineated) data of arbitrary kinds are represented in a feature space of potentially very large dimensionality, within which an optimal class decision boundary is determined via an appropriate optimization process (such that arbitrary unlabeled data can be attributed to one of the training classes).

Syntactically-valid sentences in a formal language may thus seem superficially well suited a machine learning approach in that they consist of a finite set of labeled data, for which the attribution of a label is a difficult, potentially non-polynomial process (the labels in question being the True/False values of the theorem prover [or appropriate alternative discrete truth values]).

However, the set of syntactically-valid sentences does not naturally lend itself to a feature-based model. In particular, the potentially infinite variation of sentence length would suggest that sentences have an intrinsically arbitrary feature-dimensionality, and are thus not collectively embedded in the *same* feature space. (While it is possible to enforce a consistent dimensionality amongst sentences by, for example, using a normalized symbol histogram ('bag of words') approach [14], this would invariably constitute an information losing process).

In this paper we propose a 'featureless' polynomial-time approach to the learning of proof heuristics that eliminates feature space representation entirely and works directly on the individual sentences as given; the classification process thus takes place in an entirely implicit feature-space. This implicit feature space will turn out to have interesting characteristics; in particular it will provide a continuous Hilbert (or Krein) space in which the theorems exist. As well as providing an intriguing implicit discrete-to-continuous space mapping, continuousness provides a range of useful properties for efficient optimization.

Our principle contribution is thus the outlining of strategy for NP-complete-$\rightarrow$polynomial heuristic mapping for formal-verification problems.

2.1 Adopted Machine Learning Paradigm

Support Vector Machine (SVMs) are archetypal *binary classifiers*, i.e. entities capable of learning an optimal discriminative decision hyperplane from labeled vectors $\{(\mathbf{x}, y) \mid \mathbf{x} \in \tilde{X},\ y \in \{-1, +1\}\}$ existing within a feature space $\tilde{X}$.

SVMs are especially useful in classical machine learning because they have the capacity to be *kernelized*; that is, the gram matrix inherent in the (dual form of the) SVM optimization problem can be replaced by an arbitrary kernel function obeying the Mercer condition (essentially positive semi-definiteness -see later), enormously extending its capability.

Kernel functions thus constitute a form of similarity measure (specifically, a highly generalized inner product) between classification objects. Indeed they can be demonstrated to be the equivalent of inner products within an implicit embedding space (the Mercer space) generated by the kernel as a *feature mapping*

of classification objects (which need not be directly computed in itself). This is enormously powerful in machine learning in that it enables classification to apply in areas in which there is not a readily apparent real vector space of feature measures. A relevant motivating example is genomics, for which it is much more straightforward to compute a similarity measure between pairs of DNA strands (using e.g. least common ancestor distance or mutation distance) than it is to embed each strand individually into a vector space of feature measurements. However, computation of kernels can be a complex, potentially NP-hard, exercise; we will thus, in the following section set out the specifics of SVM learning computation in order to show how it applies to the theorem-proving problem.

3 Methodology

In order to specify our strategy for kernel machine learning within a theorem proving context, we shall firstly give a general description of the support vector machine optimzation problem:

3.1 The SVM and its Kernelization

The standard SVM [3] seeks to maximize the margin (*i.e.*, the distance of the decision hyperplane to the nearest data point), subject to a constraint on the classification accuracy of the labeling induced by the hyperplane's delineation of a general decision boundary. In its primal form, the soft margin SVM optimization takes the form of a Lagrange optimization problem:

$$\arg\min_{(\mathbf{w},b)} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{M} \xi_i \right\} \tag{1}$$

subject to:

$$\forall_i \; y_i(\mathbf{w}\cdot\mathbf{x_i} - b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{2}$$

where $(\mathbf{x_i}, y_i)$ $i = 1 \ldots M$ are the training vectors/labels, $y_i \in \{-1, +1\}$, $\mathbf{w}$ is the weight orientation vector of the decision hyperplane, and b is its bias offset. (The margin is inversely proportional to $\|\mathbf{w}\|$). The ξ_i are slack variables that give rise to the soft margin with sensitivity controlled by hyper-parameter C.

In the dual form [3], the slack parameters disappear such that the problem is solved in terms of the Karush–Kuhn–Tucker (KKT) multipliers α_i:

$$\arg\max_{(\alpha_i)} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j (\mathbf{x}_i^T\mathbf{x}_j) \tag{3}$$

subject to:

$$\sum \alpha_i y_i = 0 : \quad \forall_i \; 0 \leq \alpha_i \leq C \tag{4}$$

The problem is one of quadratic programming. As the optimization proceeds, only a sparse set of the α_ss retain non-zero values. These denote the support vectors defining the decision hyperplane. This sparsity (i.e. the low parametric complexity of the decision boundary with respect to the training data) gives the SVM substantial resilience to over-fitting (and thus reduces classifier variance).

Notably, it may be shown that the term $(\mathbf{x}_i^T \mathbf{x}_j)$ in the above (equating to the training vector Gram matrix) may be freely replaced by any kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ that satisfies the Mercer condition (guaranteeing positive semi-definiteness). This is the principle (but by no means the only) use of kernel methods in machine learning, one which vastly extends the utility of the SVM by enabling the mapping of the input decision space into a large variety of alternative Hilbert spaces of potentially infinite dimensionality (thus guaranteeing linear separability). The decision boundary in the input space may thus undergo significant morphology variation while crucially retaining the low parametric support vector characterization of the decision boundary in the Mercer embedding space (the space defined by $\boldsymbol{\phi}(\mathbf{x})$, where $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \boldsymbol{\phi}(\mathbf{x_i})^T(\boldsymbol{\phi}(\mathbf{x_j})))$. Critically, at no stage are we required to compute $\boldsymbol{\phi}(\mathbf{x_i})$. The Mercer condition guarantees the existence of $\boldsymbol{\phi}$, but the kernel itself may be calculated based on any similarity function that gives rise to a legitimate (ie PSD) kernel matrix. (It is also possible to formulate the SVM problem in the explicit absence of similarity functions encompassing a guarantee of positive semi definiteness via the use of *Krein spaces*. However, in this case the SVM optimization problem is no longer strictly convex and requires an alternative (though often straightforward) saddle-point gradient descent process without guarantees of achieving the global minimum).

3.2 Edit Distance Kernels

In the terminology of Neuhaus & Bunke [8], a string t over V consists in an ordered, finite sequence of symbols drawn from V such that:

$$t = t_1 ... t_n \in V^\star = \bigcup_{i=0}^{\inf} V_i \quad \text{with } V_0 = \{\} \ \& \ n > 0 \tag{5}$$

where V_i is the set of strings of length i over V and $V^\star$ is the set of all finite sequences of symbols drawn from V. V is thus typically a finite set of symbols, but may equally represent vector spaces etc. This format can thus capture a wide range of sequential data within machine leaning, from written text to DNA sequences.

The standardized set of string edit operations that can be performed on such a string consists in:

1) An insertion operation $\{\} \rightarrow q$ (inserting symbol q into a string)
2) A deletion operation $p \rightarrow \{\}$ (removing symbol p from a string)
3) A substitution operation $p \rightarrow q$ (exchanging symbol p in a string with symbol q).

Clearly, by recursive operation, any string can be transformed into any other string using just these operations. For example, to utilize our running example, we may obtain the final proof string "$[A \wedge B \wedge (\neg B \vee \neg A), B \wedge \neg B, false]$" via a series of transformations of the initial string "$A \wedge B \wedge (\neg B \vee \neg A)$" (which in the proof theoretic context corresponds to the application of the resolution refutation rule). Further type-dependency can be introduced into the above rules while retaining its character as an edit distance e.g. by including rules disallowing the deletion of negation operators.

In order to arrive at a kernel, however, it is necessary to obtain a *unique* measure of the edits required to relate arbitrary pairs of strings. Following [8], if we let $e(t, t')$ denote the set of all edit operation sequences that connect t to t' , then the *string edit distance*, $d(t, t')$, between the two strings is defined as the minimum cost required to edit t into t':

$$d(t, t') = \underset{(w1,\ldots,wk)\in e(t,t')}{\arg\min} \sum_{i=1}^{k} c(w_i) \tag{6}$$

where c is the positive real-valued *edit cost function.* The fact that this distance is parametrized means that, we in effect, have a *family* of distance measures (and later kernels).

Note, critically, that this is a true metric obeying the triangle inequality. Recursively evaluating this distance, however, takes exponential time; it is consequently computed within feasible machine learning scenarios via a dynamic programming approach that reduces the cost to approximately quadratic time (though provably not strongly subquadratic time [1]).

By contrast, the *graph edit distance* between discrete attributed graphs can be computed in essentially the same way as the general edit distance, with the exception that the nodes of a graph (whose connectivity may be represented via a matrix) have no intrinsic order to them, meaning that dynamic programming does not readily apply [5]). Thus (to extend the above terminology), the node substitution $u \rightarrow v$, replaces node u in a graph with node v; an edge insertion $\{\} \rightarrow (p, q)$ on the other hand, inserts the edge connecting node p with node q into the graph.

Graph edit distance is hence similarly NP-complete in its native formulation to the general edit distance, however only non-upper-bounded heuristic processes are available to render it tractable [11].

It is thus the fact that theorems are one-dimensional strings of symbols that we propose to exploit; any reasonable (i.e. monotonic) variant of the string edit distance will retain its polynomial-time characterization in the theorem-proving domain (following the application of dynamic programming); thus any non-polynomial proof-theoretic problem can, in principle, be substituted by a polynomial-time Kernel matrix formulation problem (in conjunction with a polynomial $O(n^3 log(n))$ SVM optimization problem); the attribution of a truth value to an arbitrary sentence is essentially linear once the learning process has taken place.

Applying these notions within a machine learning context thus involves construction of a kernel function between two string objects x and x' based on their edit distance with respect to a fixed pattern string x_0:

$$k(x, x') = k_{x_0}(x, x') = \frac{1}{2}(d(x, x_0)^2 + d(x_0, x')^2 - d(x, x')^2) \tag{7}$$

In particular, we can guarantee the positive semidefiniteness of this kernel by virtue of the metricality of d, which thus obeys the Mercer condition.

Neuhaus & Bunke [8] go on to construct *sum* and *product* kernels with respect to the full set of fixed pattern strings I in the training set:

$$k_I^+(x, x') = \sum_{x_0 \in I} k_{x_0}(x, x) \tag{8}$$

$$k_I^\star(x, x') = \prod_{x_0 \in I} k_{x_0}(x, x) \tag{9}$$

which both retain the properties of a kernel. (Indeed any arbitrary convex sum or product over kernels is also a kernel and we can thus treat the determination of the optimal kernel coefficients as a polynomial-time optimization problem in its own right, for instance when we are confronted with a family of kernels as in the parametrized edit distances above).

Having thus obtained a kernel suitable for application to theorem proving, such that *any* logical sentence is guaranteed to exist within the Mercer space of the kernel, it becomes possible to apply machine learning (and, in particular, the SVM algorithm) in order to anticipate (i.e. predict) the outcome of the theorem prover in question on the basis of it previous outcomes (i.e. via an implicit labeling of previously derived sentences using the binary class labels *theorem/non-theorem*). Moreover, it does so in an adaptive, on-line manner such that the mechanism improves over time. It can thus be deployed alongside the theorem prover in a hybridized form in order to determine promising directions of application of the full machinery of theorem proving.

It is thus clear that although computing an edit-distance kernel constitutes an optimization problem in its own right (the problem being NP-hard in its recursive form without optimization, and irretrievably NP-hard in the case of graph edit distance kernels), the specific case applicable to theorem proving is always tractable in polynomial time.

Thus, provided that the domain exhibits learnability (which only requires that the learning domain is not fully topologically discontinuous), machine learning has the capability to provide a polynomial substitution for a non-polynomial theorem-proving problem (with a utility that is proportional to domain learnability).

4 Conclusions

We have provided a concise introduction to the theorem proving problem with respect to the issues of decidability and computational complexity before pro-

ceeding to demonstrate that *kernelized machine learning* is capable of providing a hybridized approach to reducing the theorem proving problem to a polynomial one with probabilistic success dependent on the intrinsic learnability of the domain. Critically, domain learnability is related to the metric proximity of similar data; a domain thus only exhibits non-learnability when its class labels are *entirely* discontinuous. There is no reason, however, to suppose that this extreme case applies in relation to theorem proving with sequential strings.

In terms of the state of the art on the theorem proving problem, more precisely Corollary 2, it may appear that we are thus claiming $P = NP$ in contradiction to the current conviction in the scientific community that $P \neq NP$. Far from trying to do so, we believe that our results presented in this paper are fully valid given that what we have presented is *heuristic* learning. Therefore, findings in relation to machine learning theorem proving (which is the subject of ongoing experimental work) need to be understood as approximate behaviours true in the average case.

Acknowledgment

The first author would like to acknowledge financial support from the Horizon 2020 European Research project DREAMS4CARS (number 731593).

Bibliography

[1] Backurs, A., Indyk, P.: Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). CoRR abs/1412.0348 (2014), http://arxiv.org/abs/1412.0348

[2] Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. J. Symb. Log. 44(1), 36–50 (1979), https://doi.org/10.2307/2273702

[3] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)

[4] Das, A.: The complexity of propositional proofs in deep inference. Ph.D. thesis, University of Bath Department of Computer Science (2014)

[5] Fischer, A., Uchida, S., Frinken, V., Riesen, K., Bunke, H.: Improving hausdorff edit distance using structural node context. In: International Workshop on Graph-Based Representations in Pattern Recognition. pp. 148–157. Springer (2015)

[6] Gonthier, G.: Formal proof - the four color theorem. Notices of the American Mathematical Society 55(11), 1382–1393 (2008)

[7] Hales, T.C., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., Rute, J.M., Solovyev, A., Ta, A.H.T., Tran, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the kepler conjecture. CoRR abs/1501.02155 (2015), http://arxiv.org/abs/1501.02155

[8] Neuhaus, M., Bunke, H.: Edit distance-based kernel functions for structural pattern classification. Pattern Recognition 39(10), 1852–1863 (Oct 2006), http://www.sciencedirect.com/science/article/B6V14-4K48N7S-4/2/1e7743302ffe0f0662da24f14c7d5a8f

[9] Paulson, L.C.: Gödel's incompleteness theorems. Archive of Formal Proofs (nov 2013), http://isa-afp.org/entries/Incompleteness.html, Formal proof development

[10] Paulson, L.C.: A mechanised proof of gödel's incompleteness theorems using nominal isabelle. J. Autom. Reasoning 55(1), 1–37 (2015), https://doi.org/10.1007/s10817-015-9322-8

[11] Riesen, K., Fankhauser, S., Bunke, H.: Speeding up graph edit distance computation with a bipartite heuristic.

[12] Robinson, J.A.: A machine oriented logic based on the resolution principle. J.ACM 12(1), 23–41 (1965)

[13] Urquhart, A.: The complexity of propositional proofs. Bulletin of Symbolic Logic 1, 425–467 (1995)

[14] Wallach, H.M.: Topic modeling: beyond bag-of-words. In: Proceedings of the 23rd international conference on Machine learning. pp. 977–984. ACM (2006)