Compiling Combinatorial Prediction Games

Frederic Koriche [1]

Abstract

In online optimization, the goal is to iteratively choose solutions from a decision space, so as to minimize the average cost over time. As long as this decision space is described by combinatorial constraints, the problem is generally intractable. In this paper, we consider the paradigm of compiling the set of combinatorial constraints into a deterministic and Decomposable Negation Normal Form (`dDNNF`) circuit, for which the tasks of linear optimization and solution sampling take linear time. Based on this framework, we provide efficient characterizations of existing combinatorial prediction strategies, with a particular attention to mirror descent techniques. These strategies are compared on several real-world benchmarks for which the set of Boolean constraints is preliminarily compiled into a `dDNNF` circuit.

1. Introduction

Combinatorial optimization is a important topic of computer science and discrete mathematics, with a wide spectrum of applications ranging from resource allocation and job scheduling, to automated planning and configuration softwares. A common problem is to minimize a modular loss function $\boldsymbol{\ell}$ over a discrete space $\mathcal{S} \subseteq \{0,1\}^d$ of feasible solutions represented in a concise manner by a set of combinatorial constraints. In the *offline* version of this problem, all information necessary to define the optimization task is available beforehand, and the challenge is to develop algorithms which are provably or practically better than enumerating all feasible solutions. Contrastingly, in *online* version of this problem (Audibert et al., 2014), the objective function $\boldsymbol{\ell}$ is subject to change over time. The challenge here is more acute, since the optimization algorithm is required to perform repeated choices on $\mathcal{S}$ so as to minimize the average cost of those choices in the long run.

*Equal contribution [1]CRIL, CNRS UMR 8188, Université d'Artois, France. Correspondence to: Frederic Koriche <koriche@cril.fr>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

Conceptually, an online combinatorial optimization problem can be cast as a repeated prediction game between a learning algorithm and its environment (Audibert et al., 2011; 2014). During each trial t, the learner chooses a feasible solution $\boldsymbol{s}_t$ from its decision set $\mathcal{S}$ and, simultaneously, the environment selects a loss vector $\boldsymbol{\ell}_t \in [0,1]^d$. Then, the learner incurs the loss $\langle \boldsymbol{\ell}_t, \boldsymbol{s}_t \rangle = \sum_{i=1}^d \ell_t(i) s_t(i)$ and, in light of the feedback provided by its environment, updates its strategy in order to improve the chance of selecting better solutions on subsequent trials.

Several classes of combinatorial prediction games can be distinguished, depending on the type of decision set, and the type of observed feedback. In this paper, we focus on *full information* games in which it is assumed that the feedback supplied at trial t by the environment is the entire vector $\boldsymbol{\ell}_t$. On the other hand, we make very few assumptions about the decision set: $\mathcal{S}$ may be described by an arbitrary `SAT` formula, that is, any set of combinatorial constraints representable by Boolean clauses. As `SAT` encodings of discrete solution spaces are frequently used in academic and industrial applications (Biere et al., 2009), our setting covers an important class of combinatorial prediction games.

As usual, the performance of an online learning algorithm is measured according to two metrics. The first, called *regret*, measures the difference in cumulative loss between the algorithm and the best solution in hindsight. In this study, we make no assumption about the sequence of loss vectors; in particular $\boldsymbol{\ell}_t$ may depend on the previous decisions $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_{t-1}$ made by the learner. In such *non-oblivious* or *adversarial* environments, the learner is generally allowed to make decisions in a randomized way, and its predictive performance is measured by the *expected regret*:

$$R_T = \mathbb{E}\left[\sum_{t=1}^{T} \langle \boldsymbol{\ell}_t, \boldsymbol{s}_t \rangle\right] - \min_{\boldsymbol{s} \in \mathcal{S}} \sum_{t=1}^{T} \langle \boldsymbol{\ell}_t, \boldsymbol{s} \rangle$$

The second metric is *computational complexity*, i.e. the amount of resources required to compute $\boldsymbol{s}_t$ at each round t, given the sequence of feedbacks observed so far.

Related Work. In the literature of combinatorial prediction games, three main strategies have been proposed to attain an expected regret that is sublinear in the game horizon T and polynomial in the input dimension d. The first, and arguably simplest strategy, is to *Follow the Perturbed Leader* (FPL): on each trial t, the learner draws at random a perturbation vector $\boldsymbol{z}_t \in \mathbb{R}^d$, and then selects in $\mathcal{S}$ a minimizer of $\eta \boldsymbol{L}_t + \boldsymbol{z}_t$, where $\eta \in (0, 1]$ is a step-size parameter, and $\boldsymbol{L}_t$ is the cumulative loss $\boldsymbol{L}_t = \boldsymbol{\ell}_1 + \cdots + \boldsymbol{\ell}_{t-1}$. Based on the pioneering work of Hannan (1957), refined in (Hutter & Poland, 2005; Kalai & Vempala, 2005), the FPL algorithm achieves an expected regret of $\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$.

The second strategy is based on the popular exponentially weighted average forecaster in the framework of prediction with expert advice (Cesa-Bianchi & Lugosi, 2006). The overall idea is to maintain a weight for each feasible solution $\boldsymbol{s} \in \mathcal{S}$, which decays exponentially according to the estimated cumulative loss of $\boldsymbol{s}$. Specifically, on each trial t, the learner draws a solution $\boldsymbol{s}_t \in \mathcal{S}$ at random from the exponential family $p_t(\boldsymbol{s}) \sim \exp(-\eta \langle \boldsymbol{L}_t, \boldsymbol{s} \rangle)$. This strategy, referred to as *Expanded Hedge* (EH) in (Koolen et al., 2010), attains an expected regret of $\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$.

Finally, the third strategy is to *Follow the Regularized Leader*, a paradigm often advocated in online convex optimization (Hazan, 2016). Here, the learner operates on the convex hull of $\mathcal{S}$, denoted $\mathrm{conv}(\mathcal{S})$. On each trial t, the learner starts by choosing a point $\boldsymbol{p}_t \in \mathrm{conv}(\mathcal{S})$ that minimizes $\eta \langle \hat{\boldsymbol{L}}_t, \boldsymbol{p} \rangle + F(\boldsymbol{p})$, where F is a regularization function. Next, $\boldsymbol{p}_t$ is decomposed as a convex composition of feasible solutions in $\mathcal{S}$, and then, a decision $\boldsymbol{s}_t$ is picked at random according to the resulting distribution. For modular loss functions, this strategy is equivalent to the *Online Stochastic Mirror Descent* (OSMD) algorithm (Audibert et al., 2014; Rajkumar & Agarwal, 2014), which iteratively performs a gradient descent in the dual space of $\mathrm{conv}(\mathcal{S})$ under F, and projects back to the primal space according to the Bregman divergence defined from F. Notably, when F is the Euclidean regularizer, OSMD coincides with the popular *stochastic gradient descent* (SGD) algorithm (Robbins & Monro, 1951). Alternatively, when F is the entropic regularizer, OSMD corresponds to the *Component Hedge* (CH) algorithm (Koolen et al., 2010), which achieves an *optimal* expected regret of $\mathcal{O}(d\sqrt{T})$.

From the viewpoint of regret, the results outlined above indicate that few improvements remain to be made in full information games. However, we get a different picture if computational considerations are taken into account: all aforementioned algorithms rely on powerful oracles for making decisions in spaces $\mathcal{S}$ represented by combinatorial constraints. Namely, the EH algorithm is required, at each iteration, to sample a solution according to an exponential family over $\mathcal{S}$, a problem which is generally #P-hard (Dyer et al., 2009). Similarly, the FPL strategy has to repeatedly solve a linear optimization task over $\mathcal{S}$, which is generally NP-hard (Creignou et al., 2001). For the OSMD algorithm, and its specializations SGD and CH, the computational issue is exacerbated by the fact that, even if the learner has access to a linear optimization oracle, it still has to perform, at each trial, a Bregman projection step for which the best known algorithms run in $\mathcal{O}(d^6)$ time (Suehiro et al., 2012).

Although combinatorial prediction games are generally intractable, efficient implementations of sampling and optimization oracles may be obtained for several decision sets $\mathcal{S}$. For example, when the feasible solutions in $\mathcal{S}$ coincide with the bases of a binary matroid, or the perfect matchings of a bipartite graph, linear optimization can be performed in polynomial time, and tractable forms of FPL and OSMD may be derived (Helmbold & Warmuth, 2009; Koolen et al., 2010; Takimoto & Hatano, 2013; Rajkumar & Agarwal, 2014). On the other hand, when the feasible solutions in $\mathcal{S}$ correspond to the paths or multi-paths of a rooted Directed Acyclic Graph (DAG), the sampling oracle may be implemented by the *weight pushing* technique (Mohri, 1998), that recursively evaluates the partition function of an exponential family over the edges of the input DAG. Based on this technique, tractable forms of EH can be derived (Takimoto & Warmuth, 2003; Rahmanian & Warmuth, 2017).

Our Results. Viewing feasible solutions as paths in a DAG is only one of many abstractions that have been proposed in the literature of circuit complexity for representing combinatorial spaces. In the related field of knowledge compilation (Darwiche & Marquis, 2002), various classes of Boolean circuits have been identified, each associated with a set of inference tasks which can be performed in polynomial time. These theoretical results naturally motivate the following question: can we *compile* a set of constraints representing a combinatorial space $\mathcal{S}$ into a compact and Boolean circuit for which both solution sampling and linear optimization are tractable? By viewing the compilation process as a "pre-processing step", we may get for free efficient implementations of sampling and optimization oracles, provided that the size of the resulting circuit is not too large.

The present study aims at solving combinatorial prediction games, by compiling decision sets into *deterministic Decomposable Negation Normal Form* (dDNNF) circuits (Darwiche, 2001). This class comes with generic compilers which take as input a SAT formula representing a decision set $\mathcal{S}$, and return a dDNNF circuit C that encodes $\mathcal{S}$ (Darwiche, 2002; Lagniez & Marquis, 2017). Although the size of C may grow exponentially in the treewidth of the input formula, it is usually much smaller in practice; existing compilers are able to compress combinatorial spaces defined over thousands of variables and constraints.

With these compilation tools in hand, our contributions are threefold: (i) we show that for dDNNF circuits, the sampling oracle in EH and the linear optimization oracle in FPL, run in linear time using a simple variant of the weight-pushing technique; (ii) for the SGD and CH strategies, we develop a Bregman projection-decomposition method that uses $\mathcal{O}(d^2 \ln(dT))$ calls to the linear optimization oracle; (iii) we experimentally show on online configuration and planning tasks that EH and FPL are fast, but our variants of SGD and CH are more efficient to minimize the empirical regret.

Before proceeding to the core of the paper, we emphasize that the compilation approach to online optimization is not entirely new. Recently, Sakaue et. al. (2018) used the class of *Ordered Binary Decision Diagrams* (OBDDs) (Bryant, 1986) for implementing the EWA forecaster in combinatorial bandits. Here, $\mathcal{S}$ is described by a graph over d edges, together with a constraint specifying the type of objects we desire (e.g. paths or cliques). By contrast, our study assumes that $\mathcal{S}$ is described with an arbitrary set of Boolean constraints. So, both studies are targeting different classes of combinatorial prediction games. Moreover, it is known that dDNNF is *strictly more succinct* than OBDD (Darwiche & Marquis, 2002). Namely, any OBDD can be transformed in linear time and space into an equivalent dDNNF circuit, but the converse is not true: dDNNF includes simple circuits which require an exponential size representation in OBDD. In fact, the key point of compiling combinatorial prediction games is to use both tractable and succinct languages, for allowing prediction strategies to be efficient on a wide variety of combinatorial domains.

2. Tractable Inference via Compilation

For the combinatorial prediction games considered in this paper, we assume that the input decision space $\mathcal{S}$ is defined from a set of n binary-valued attributes, and we use $X = \{x_1, \cdots, x_d\}$, where $d = 2n$, to denote the set of all "attribute-value" pairs, called *literals*. A *solution* is a vector $\boldsymbol{s} \in \{0,1\}^d$ such that $s(i) + s(j) = 1$ for every pair of distinct literals $x_i, x_j \in X$ defined on the same attribute. Thus, $\|\boldsymbol{s}\|_1 = n$ for any feasible solution $\boldsymbol{s} \in \mathcal{S}$.

An NNF circuit over X is a rooted DAG, whose internal nodes are labeled by $\vee$ (or-node) or $\wedge$ (and-node), and whose leaves are labeled by either a literal in X, or a constant in $\{0, 1\}$. The size of C, denoted $|C|$, is given by the number of its edges. The set of attributes occurring in the subgraph of C rooted at some node c is denoted $att(c)$.

For the sake of clarity, we assume that any NNF circuit C satisfies two basic properties, namely (i) any internal node c in C has exactly two children, denoted c_l and c_r, and (ii) $att(c_l) = att(c_r) \neq \varnothing$ for any or-node c of C. An NNF circuit satisfying both conditions is called *smooth*. As shown in (Darwiche, 2001), any Boolean circuit C can be transformed in to an equivalent smooth NNF circuit of size linear in $|C|$.

By viewing literals as "input gates", and nodes as "output gates", we may specify various inference tasks on Boolean

Q	R	$\oplus$	$\otimes$	$\top$	$\bot$
maxmin	$\{0,1\}$	max	min	1	0
minsum	$\mathbb{R} \cup \{+\infty\}$	min	$+$	0	$+\infty$
sumprod	$\mathbb{R}$	$+$	$*$	1	0

Table 1: Commutative semirings

circuits, depending on the type of input values and the semantics of nodes. As suggested by Friesen & Domingos for sum-product functions (2016), inference tasks can be captured through semiring operations. To this point, recall that a *commutative semiring* is a tuple $(R, \oplus, \otimes, \bot, \top)$ such that R is a set including the elements $\bot$ and $\top$, $\oplus$ is a associative and commutative binary operation on R with identity element $\bot$, $\otimes$ is an associative binary operation on R with identity element $\top$ and absorbing element $\bot$, and the operator $\otimes$ left and right distributes over the operator $\oplus$.

Inference tasks on an NNF circuit C are defined using a commutative semiring $Q = (R, \oplus, \otimes, \bot, \top)$ and an input vector $\boldsymbol{w} \in R^d$. The *output* of a node c in C for Q given $\boldsymbol{w}$ is denoted $Q(c \,|\, \boldsymbol{w})$, and recursively defined by

$$Q(c|w) = \begin{cases} w(i) & \text{if } c \text{ is the literal } x_i, \\ \top & \text{if } c \text{ is the constant } 1, \\ \bot & \text{if } c \text{ is the constant } 0, \\ Q(c_l \,|\, \boldsymbol{w}) \oplus Q(c_r \,|\, \boldsymbol{w}) & \text{if } c \text{ is a node } \vee, \text{ and} \\ Q(c_l \,|\, \boldsymbol{w}) \otimes Q(c_r \,|\, \boldsymbol{w}) & \text{if } c \text{ is a node } \wedge \end{cases}$$

By $Q(C \,|\, \boldsymbol{w})$, we denote the output of the root of C for Q given $\boldsymbol{w}$. Of particular interest in this study are the semirings described in Table 1; maxmin, minsum, and sumprod, and used to capture the inference tasks of model checking, linear optimization, and model sampling, respectively.

2.1. Model Checking

Given an NNF circuit C over X, the task of model checking is to decide whether a Boolean input $\boldsymbol{s} \in \{0,1\}^d$ is true in C according to the propositional semantics of nodes. Obviously, $\boldsymbol{s}$ is a model of C iff $\text{maxmin}(C \mid \boldsymbol{s}) = 1$, which can be determined in $\mathcal{O}(|C|)$ time. An NNF circuit C is called a *representation* of a set of feasible solutions $\mathcal{S} \subseteq \{0,1\}^d$ if $sol(C) = \mathcal{S}$, where $sol(C)$ is the set of models of C.

Apart from model checking, virtually all inference tasks in NNF circuits are NP-hard. Indeed, the NNF language covers the class of SAT formulas. So, we need to refine this class in order to get tractable forms of optimization and sampling.

2.2. Decomposability and Optimization

A Boolean circuit C is *decomposable* if for every and-node c of C, we have $att(c_l) \cap att(c_r) = \varnothing$. The class of decomposable NNF circuits is denoted DNNF. For such circuits, which are similar to Boolean sum-product networks (Poon & Domingos, 2011), we can get an efficient implementation of the linear optimization oracle.

Proposition 1. Let $\mathcal{S} \subseteq \{0,1\}^d$ be a (nonempty) decision set represented by a DNNF circuit C, and let $\boldsymbol{w} \in \mathbb{R}^d$ be a modular objective. Then, finding a minimizer of $\boldsymbol{w}$ in $\mathcal{S}$ can be done in $\mathcal{O}(|C|)$ time.

Proof. Based on the minsum semiring, we have

$$\min_{\boldsymbol{s}\in\mathcal{S}} \langle \boldsymbol{w}, \boldsymbol{s} \rangle = \min_{\boldsymbol{s}\in sol(C)} \langle \boldsymbol{w}, \boldsymbol{s} \rangle = \mathrm{minsum}(C \,|\, \boldsymbol{w})$$

This observation suggests a two-pass weight pushing method for finding a minimizer $\boldsymbol{s}$ of $\boldsymbol{w}$ in $\mathcal{S}$ in $\mathcal{O}(|C|)$ time. Given a topological ordering of C, the first pass stores the value $Q(c \mid \boldsymbol{w})$ of each node $c \in C$, using $Q = \mathrm{minsum}$. The second pass performs a top-down search over C, by selecting all children of a visited and-node, and by selecting exactly one child $c' \in \{c_l, c_r\}$ of a visited or-node c such that $Q(c' \mid \boldsymbol{w}) = Q(c \mid \boldsymbol{w})$. Let T be the corresponding search tree, and let $\boldsymbol{s} \in \{0,1\}^d$ be the indicator vector of the set of literals occurring in T. By construction, we have $Q(T \,|\, \boldsymbol{w}) = Q(C \,|\, \boldsymbol{w})$, which implies that $\boldsymbol{s}$ is a minimizer of $\boldsymbol{w}$. Since $\mathcal{S}$ is not empty, we know that $Q(C \,|\, \boldsymbol{w}) < +\infty$. This, together with the fact that $\mathrm{minmax}(T \mid \boldsymbol{s}) = 1$ whenever $Q(T \,|\, \boldsymbol{w}) < +\infty$, implies that $\boldsymbol{s} \in \mathcal{S}$. ☐

2.3. Determinism and Sampling

As the problem of counting the number of models in a DNNF circuit is #P-hard (Darwiche & Marquis, 2002), we need to refine this class in order to get an efficient implementation of the sampling oracle. To this end, an NNF circuit C is called *deterministic* if $\mathrm{minmax}(c_l \mid \boldsymbol{s}) + \mathrm{minmax}(c_r \mid \boldsymbol{s}) \leq 1$ for every or-node $c \in C$ and every feasible solution $\boldsymbol{s}$. The class of deterministic DNNF circuits is denoted dDNNF.

Proposition 2. Let $\mathcal{S} \subseteq \{0,1\}^d$ be a decision set represented by a dDNNF circuit C, and for a vector $\boldsymbol{w} \in \mathbb{R}^d$, let $\mathbb{P}_{\boldsymbol{w}}$ be the exponential family on $\mathcal{S}$ given by:

$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{s}) = \frac{\exp\langle \boldsymbol{w}, \boldsymbol{s} \rangle}{\sum_{\boldsymbol{s}'\in\mathcal{S}} \exp\langle \boldsymbol{w}, \boldsymbol{s}' \rangle}$$

Then, sampling $\boldsymbol{s} \sim \mathbb{P}_{\boldsymbol{w}}$ can be done in $\mathcal{O}(|C|)$ time.

Proof. Based on the $\mathrm{sumprod}$ semiring, we have

$$\mathbb{P}_{\boldsymbol{w}}(\boldsymbol{s}) = \frac{\exp\langle \boldsymbol{w}, \boldsymbol{s} \rangle}{\sum_{\boldsymbol{s}'\in sol(C)} \exp\langle \boldsymbol{w}, \boldsymbol{s}' \rangle} = \frac{\exp\langle \boldsymbol{w}, \boldsymbol{s} \rangle}{\mathrm{sumprod}(C \,|\, \boldsymbol{w}')}$$

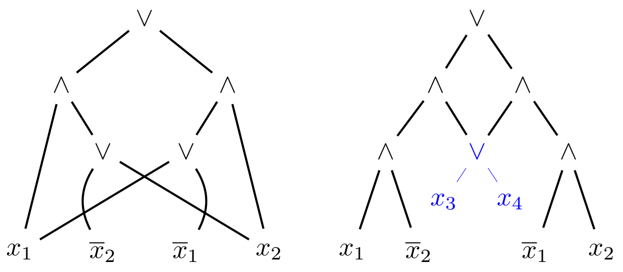

Figure 1: A smoothed DNNF circuit C (left), and a dDNNF circuit C' (right). Here, x_i and $\overline{x}_i$ are distinct literals sharing the same attribute. Note that C' can be smoothed by simply replacing x_3 with $x_3 \wedge (x_4 \vee \overline{x}_4)$, and using a symmetric substitution for x_4.

where $\boldsymbol{w}' = (e^{w(1)}, \cdots, e^{w(d)})$. Again, such an equivalence suggests a two-pass weight pushing method for sampling a solution $\boldsymbol{s}$ according to $\mathbb{P}_{\boldsymbol{w}}$ in $\mathcal{O}(|C|)$ time. Using a topological ordering of C, the first pass stores the values $Q(c \,|\, \boldsymbol{w}')$, where $Q = \mathrm{sumprod}$. The second pass performs a top-down randomized search over C, by selecting all children of a visited and-node, and by drawing at random one of the children of a visited or- node c according to the distribution $p(c_l) = Q(c_l \mid \boldsymbol{w}')/Q(c \mid \boldsymbol{w}')$ and $p(c_r) = 1 - p(c_l)$. Let T be the tree of visited nodes, and $\boldsymbol{s}$ be the indicator vector of the literals in T. Since $\mathcal{S} \neq \varnothing$, we must have $Q(C \,|\, \boldsymbol{w}) > 0$. Thus, each Bernoulli test performed in T is valid, and hence, $\boldsymbol{s} \in \mathcal{S}$. For any literal x_i occurring in T, let $p(x_i)$ denote the probability of the (unique) path connecting the root to x_i. By a telescoping product of Bernoulli distributions, we get that $p(x_i) = e^{w(i)}/Q(C \,|\, \boldsymbol{w}')$. Therefore, $p(\boldsymbol{s}) = \prod_{i:s(i)=1} p(x_i) = \mathbb{P}_{\boldsymbol{w}}(\boldsymbol{s})$, as desired. ☐

We close this section by highlighting some interesting subclasses of dDNNF. A *decision node* is an or-node of the form $(x_i \wedge c'_l) \vee (\overline{x}_i \wedge c'_r)$, where x_i and $\overline{x}_i$ are opposite literals, and c'_l and c'_r are arbitrary nodes. The class of *Free Binary Decision Diagrams* (FBDD) is the subset of dDNNF in which every or-node is a decision node, and at least one child of any and-node is a literal (Wegener, 2000). For example, if in the dDNNF circuit of Figure 1, we replace the or-node (in blue) by a simple literal, say x_3, then we get an FBDD circuit. The family of Ordered Binary Decision Diagrams (OBDD) is the subclass of FBDD obtained by imposing a fixed ordering on the decision variables. Alternatively, the well-known family of (Binary) Decision Trees (DT) is the subclass of FBDD circuits for which the primal graph is cycle-free. Since all these classes are (strict) subsets of dDNNF, they admit linear-time algorithms for linear optimization and model sampling.

3. Tractable Prediction via Compilation

After an excursion into compilation languages, we are now ready to provide efficient characterizations of combinatorial prediction strategies. Our results are summarized in Table 2.

Algorithm	Regret	Runtime
EH	$\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$	$\mathcal{O}(\|C\|)$
FPL	$\mathcal{O}(d^{\frac{3}{2}}\sqrt{T})$	$\mathcal{O}(\|C\|)$
SGD+PCG	$\mathcal{O}(d(\sqrt{T}+\ln T))$	$\mathcal{O}(d^2\|C\|\ln T)$
δ-CH+PCG	$\mathcal{O}(d(\sqrt{T}+\ln T))$	$\mathcal{O}\left(\frac{d^2\|C\|}{\delta}\ln\frac{T}{\delta}\right)$

Table 2: Expected regrets and per-round running times of combinatorial prediction strategies, implemented on a dDNNF circuit C.

Notably, using the fact that $\|\boldsymbol{s}\|_1 = d/2$, the regret bounds for EH and FPL can easily be derived from (Audibert et al., 2011) and (Hutter & Poland, 2005), respectively. Both strategies are straightforward to implement on dDNNF circuits. Indeed, recall that EH draws, at each trial t, a feasible solution $\boldsymbol{s}_t \in \mathcal{S}$ at random according to the distribution $\mathbb{P}_{-\eta \boldsymbol{L}_t}$, where $\boldsymbol{L}_t = \boldsymbol{\ell}_1 + \cdots + \boldsymbol{\ell}_{t-1}$. So, by direct application of Proposition 2, this strategy runs in $\mathcal{O}(|C|)$ time per round, using a dDNNF representation C of the decision set $\mathcal{S}$. For the FPL strategy, each round t is performed by choosing a minimizer $\boldsymbol{s}_t \in \mathcal{S}$ of the objective function $\eta \boldsymbol{L}_t - \boldsymbol{z}_t$, where $\boldsymbol{z}_t \in \mathbb{R}^d$ is a perturbation vector whose components are independent exponentially distributed random variables. By Proposition 1, the FPL strategy also runs in $\mathcal{O}(|C|)$ time per round, using a dDNNF encoding C of $\mathcal{S}$, and the fact that $|C|$ is in $\Omega(d)$.

However, the OSMD strategy and its specializations, SGD and CH, require more attention, due to the projection-decomposition step involved at each iteration.

3.1. Online Stochastic Mirror Descent

The overall idea of Online Mirror Descent (OMD) is to “follow the regularized leader” through a primal-dual approach (Nemirovski & Yudin, 1983; Beck & Teboulle, 2003). Let $\mathcal{K}$ be a convex set, and let $int(\mathcal{K})$ denotes its interior. Given a regularization function F defined on $\mathcal{K}$, OMD iteratively performs a gradient descent in the interior of the dual space $\mathcal{K}^*$, and projects back the dual point into the primal space $\mathcal{K}$. The connection between $\mathcal{K}$ and $\mathcal{K}^*$ is ensured using the gradients ∇F and ∇F^*, where F^* is the convex conjugate of F, defined on $\mathcal{K}^*$. The projection step is captured by the Bregman divergence of F, which is a function $B_F : \mathcal{K} \times \text{int}(\mathcal{K}) \rightarrow \mathbb{R}$ given by:

$$B_F(\boldsymbol{p}, \boldsymbol{q}) = F(\boldsymbol{p}) - F(\boldsymbol{q}) - \langle \nabla F(\boldsymbol{q}), \boldsymbol{p} - \boldsymbol{q} \rangle$$

In the stochastic variant of OMD, introduced by Audibert et. al. (2011; 2014), and specified in Algorithm 1, each projection is performed onto the subset $\text{conv}(\mathcal{S})$ of $\mathcal{K}$, and the resulting point $\boldsymbol{p}_t$ is decomposed into a convex combination of feasible solutions in $\mathcal{S}$, from which one is picked at random for the prediction task.

Algorithm 1 OSMD

```
**Input:** decision set $\mathcal{S} \subseteq \{0,1\}^d$, horizon $T \in \mathbb{Z}_+$
**Parameters:** regularizer $F$ on $\mathcal{K} \supseteq \text{conv}(\mathcal{S})$, step-size $\eta \in (0,1]$

set $\boldsymbol{u}_1 = \boldsymbol{0}$
**for** $t = 1$ **to** $T$ **do**
    set $\boldsymbol{p}_t \in \text{Argmin}_{\boldsymbol{p} \in \text{conv}(\mathcal{S})} B_F(\boldsymbol{p}, \nabla F^*(\boldsymbol{u}_t))$
    play $\boldsymbol{s}_t \sim \boldsymbol{p}_t$ and observe $\boldsymbol{\ell}_t$
    set $\boldsymbol{u}_{t+1} = \nabla F(\boldsymbol{p}_t) - \eta \boldsymbol{\ell}_t$
**end for**
```

For common regularizers, the gradient $\nabla F(\boldsymbol{p}_t)$ and its dual $\nabla F^*(\boldsymbol{u}_t)$ are easily calculable, and we shall assume that the time spent for their construction is negligible compared with the running time of the linear optimization oracle. In fact, the computational bottleneck of OSMD is to find a minimizer $\boldsymbol{p}_t$ of $B_F(\boldsymbol{p}, \nabla F^*(\boldsymbol{u}_t))$ in the convex hull of $\mathcal{S}$, and to decompose $\boldsymbol{p}_t$ into a convex combination of solutions in $\mathcal{S}$. Fortunately, under reasonable assumptions about the curvature of B_F, this projection-decomposition step can be efficiently computed, using recent results in projection-free convex optimization algorithms.

To this end, we need additional definitions. For a convex set $\mathcal{K}$, a differentiable function $f : \mathcal{K} \rightarrow \mathbb{R}$ is called *α-strongly convex* with respect to a norm $\|\cdot\|$ if

$$f(\boldsymbol{p}') - f(\boldsymbol{p}) \geq \langle \nabla f(\boldsymbol{p}), \boldsymbol{p}' - \boldsymbol{p} \rangle + \frac{\alpha}{2} \|\boldsymbol{p}' - \boldsymbol{p}\|^2$$

Furthermore, f is called *β-smooth*[1] with respect to $\|\cdot\|$ if

$$f(\boldsymbol{p}') - f(\boldsymbol{p}) \leq \langle \nabla f(\boldsymbol{p}), \boldsymbol{p}' - \boldsymbol{p} \rangle + \frac{\beta}{2} \|\boldsymbol{p}' - \boldsymbol{p}\|^2$$

Based on these notions, we say that a Bregman divergence B_F has the *condition number* β/α if B_F is both α-strongly convex and β-smooth with respect to the Euclidean norm $\|\cdot\|_2$ in its first argument. For such regularizers, the next result states that the projection-decomposition step can be approximated in low polynomial time, by exploiting the *Pairwise Conditional Gradient* (PCG) method, a variant of the Frank-Wolfe convex optimization algorithm, whose convergence rate has been analyzed in (Lacoste-Julien & Jaggi, 2015; Garber & Meshi, 2016; Bashiri & Zhang, 2017).

Lemma 1. Let $\mathcal{S} \subseteq \{0,1\}^d$ be a decision set represented by a dDNNF circuit C, and F be a regularizer on $\mathcal{K} \supseteq \text{conv}(\mathcal{S})$ such that B_F has condition number β/α. Then, for any $\boldsymbol{q} \in \text{int}(\mathcal{K})$ and $\epsilon \in (0,1)$, one can find in $\mathcal{O}(\frac{\beta}{\alpha} d^2 |C| \ln \frac{\beta d}{\epsilon})$ time a convex decomposition of $\boldsymbol{p} \in \text{conv}(\mathcal{S})$ such that

$$B_F(\boldsymbol{p}, \boldsymbol{q}) - \min_{\boldsymbol{p}' \in \text{conv}(\mathcal{C})} B_F(\boldsymbol{p}', \boldsymbol{q}) \leq \epsilon$$

[1]This notion of geometric smoothness should not be confused with the structural smoothness of NNF circuits in Section 2.

Algorithm 2 PCG

```
Input: $\mathcal{S} \subseteq \{0,1\}^d$, $f : \mathcal{K} \to \mathbb{R}$, $m \in \mathbb{Z}_+$
Parameters: step-sizes $\{\eta_j\}_{j=1}^m$

let $\boldsymbol{p}_1$ be some point in $\mathcal{S}$
for $j = 1$ to $m$ do
    let $\sum_{i=1}^j \alpha_i \boldsymbol{s}_i$ be the convex decomposition of $\boldsymbol{p}_j$
    set $\boldsymbol{s}_j^+ \in \mathrm{Argmin}_{\boldsymbol{p} \in \mathrm{conv}(\mathcal{S})} \langle \nabla f(\boldsymbol{p}_j), \boldsymbol{p} \rangle$
    set $\boldsymbol{s}_j^- \in \mathrm{Argmin}_{\boldsymbol{s} \in \{s_1, \cdots, s_j\}} \langle -\nabla f(\boldsymbol{p}_j), \boldsymbol{s} \rangle$
    set $\boldsymbol{p}_{j+1} = \boldsymbol{p}_j + \eta_j(\boldsymbol{s}_j^+ - \boldsymbol{s}_j^-)$
end for
```

Proof. Observe that $\mathrm{conv}(\mathcal{S})$ is a *simplex-like polytope* (Bashiri & Zhang, 2017), defined by the linear constraints $\boldsymbol{p} \geq \boldsymbol{0}$, $\sum_{i=1}^N \alpha_i \boldsymbol{s}_i = \boldsymbol{p}$, $\boldsymbol{\alpha} \geq 0$, and $\sum_{i=1}^N \alpha_i = 1$, where $N = |\mathcal{S}|$. So, $\mathrm{conv}(\mathcal{S})$ and B_F satisfy the conditions of Theorem 1 in (Garber & Meshi, 2016), and using the step-sizes advocated by the authors, we get that

$$B_F(\boldsymbol{p}_m, \boldsymbol{q}) - B_F(\boldsymbol{p}^*, \boldsymbol{q}) \leq \frac{\beta d}{2} \exp\left(-\frac{\alpha}{8\beta d^2} m\right)$$

where $\boldsymbol{p}_m$ is the point obtained at the last iteration of PCG, and $\boldsymbol{p}^*$ is the (unique) minimizer of $B_F(\boldsymbol{p}, \boldsymbol{q})$ on $\mathrm{conv}(\mathcal{S})$. Therefore, after $m \geq (8d^2\beta/\alpha)\ln(\beta d/(2\epsilon))$ iterations, we have $B_F(\boldsymbol{p}_m, \boldsymbol{q}) - B_F(\boldsymbol{p}^*, \boldsymbol{q}) \leq \epsilon$. Finally, since each iteration of PCG makes one call to the linear optimization oracle, the runtime complexity follows from Proposition 1. ☐

By OSMD+PCG, we denote the refined version of the OSMD algorithm that uses the PCG method at each trial t in order to approximate the Bregman projection-decomposition step. In addition to a regularizer F and a step-size η, OSMD+PCG takes as parameters a sequence $\{\epsilon_t\}_{t=1}^T$ such that

$$B_F(\boldsymbol{p}_t, \boldsymbol{q}_t) - B_F(\boldsymbol{p}_t^*, \boldsymbol{q}_t) \leq \epsilon_t$$

where $\boldsymbol{p}_t$ is the point returned by PCG, $\boldsymbol{q}_t = \nabla F^*(\boldsymbol{u}_t)$, and $\boldsymbol{p}_t^*$ is the minimizer of $B_F(\boldsymbol{p}, \boldsymbol{q}_t)$ over $\mathrm{conv}(\mathcal{S})$.

Theorem 1. Suppose that OSMD+PCG takes as input a dDNNF representation C of a decision set $\mathcal{S} \subseteq \{0,1\}^d$, and a horizon T, and uses a regularizer F on $\mathcal{K} \supseteq \mathrm{conv}(\mathcal{S})$ such that B_F has condition number β/α, together with a step-size $\eta \in (0,1]$ and a sequence of $\{\epsilon_t\}_{t=1}^T$ such that $\epsilon_t = \gamma/t^2$ for $\gamma > 0$. Then, OSMD+PCG attains the expected regret

$$R_T \leq \sqrt{\frac{2\gamma d}{\alpha}}(\ln T + 1) + \frac{1}{\eta}\max_{\boldsymbol{s}\in\mathcal{S}} B_F(\boldsymbol{s}, \boldsymbol{p}_1^*) + \frac{1}{\eta}\sum_{t=1}^T B_{F^*}(\nabla F(\boldsymbol{p}_t^*) - \eta \boldsymbol{\ell}_t, \nabla F(\boldsymbol{p}_t)) \quad (1)$$

with a per-round running time in $\mathcal{O}\left(\frac{\beta}{\alpha} d^2 |C| \ln \frac{\beta d T}{\gamma}\right)$.

Proof. Let $\boldsymbol{s}^* \in \mathcal{S}$ be the optimal solution chosen with the benefit of hindsight. By decomposing the regret, we have

$$R_T = \sum_{t=1}^T \langle \boldsymbol{\ell}_t, \boldsymbol{p}_t^* - \boldsymbol{s}^* \rangle + \sum_{t=1}^T \mathbb{E}\langle \boldsymbol{\ell}_t, \boldsymbol{s}_t - \boldsymbol{p}_t^* \rangle \quad (2)$$

By Theorem 2 in (Audibert et al., 2014), the first term in (2) is bounded by the last two terms in (1). For the second term in (2), we get from the Cauchy-Schwarz inequality that

$$\mathbb{E}\langle \boldsymbol{\ell}_t, \boldsymbol{s}_t - \boldsymbol{p}_t^* \rangle \leq \|\boldsymbol{\ell}_t\|_2 \|\boldsymbol{p}_t - \boldsymbol{p}_t^*\|_2 \leq \sqrt{d}\|\boldsymbol{p}_t - \boldsymbol{p}_t^*\|_2$$

Moreover, by applying the Generalized Pythagorean Theorem (Cesa-Bianchi & Lugosi, 2006), we know that $B_F(\boldsymbol{p}, \boldsymbol{q}_t) \geq B_F(\boldsymbol{p}, \boldsymbol{p}_t^*) + B_F(\boldsymbol{p}_t^*, \boldsymbol{q}_t)$, for any $\boldsymbol{p} \in \mathrm{conv}(\mathcal{S})$. Using $\boldsymbol{p} = \boldsymbol{p}_t$ and rearranging,

$$B_F(\boldsymbol{p}_t, \boldsymbol{p}_t^*) \leq B_F(\boldsymbol{p}_t, \boldsymbol{q}_t) - B_F(\boldsymbol{p}_t^*, \boldsymbol{q}_t) \leq \epsilon_t \quad (3)$$

Since B_F is α-strongly convex with respect to $\|\cdot\|_2$ in its first argument, we also have $\frac{\alpha}{2}\|\boldsymbol{p}_t - \boldsymbol{p}_t^*\|_2^2 \leq B_F(\boldsymbol{p}_t, \boldsymbol{p}_t^*)$. Thus by plugging this inequality into (3), we get that $\mathbb{E}\langle \boldsymbol{\ell}_t, \boldsymbol{s}_t - \boldsymbol{p}_t^* \rangle \leq \sqrt{2d\epsilon_t/\alpha}$. Finally, by substituting ϵ_t with ρ/t^2, summing other T, and applying the logarithmic bound on harmonic series, we obtain the desired result. ☐

3.2. Stochastic Gradient Descent

The (online) SGD algorithm is derived from OSMD using the Euclidean regularizer $F(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{p}\|_2^2$. In this simple framework, the primal and dual spaces coincide with $\mathbb{R}^d$, and hence, $F^*(\boldsymbol{u}) = \boldsymbol{u}$, $\nabla F(\boldsymbol{p}) = \boldsymbol{p}$, and $\nabla F^*(\boldsymbol{u}) = \boldsymbol{u}$. Furthermore, B_F has the condition number $1/1$, since $B_F(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{2}\|\boldsymbol{p} - \boldsymbol{q}\|_2^2$. We denote by SGD+PCG the instance of OSMD+PCG defined on the Euclidean regularizer.

Proposition 3. The SGD+PCG algorithm achieves an expected regret bounded by $d(\sqrt{T} + \ln T + 1)$ with a per-round runtime complexity in $\mathcal{O}(d^2|C|\ln T)$ using $\eta = 1/\sqrt{T}$ and $\gamma = d/2$.

Proof. This simply follows from Theorem 1, together with the fact that $\max_{\boldsymbol{s}\in\mathcal{S}} B_F(\boldsymbol{s}, \boldsymbol{p}_1^*) \leq d$ and $\|\boldsymbol{\ell}_t\|_2^2 \leq d$. ☐

3.3. Component Hedge

The CH algorithm is derived from OSMD using the entropic regularizer $F(\boldsymbol{p}) = \sum_{i=1}^d p(i)(\ln p(i) - 1)$, for which the conjugate is $F^*(\boldsymbol{u}) = \sum_{i=1}^d \exp u(i)$. Here, we cannot find a finite condition number for the associated divergence $B_F(\boldsymbol{p}, \boldsymbol{q}) = \sum_{i=1}^d p(i) \ln \frac{p(i)}{q(i)} - (p(i) - q(i))$, since its gradient is unbounded. This issue may, however, be circumvented using a simple trick advocated in (Krichene et al., 2015), which consists in replacing the entropic regularizer with the function $F_\delta(\boldsymbol{p}) = F(\boldsymbol{p} + \boldsymbol{\delta})$, where

$\delta \in (0,1)$ and $\boldsymbol{\delta} = (\delta, \cdots, \delta)$. For this function, the primal space is $(-\delta, +\infty)$, and since $F_\delta^*(\boldsymbol{u}) = F^*(\boldsymbol{u}) - \langle \boldsymbol{u}, \boldsymbol{\delta} \rangle$, the dual space is $\mathbb{R}^d$. It is easy to show that

$$\frac{\partial F_\delta(\boldsymbol{p})}{\partial p(i)} = \ln(p(i) + \delta) \qquad \frac{\partial F_\delta^*(\boldsymbol{u})}{\partial u(i)} = e^{u(i)} - \delta$$
$$B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q}) = B_F(\boldsymbol{p}+\boldsymbol{\delta}, \boldsymbol{q}+\boldsymbol{\delta}) \quad B^*_{F_\delta}(\boldsymbol{u}, \boldsymbol{v}) = B^*_F(\boldsymbol{u}, \boldsymbol{v})$$

where $B^*_F(\boldsymbol{u}, \boldsymbol{v}) = \sum_{i=1}^d e^{v(i)}(e^{v(i)-u(i)} + v(i) - u(i) - 1)$. Furthermore, since the first and second order partial derivatives of $B^*_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})$ at the coordinate $p(i)$ are

$$\frac{\partial B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})}{\partial p(i)} = \ln \frac{p(i) + \delta}{q(i) + \delta} \quad \frac{\partial^2 B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})}{\partial^2 p(i)} = \ln \frac{1}{p(i) + \delta}$$

it follows that B_{F_δ} has the condition number $1+\delta/\delta$. Indeed, given an arbitrary point $\boldsymbol{q} \in \text{int}(-\delta, +\infty)$, let $H_{\boldsymbol{q}}(\boldsymbol{p})$ denote the the Hessian matrix of $B_{F_\delta}(\boldsymbol{p}, \boldsymbol{q})$ at $\boldsymbol{p} \in \text{conv}(\mathcal{S})$. Then, for any $\boldsymbol{z} \in \mathbb{R}^d$, the diagonal entries of $H_{\boldsymbol{q}}(\boldsymbol{p})$ satisfy

$$\frac{1}{1+\delta} \leq \frac{\partial^2 B_F(\boldsymbol{p}, \boldsymbol{q})}{\partial^2 p(i)} z(i)^2 \leq \frac{1}{\delta}$$

using the fact that $p(i) \in [0,1]$. Thus, $\alpha \boldsymbol{I} \preccurlyeq H_{\boldsymbol{q}}(\boldsymbol{p}) \preccurlyeq \beta \boldsymbol{I}$ for $\alpha = 1/1+\delta$ and $\beta = 1/\delta$. In what follows, the instance of OSMD+PCG that uses F_δ as regularizer is called δ-CH+PCG.

Proposition 4. The δ-CH+PCG algorithm achieves an expected regret bounded by $d(1 + 2\delta)(\sqrt{T} + \ln T + 1)$ with a per-round runtime complexity in $\mathcal{O}\left(d^2|C|/\delta \ln T/\delta\right)$ using $\eta = 1/\sqrt{T}$ and $\gamma = 2d(1/2 + \delta)/(1 + \delta)$.

Proof. The runtime complexity simply follows from Theorem 1. The regret bound is obtained by bounding the second and third terms of (1), and using the above values for η and γ. Using $\boldsymbol{s}_1^*$ as a maximizer of the second term of (1), we have $B_{F_\delta}(\boldsymbol{s}_1^*, \boldsymbol{p}_1^*) = F_\delta(\boldsymbol{s}_1^*) - F_\delta(\boldsymbol{p}_1^*)$. Using the notation $\tilde{\boldsymbol{p}}_1 = \boldsymbol{p}_1^* + \boldsymbol{\delta}$ and $r = d(1/2 + \delta)$, we get that

$$F_\delta(\boldsymbol{s}_1^*) - F_\delta(\boldsymbol{p}_1^*) \leq \sum_{i=1}^d \tilde{p}_1(i) \ln \frac{1}{\tilde{p}_1(i)} \leq r \ln \frac{d}{r}$$

which is bounded by r. For the third term of (1), observe that F_δ is $\frac{1}{(1+\delta)d}$-strongly convex with respect to the norm $\|\cdot\|_1$, since $\|\boldsymbol{p} - \boldsymbol{p}'\|_1^2 \leq d\|\boldsymbol{p} - \boldsymbol{p}'\|_2^2$. By Theorem 3 in (Kakade et al., 2012), it follows that F_δ^* is $(1 + \delta)d$-smooth with respect to the norm $\|\cdot\|_\infty$. Therefore,

$$\frac{1}{\eta} B_{F^*}(\nabla F(\boldsymbol{p}_t^*) - \eta \boldsymbol{\ell}_t, \nabla F(\boldsymbol{p}_t)) \leq \frac{\eta}{2} d(1 + \delta) \|\boldsymbol{\ell}_t\|_\infty^2$$

which is bounded by ηr. □

4. Experiments

In order to evaluate the performance of the different online combinatorial optimization strategies examined in Section 3, we have considered 16 instances of the SAT Library[2], described in Table 3. Namely, the first six rows of the table are (car) configuration tasks, while the remaining rows are planning problems. In the first four columns of the table are reported the name of the SAT instance, the number of attributes ($d/2$), the number of constraints ($|\text{SAT}|$), and the number $|\mathcal{S}|$ of feasible solutions. We have used the recent D4 compiler [3] (Lagniez & Marquis, 2017) for transforming SAT instances into dDNNF circuits. The size $|C|$ of the compiled circuit is reported in the fifth column.

In order to simulate combinatorial prediction games, we have used the following protocol. Suppose that the set $X = \{x_1, \cdots, x_d\}$ of literals is sorted in a lexicographic way, so that for each odd integer i, the pair (x_i, x_{i+1}) encodes both configurations of the same binary attribute. First, we construct a vector $\boldsymbol{\mu}_0$ of $d/2$ independent Bernoulli variables. At each round $t \in \{1, \cdots, T\}$, $\boldsymbol{\mu}_t$ is set to $\boldsymbol{\mu}_{t-1}$ with probability 0.9, or picked uniformly at random from $[0,1]^{d/2}$ with probability 0.1. Then, the feedback supplied to the learner is a vector $\boldsymbol{\ell}_t \in \{0,1\}^d$ such that $\ell_t(i) + \ell_t(i+1) = 1$, and $\ell_t(i) = 1$ with probability $\mu_t(i+1/2)$ for each odd integer i. So, $\ell_t(i+1) = 1$ with probability $1 - \mu_t(i+1/2)$. Although this protocol is essentially stochastic, the environment secretly resets $\boldsymbol{\mu}_t$ with probability 0.1 at each round to foil the learner.

The combinatorial prediction strategies were implemented in C++ and tested on a six-core Intel i7-5930K with 32 GiB RAM. For the FPL and EH algorithms, we used the step-size η reported in (Audibert et al., 2011) and (Hutter & Poland, 2005), respectively. Concerning the SGD+PCG and δ-CH+PCG algorithms, we used for η and γ the values determined by our theoretical analysis; the step-sizes $\{\eta_t\}$ of PCG were computed from binary search as advocated by Garber & Meshi (2016) in their experiments, and the value of δ was fixed to $1/\ln d$ in order to keep a quadratic runtime complexity for δ-CH+PCG. Finally, the horizon T was set to 10^3, and a timeout of one day was fixed for learning.

In our experiments, the regret is measured by the difference in cumulative loss between the algorithm and the best feasible solution in hindsight, which is obtained using the linear optimization oracle at horizon T. This measure is averaged on 10 simulations, and divided by T to yield an average empirical regret. Similarly, the per-round runtimes (in seconds) are averaged on 10 simulations. The corresponding results are reported in the last four columns of Table 3.

[2]www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html

[3]www.cril.univ-artois.fr/KC/d4.html

Name	$d/2$	$\lvert$SAT$\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert C\rvert$	EH	FPL	SGD+PCG	δ-CH+PCG
c140-fc	1828	4267	$5.74\ 10^{151}$	$6.94\ 10^{5}$	164 ± 25 (22s)	215 ± 48 (21s)	-	-
c163-fw	1815	3580	$2.97\ 10^{140}$	$8.93\ 10^{5}$	98 ± 27 (24s)	112 ± 44 (23s)	-	-
c169-fv	1411	637	$3.22\ 10^{15}$	$7.20\ 10^{2}$	3 ± 1 (<1s)	5 ± 2 (<1s)	1 ± 0.2 (1s)	1 ± 0.2 (1s)
c211-fs	1635	2536	$1.37\ 10^{67}$	$1.75\ 10^{3}$	12 ± 3 (<1s)	12 ± 5 (<1s)	9 ± 2 (1s)	8 ± 2 (2s)
c250-fv	1465	1050	$1.20\ 10^{37}$	$1.82\ 10^{2}$	11 ± 3 (<1s)	16 ± 4 (<1s)	9 ± 2 (1s)	9 ± 1 (1s)
c638-fvk	1761	1893	$8.83\ 10^{121}$	$5.45\ 10^{3}$	104 ± 15 (<1s)	127 ± 28 (<1s)	68 ± 4 (2s)	62 ± 4 (4s)
4-step	165	396	$8.34\ 10^{4}$	$1.29\ 10^{2}$	3 ± 2 (<1s)	5 ± 3 (<1s)	1 ± 0.8 (<1s)	1 ± 0.8 (<1s)
5-step	177	459	$8.13\ 10^{4}$	$5.80\ 10^{1}$	3 ± 1 (<1s)	3 ± 1 (<1s)	1 ± 0.9 (<1s)	1 ± 0.8 (<1s)
log-1	939	3742	$5.64\ 10^{20}$	$9.43\ 10^{2}$	46 ± 5 (<1s)	51 ± 8 (<1s)	7 ± 3 (<1s)	7 ± 1 (<1s)
log-2	1337	24735	$3.23\ 10^{10}$	$1.16\ 10^{4}$	16 ± 3 (1s)	19 ± 6 (1s)	12 ± 3 (16s)	11 ± 3 (22s)
log-3	1413	29445	$2.79\ 10^{11}$	$4.96\ 10^{3}$	19 ± 4 (<1s)	21 ± 7 (<1s)	15 ± 4 (5s)	13 ± 4 (5s)
log-4	2303	20911	$2.34\ 10^{28}$	$9.47\ 10^{4}$	77 ± 11 (2s)	94 ± 27 (2s)	19 ± 4 (72s)	17 ± 4 (81s)
tire-1	352	1022	$8.29\ 10^{36}$	$1.37\ 10^{3}$	3 ± 1 (<1s)	4 ± 2 (<1s)	1 ± 0.8 (<1s)	1 ± 0.8 (< 1s)
tire-2	550	1980	$7.39\ 10^{11}$	$7.26\ 10^{2}$	9 ± 2 (<1s)	12 ± 4 (<1s)	7 ± 2 (<1s)	5 ± 1 (<1s)
tire-3	577	1984	$2.23\ 10^{11}$	$6.31\ 10^{3}$	9 ± 4 (<1s)	14 ± 7 (<1s)	7 ± 2 (1s)	5 ± 2 (2s)
tire-4	812	3197	$1.03\ 10^{14}$	$3.97\ 10^{3}$	15 ± 3 (<1s)	17 ± 5 (<1s)	8 ± 3 (<1s)	7 ± 3 (1s)

Table 3: Experimental results for online combinatorial optimization strategies on SAT instances encoded into dDNNF circuits.

Here, the symbol "−" indicates that the learner was not able to perform the T rounds in one day. From the viewpoint of regret, SGD+PCG and δ-CH+PCG outperform EH and FPL, which confirms our theoretical results. We mention in passing that SGD+PCG and δ-CH+PCG are remarkably stable. Contrastingly, FPL exhibits a larger variance.

Concerning runtimes, EH and FPL are unsurprisingly faster than SGD+PCG and δ-CH+PCG. Notably, for the hard-to-compile instances c140-fc and c163-fw, both EH and FPL were able to perform each trial in few tens of seconds, while OSMD+PCG algorithms took several minutes per-round (and hence, they were unable to process 10^3 rounds in one day), due to the time spent in approximating the Bregman projection step. Yet, it is important to emphasize that the convergence rate of PCG is, in practice, much faster than the theoretical bound of $\tilde{\mathcal{O}}(d^2|C|)$. Both SGD+PCG and δ-CH+PCG were able to process nearly all instances in few seconds per round. For circuits of moderate size, all algorithms run in less than one second per trial. We also observed that SGD+PCG is slightly faster than δ-CH+PCG, especially for large domains where small values of δ have a significant impact on the the runtime complexity. In essence, SGD+PCG offers the best compromise between predictive performance and running time; since all feasible solutions are dense ($\|\boldsymbol{s}\|_1 = d/2$), there is no significant difference in accuracy between SGD+PCG and δ-CH+PCG.

5. Conclusions

We have proposed a general framework for compiling online combinatorial optimization problems, whose space of feasible solutions is described using a set of Boolean constraints. Namely, we have focused on the class of dDNNF circuits which is endowed with fast inference algorithms for the linear optimization oracle and the sampling oracle. Based on this framework, we have shown than both EH and FPL admit fast implementation for tackling large scale online combinatorial problems. A particular attention was devoted to the generic OSMD strategy, which involves a computationally expensive projection-decomposition step at each iteration. To this point, we made use of projection-free algorithms, and in particular the PCG method, for approximating this operation. The resulting algorithms, SGD+PCG and δ-CH-PCG, are inevitably slower than EH and FPL, but achieve a better regret performance, as corroborated by our experiments.

We conclude with a few remarks. In light of the current results, a natural perspective of research is to extend our framework to other classes of combinatorial prediction games. Notably, the *semi-bandit* setting seems within reach. Indeed, the semi-bandit variant of EH, often referred to as EXP2 (Audibert et al., 2014), uses importance weights for estimating the loss at each iteration. By simple adaptation of Proposition 2, such weights can be computed in linear time. Similarly, the semi-bandit extension of FPL exploits the geometric sampling method for estimating loss vectors (Neu & Bartók, 2016). Again, this iterative method can be implemented in linear time (per iteration) using Proposition 1. Less obvious, however, is the extension of OSMD to semi-bandits: although the extension of CH achieves an optimal expected regret in this setting, its practical use remain limited due to projection-decomposition step. An interesting open question is to determine whether a combination of CH with PCG is able, in the semi-bandit case, to achieve a quasi-optimal regret in low-polynomial time. Of course, the *bandit* setting is even more challenging. To this point, Sakaue et. al. (2018) have paved the way using OBDDs for an efficient implementation of the COMBBAND algorithm (Cesa-Bianchi & Lugosi, 2012), Extending their approach to dDNNF, which is more succinct than OBDD, is a promising direction of future research.

References

Audibert, J-Y., Bubeck, S., and Lugosi, G. Minimax policies for combinatorial prediction games. In *Proceedings of the 24th Annual Conference on Learning Theory (COLT 2011)*, pp. 107–132, 2011.

Audibert, J-Y., Bubeck, S., and Lugosi, G. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1):31–45, 2014.

Bashiri, M. A. and Zhang, X. Decomposition-invariant conditional gradient for general polytopes with line search. In *Advances in Neural Information Processing Systems 30, (NIPS 2017)*, pp. 2687–2697, 2017.

Beck, A. and Teboulle, M. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operational Research Letters*, 31(3):167–175, 2003.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. *Handbook of Satisfiability*. IOS Press, 2009.

Bryant, R. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35 (8):677–691, 1986.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Cesa-Bianchi, N. and Lugosi, G. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5): 1404–1422, 2012.

Creignou, N., Khanna, S., and Sudan, M. *Complexity Classification of Boolean Constraint Satisfaction Problems*. SIAM Monographs on Discrete Mathematics and Applications, 2001.

Darwiche, A. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

Darwiche, A. A compiler for deterministic, decomposable negation normal form. In *Proceedings of the 18th National Conference on Artificial Intelligence, (AAAI 2002)*, pp. 627–634, 2002.

Darwiche, A. and Marquis, P. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

Dyer, M. E., Goldberg, L. A., and Jerrum, M. The complexity of weighted Boolean #CSP. *SIAM Journal of Computing*, 38(5):1970–1986, 2009.

Friesen, A. L. and Domingos, P. M. The sum-product theorem: A foundation for learning tractable models. In *Proceedings of the 33nd International Conference on Machine Learning, (ICML 2016)*, pp. 1909–1918, 2016.

Garber, D. and Meshi, O. Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes. In *Advances in Neural Information Processing Systems 29, (NIPS 2016)*, pp. 1001–1009, 2016.

Hannan, J. Approximation to Bayes risk in repeated play. *Contributions to the Theory of Games*, 3:97–139, 1957.

Hazan, E. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

Helmbold, D. P. and Warmuth, M. K. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10:1705–1736, 2009.

Hutter, M. and Poland, J. Adaptive online prediction by following the perturbed leader. *Journal of Machine Learning Research*, 6:639–660, 2005.

Kakade, S. M., Shalev-Shwartz, S., and Tewari, A. Regularization techniques for learning with matrices. *Journal of Machine Learning Research*, 13:1865–1890, 2012.

Kalai, A. T. and Vempala, S. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Koolen, W. M., Warmuth, M. K., and Kivinen, J. Hedging structured concepts. In *Proceedings of the 23rd Conference on Learning Theory (COLT 2010)*, pp. 93–105, 2010.

Krichene, W., Krichene, S., and Bayen, A. M. Efficient bregman projections onto the simplex. In *Proceedings of the 54th IEEE Conference on Decision and Control, (CDC 2015)*, pp. 3291–3298, 2015.

Lacoste-Julien, S. and Jaggi, M. On the global linear convergence of frank-wolfe optimization variants. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pp. 496–504, 2015.

Lagniez, J-M. and Marquis, P. An improved decision-dnnf compiler. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI 2017)*, pp. 667–673, 2017.

Mohri, M. General algebraic frameworks and algorithms for shortest-distance problems. Technical Report 981210-10TM, AT & T Labs-Research, 1998.

Nemirovski, A. S and Yudin, D. B. *Problem Complexity and Method Efficiency in Optimization*. J. Wiley and Sons, 1983.

Neu, G. and Bartók, G. Importance weighting without importance weights: An efficient algorithm for combinatorial semi-bandits. *Journal of Machine Learning Research*, 17:154:1–154:21, 2016.

Poon, H. and Domingos, P. M. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pp. 337–346, 2011.

Rahmanian, H. and Warmuth, M. K. Online dynamic programming. In *Advances in Neural Information Processing Systems 30, (NIPS 2017)*, pp. 2824–2834, 2017.

Rajkumar, A. and Agarwal, S. Online decision-making in general combinatorial spaces. In *Advances in Neural Information Processing Systems 27, (NIPS 2014)*, pp. 3482–3490, 2014.

Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3): 400–407, 1951.

Sakaue, S., Ishihata, M., and Minato, S-I. Efficient bandit combinatorial optimization algorithm with zero-suppressed binary decision diagrams. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS 2018)*, 2018.

Suehiro, D., Hatano, K., Kijima, S., Takimoto, E., and Nagano, K. Online prediction under submodular constraints. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory (ALT 2012)*, pp. 260–274, 2012.

Takimoto, E. and Hatano, K. Efficient algorithms for combinatorial online prediction. In *Proceedings of the-24th International Conference on Algorithmic Learning Theory (ALT 2013)*, pp. 22–32, 2013.

Takimoto, E. and Warmuth, M. K. Path kernels and multiplicative updates. *Journal of Machine Learning Research*, 4:773–818, 2003.

Wegener, I. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Discrete mathematics and applications. SIAM Monographs on Discrete Mathematics and Applications, 2000.