Proceedings of Machine Learning Research vol 99:1–4, 2019 32nd Annual Conference on Learning Theory

Learning to Prune: Speeding up Repeated Computations

Daniel Alabi ALABID@G.HARVARD.EDU
Harvard University

Adam Tauman Kalai NOREPLY@MICROSOFT.COM
Microsoft Research

Katrina Ligett KATRINA@CS.HUJI.AC.IL
Hebrew University

Cameron Musco CAMUSCO@MICROSOFT.COM
Microsoft Research

Christos Tzamos TZAMOS@WISC.EDU
University of Wisconsin-Madison

Ellen Vitercik VITERCIK@CS.CMU.EDU
Carnegie Mellon University

Editors: Alina Beygelzimer and Daniel Hsu

Abstract

Algorithms often must solve sequences of closely related problems. If the algorithm runs a standard procedure with worst-case runtime guarantees on each instance, it will fail to take advantage of valuable structure shared across the problem instances. When a commuter drives from work to home, for example, there are typically only a handful of routes that will ever be the shortest path. A naïve algorithm that does not exploit this common structure may spend most of its time checking roads that will never be in the shortest path. More generally, we can often ignore large swaths of the search space that will likely never contain an optimal solution.

We present an algorithm that learns to maximally prune the search space on repeated computations, thereby reducing runtime while provably outputting the correct solution each period with high probability. Our algorithm employs a simple explore-exploit technique resembling those used in online algorithms, though our setting is quite different. We prove that, with respect to our model of pruning search spaces, our approach is optimal up to constant factors. Finally, we illustrate the applicability of our model and algorithm to three classic problems: shortest-path routing, string search, and linear programming. We present experiments confirming that our simple algorithm is effective at significantly reducing the runtime of solving repeated computations[1].

1. Introduction

For a commuter traveling from home to work every morning, the shortest path may vary from day to day—sometimes side roads beat the highway; sometimes the bridge is closed due to construction. Although San Francisco and New York are contained in the same road network, however, it is unlikely that a San Francisco-area commuter would ever find New York along her shortest path—the edge times in the graph do not change *that* dramatically from day to day.

1. Extended abstract. Full version appears as [arXiv reference, v1904.11875].

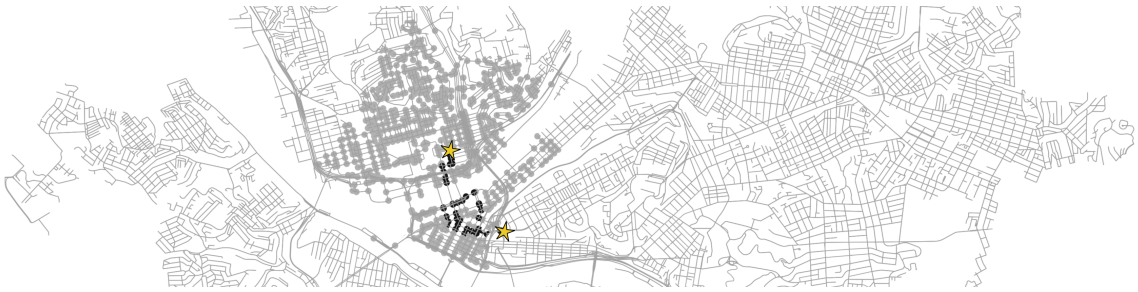

Figure 1: A standard algorithm computing the shortest path from the upper to the lower star will explore many nodes (grey), even nodes in the opposite direction. Our algorithm learns to prune to a subgraph (black) of nodes that have been included in prior shortest paths.

With this motivation in mind, we study a learning problem where the goal is to speed up repeated computations when the sequence of instances share common substructure. Examples include repeatedly computing the shortest path between the same two nodes on a graph with varying edge weights, repeatedly computing string matchings, and repeatedly solving linear programs with mildly varying objectives. Our paper is in the spirit of recent research in data-driven algorithm selection (e.g., Gupta and Roughgarden, 2017; Balcan et al., 2017, 2018a,b) and online learning (e.g., Cesa-Bianchi and Lugosi, 2006, although with some key differences, which we discuss in the full version).

The basis of this work is the observation that for many realistic instances of repeated problems, vast swaths of the search space may never contain an optimal solution—perhaps the shortest path is always contained in a specific region of the road network; large portions of a DNA string may never contain the patterns of interest; a few key linear programming constraints may be the only ones that bind. Algorithms designed to satisfy worst-case guarantees may thus waste substantial computation time on futile searching. For example, even if a single, fixed path from home to work were best every day, Dijkstra's algorithm would consider all nodes within distance d_i from home on day i, where d_i is the length of the optimal path on day i, as illustrated in Figure 1.

We develop a simple solution, inspired by online learning, that leverages this observation to the maximal extent possible. On each problem, our algorithm typically searches over a small, pruned subset of the solution space, which it learns over time. This pruning is the minimal subset containing all previously returned solutions. These rounds are analogous to "exploit" rounds in online learning. To learn a good subset, our algorithm occasionally deploys a worst-case-style algorithm, which explores a large part of the solution space and guarantees correctness on any instance. These rounds are analogous to "explore" rounds in online learning. If, for example, a single fixed path were always optimal, our algorithm would almost always immediately output that path, as it would be the only one in its pruned search space. Occasionally, it would run a full Dijkstra's computation to check if it should expand the pruned set. Roughly speaking, we prove that our algorithm's solution is almost always correct, but its cumulative runtime is not much larger than that of running an optimal algorithm on the maximally-pruned search space in hindsight. Our results hold for worst-case sequences of problem instances, and we do not make any distributional assumptions.

In a bit more detail, let $f : X \to Y$ be a function that takes as input a problem instance $x \in X$ and returns a solution $y \in Y$. Our algorithm receives a sequence of inputs from X. Our high-level goal is to correctly compute f on almost every round while minimizing runtime. For example, each $x \in X$ might be a set of graph edge weights for some fixed graph $G = (V, E)$ and $f(x)$ might be the shortest s-t path for some vertices s and t. Given a sequence $x_1, \ldots, x_T \in X$, a worst-case algorithm would simply compute and return $f(x_i)$ for every instance x_i. However, in many application domains, we have access to other functions mapping X to Y, which are faster to compute. These simpler functions are defined by subsets S of a universe $\mathcal{U}$ that represents the entire search space. We call each subset a "pruning" of the search space. For example, in the shortest paths problem, $\mathcal{U}$ equals the set E of edges and a pruning $S \subset E$ is a subset of the edges. The function corresponding to S, which we denote $f_S : X \to Y$, also takes as input edge weights x, but returns the shortest path from s to t using only edges from the set S. By definition, the function that is correct on every input is $f = f_{\mathcal{U}}$. We assume that for every x, there is a set $S^*(x) \subseteq \mathcal{U}$ such that $f_S(x) = f(x)$ if and only if $S \supseteq S^*(x)$ – a mild assumption we discuss in more detail in the full version.

Given a sequence of inputs $x_1, \ldots, x_T$, our algorithm returns the value $f_{S_i}(x_i)$ on round i, where S_i is chosen based on the first i inputs $x_1, \ldots, x_i$. Our goal is two fold: first, we hope to minimize the size of each S_i (and thereby maximally prune the search space), since $|S_i|$ is often monotonically related to the runtime of computing $f_{S_i}(x_i)$. For example, a shortest path computation will typically run faster if we consider only paths that use a small subset of edges. To this end, we prove that if S^* is the smallest set such that $f_{S^*}(x_i) = f(x_i)$ for all i (or equivalently, $S^* = \bigcup_{i=1}^{T} S^*(x_i)$), then

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}|S_i|\right] \leq |S^*| + \frac{|\mathcal{U}| - |S^*|}{\sqrt{T}},$$

where the expectation is over the algorithm's randomness. At the same time, we seek to minimize the the number of mistakes the our algorithm makes (i.e., rounds i where $f(x_i) \neq f_{S_i}(x_i)$). We prove that the expected fraction of rounds i where $f_{S_i}(x_i) \neq f(x_i)$ is $O(|S^*|/\sqrt{T})$. Finally, the expected runtime[2] of the algorithm is the expected time required to compute $f_{S_i}(x_i)$ for $i \in [T]$, plus $O(|S^*|\sqrt{T})$ expected time to determine the subsets $S_1, \ldots, S_T$.

We instantiate our algorithm and corresponding theorem in three diverse settings—shortest-path routing, linear programming, and string matching—to illustrate the flexibility of our approach. We present experiments on real-world maps and economically-motivated linear programs. In the case of shortest-path routing, our algorithm's performance is illustrated in Figure 1. Our algorithm explores up to five times fewer nodes on average than Dijkstra's algorithm, while sacrificing accuracy on only a small number of rounds. In the case of linear programming, when the objective function is perturbed on each round but the constraints remain invariant, we show that it is possible to significantly prune the constraint matrix, allowing our algorithm to make fewer simplex iterations to find solutions that are nearly always optimal.

2. As we formalize in the full version, when determining $S_1, \ldots, S_T$, our algorithm must compute the smallest set S such that $f_S(x_i) = f(x_i)$ on some of the inputs x_i. In all of the applications we discuss, the total runtime required for these computations is upper bounded by the total time required to compute $f_{S_i}(x_i)$ for $i \in [T]$.

Acknowledgments

This work was supported in part by Israel Science Foundation (ISF) grant #1044/16, a subcontract on the DARPA Brandeis Project, and the Federmann Cyber Security Center in conjunction with the Israel national cyber directorate.

References

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. *Proceedings of the Conference on Learning Theory (COLT)*, 2017.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. *Proceedings of the International Conference on Machine Learning (ICML)*, 2018a.

Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2018b.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.