Learning automatic schedulers with projective reparameterization

Ajay Jain*
UC Berkeley
ajayj@berkeley.edu

Saman Amarasinghe
Massachusetts Institute of Technology
saman@csail.mit.edu

ABSTRACT

Neural networks have proved effective in unconstrained regression and classification problems. Further, commonly used reparameterizations admit learning under certain fixed constraints – e.g. bounds or normalization schemes. Such neural network architectures resort to a small number of differentiable reparameterizations and transformations to express constraints on the output of the parametric model, such as sigmoid or softmax operators. In this work, we propose a generalized differentiable operator, EPOCS, that enforces a large class of dynamic constraints on the output of a neural network with an alternating projection procedure. Such constraints can vary between inputs and are applied during end-to-end training. In particular, EPOCS allows supervised learning in highly-constrained code optimization and scheduling problems. We apply EPOCS to automatic instruction scheduling subject to data-dependent partial orders over the instructions, and train a listwise scheduler to imitate ground-truth schedules end-to-end. Applying dynamic constraints during training with the EPOCS operator yields a 4.1% higher accuracy and a significantly reduces data dependency violations.

1. INTRODUCTION

There is great interest in systems that automatically select efficient schedules to execute user-specified operations. In the compiler literature, automatic assembly instruction scheduling has been formulated through Integer Linear Programming, allowing optimal though costly scheduling through the use of commercial solvers [1, 2]. However, in practice, compilers resort to hand-tuned greedy algorithms, including the popular list scheduling algorithm [3]. Reinforcement learning [4] and supervised learning [5, 6] approaches have been proposed for instruction scheduling, though these schedulers are greedy, resulting in an optimality gap.

Can we learn auto-schedulers that approach the final performance of black-box optimizers, brute-force search, or expertly designed heuristics, without the computational expense? Learning approaches may yield fast *and* near-optimal schedulers, particularly as large datasets of ground truth schedules can be generated offline. Still, training neural schedulers is challenging due to (1) the discrete and combinatorial nature of schedules, and (2) the highly constrained space of feasible schedules – the produced schedule must not alter the functionality of the program.

To allow schedulers to be trained via end-to-end gradient based optimization under input-dependent constraints, we introduce the notion of projective reparameterization, a class of operators that differentiably constrain the output of a neural network to a convex set of feasible solutions. Such reparameterization is particularly useful for learning approaches in systems, as it provides correctness by construction. As an application, we frame local instruction scheduling as a constrained optimization problem, which can be relaxed and amortized via supervised learning of a listwise neural scheduler. In contrast to prior reinforcement learning work [4], our instruction scheduling network predicts a schedule for entire basic blocks rather than parameterize a greedy policy, with efficient end-to-end learning.

Our primary contributions are as follows:

- We formalize and relax the order selection auto-scheduling problem as a constrained optimization (Section 2).
- We develop and analyze an *end-to-end differentiable projection onto convex set* (EPOCS) operator that enforces linear constraints on a neural network's output. We then specialize the operator to partial order constraints (POPOCS, Section 3).
- Finally, we train an automatic instruction scheduler to imitate schedules produced by the GCC compiler and demonstrate accuracy and correctness benefits from dynamic constraints (Section 4).

2. SCHEDULING AS OPTIMIZATION

Consider the class of auto-scheduling problems where a cost minimizing order of execution is selected. That is, $\pi \in \mathscr{P}_n$ must be selected over a partially ordered set of jobs $(X = \{x_1, ..., x_n\}, \prec)$, where $\mathscr{P}_n$ denotes the permutahedron on n objects. For cost model C, this auto-scheduling task is expressible as problem (1).

$$\underset{\pi \in \mathscr{P}_n}{\text{minimize}}\ C(x_{\pi(1)}, ..., x_{\pi(n)}) \tag{1}$$

$$\text{subject to } x_{\pi(i)} \preceq x_{\pi(j)}\ \forall i < j \tag{2}$$

$$\iff \pi^{-1}(a) < \pi^{-1}(b)\ \forall x_a \prec x_b \tag{3}$$

*Work completed while the author was at MIT.

The feasible solutions to the above problem provide the partial order preserving total orders over jobs, $\mathscr{P}_n^{\prec}$. For instance, an exhaustive instruction scheduler could minimize (1) with respect to a model of a processor such as IACA [7], the LLVM Machine Code Analyzer [8] and Ithemal [9], or minimize empirically measured latency. Such problem also generalizes matching problems with optional constraints such as packet switching. However, due to the combinatorial nature of the constraint $\pi \in \mathscr{P}_n$, where $|\mathscr{P}_n|$ grows factorially with n, the optimization is intractable for moderate n and general C. Stochastic search is expensive though feasible [10], and local search or greedy heuristics are used in practice.

As an alternative, we develop a constrained, continuous optimization framework to approximate the problem. Consider parameterization $\pi(i) = \langle \mathbf{P}_i, v \rangle$ in terms of permutation matrix $\mathbf{P}$ and index vector $v = [n]$. As $\mathbf{P}^{-1} = \mathbf{P}^T$ (double stochasticity of permutation matrices), constraint (3) is satisfied if and only if

$$\langle \mathbf{P}_a^T, v \rangle < \langle \mathbf{P}_b^T, v \rangle \; \forall x_a \prec x_b \tag{4}$$

Note that (4) is equivalent to (8).

A nonnegative matrix $\mathbf{P}$ is doubly stochastic if constraints (6) and (7) hold. Further, if $\mathbf{P}$ is binary, then it is also a permutation matrix. Hence, the integer solutions to constraints (6-9) have a bijective mapping with $\mathscr{P}_n^{\prec}$.

To admit optimization, we desire a surrogate cost on $\mathbf{P} \in \mathscr{B}_n^{\prec}$. For example, [9] defines a cost function in terms of continuous embeddings of instructions $\Phi(x_1, ..., x_n)$. Via left multiplication of $\mathbf{P}$ to permute the embeddings, we arrive at objective (5).

$$\underset{\mathbf{P} \in \mathbb{R}^{n \times n}}{\text{minimize}} \; f(\mathbf{P}\Phi(x_1, ..., x_n)) \tag{5}$$

$$\text{subject to} \sum_{i=1}^{n} \mathbf{P}_{ij} = 1 \; \forall j \in \{1, ..., n\} \tag{6}$$

$$\sum_{j=1}^{n} \mathbf{P}_{ij} = 1 \; \forall i \in \{1, ..., n\} \tag{7}$$

$$\sum_{j=1}^{n} j\mathbf{P}_{jb} - \sum_{j=1}^{n} j\mathbf{P}_{ja} \geq 1 \text{ if } x_a \prec x_b \tag{8}$$

$$\mathbf{P}_{ij} \in \{0, 1\} \text{ (relaxed: } \mathbf{P}_{ij} \geq 0) \tag{9}$$

As our relaxation, the integrality constraint (9) is replaced with a nonnegativity constraint. Let $\mathbf{P} \in \mathscr{B}_n$, the Birkhoff polytope of doubly stochastic matrices. By the Birkhoff-von Neumann lemma, $\mathscr{B}_n$ is the convex hull of the permutation matrices [11], motivating the relaxation. This relaxation is also used in prior learning to rank [12, 13, 14, 15] and structure learning [16] work, albeit without a partial order. Together, (6-8) and $\mathbf{P}_{ij} \geq 0$ define a partial order preserving doubly stochastic matrix polytope $\mathscr{B}_n^{\prec}$.

2.1 Conducting the optimization

In the optimization literature, we have well-developed tools to impose linear constraints on parameters. For instance, a gradient-based Augmented Lagrangian solver can minimize (5) (c.f. [17] with additional slack variables). However, the cost model may not be well-defined under the relaxation, where job embeddings are interpolated by multiplication with $\mathbf{P} \in \mathscr{B}_n^{\prec}$. Further, learning a cost model $f(\cdot)$ requires factorially many permutations to effectively capture job ordering effects. Instead of learning a cost model and optimizing with respect to it within $\mathscr{B}_n^{\prec}$, in Section 3, we will develop a learned scheduler that predicts permutations in the feasible set.

2.2 Correcting the relaxation

For both an optimization and learning approach, to correct the Birkhoff relaxation after optimization, we apply the matching operator as used in [15]. The matching operator determines the most similar permutation matrix to the relaxed solution, measured by Hadamard product $\langle \cdot, \cdot \rangle_H$:

$$M(A) = \underset{P \in \text{ext}(\mathscr{B}_n)}{\arg\max} \langle A, P \rangle_H \tag{10}$$

A matching can be computed in $O(n^3)$ time via the Hungarian algorithm [18, 19], and is equivalent to computing a maximum weight bipartite matching with weight matrix given by the relaxed solution.

3. PROJECTIVE REPARAMETERIZATION FOR LEARNED SCHEDULERS

3.1 Motivation for a learned scheduler

Due to challenges discussed in Section 2.1 for learning a cost model under a relaxation, we instead propose learned schedulers that imitate a ground-truth dataset of optimal or near-optimal schedules. It is possible to create such a dataset with an enumerative, stochastic, or branch-and-bound strategy e.g. via ILP solving [1, 2] or auto-tuning. Alternatively, this dataset can be collected from hand-optimized schedules. By training a neural scheduler to predict these optimal schedules, schedules for unseen examples can be derived quickly, thereby amortizing the cost of producing the dataset. We hypothesize that a learned scheduler will produce lower cost schedules than those predicted by hand-written list scheduling heuristics, especially as it can parameterize a non-greedy strategy.

3.2 Enforcing constraints via alternating projections

During forward pass inference in a neural network, there are limited tools for imposing constraints. Instead, practitioners use *reparameterizations* such as the sigmoid activation and exponentiation to map from unconstrained values to a subspace in a differentiable manner. Additional differentiable transformations for fixed constraints include rectified linear unit and softmax normalization. Further, optimization procedures including quadratic programs have been integrated into neural network architectures [20, 21]. In fact, the ReLU, sigmoid, and softmax functions are themselves solutions to optimization problems [21]. Towards a practical unification of the optimization and reparameterization perspectives, we propose a differentiable optimization procedure to impose linear constraints on the output of a network. Objective (11) specifies a minimization problem where an unconstrained prediction x is mapped into a feasible set defined by linear constraints. This is solvable via a *projection onto convex set*

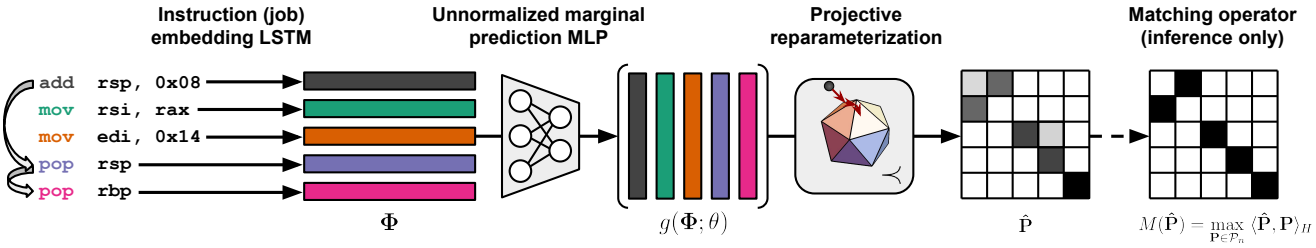


Figure 1: POCSNET network architecture for auto-scheduling as permutation selection. A set of jobs are each embedded into a continuous vector space via an LSTM. From such embeddings, the network predicts unnormalized marginal distributions over job issue times $\tilde{p}(x_i \to j)$. Conditioned on the known partial order $\prec$ over jobs, the POPOCS operator iteratively corrects constraint violations and ensures row and column normalization, yielding the doubly-stochastic matrix $\hat{\mathbf{P}}$ with configurable temperature τ. At inference time, the non-differentiable matching operator $M(\cdot)$ corrects the continuous relaxation, while $\hat{\mathbf{P}}$ is used during training.

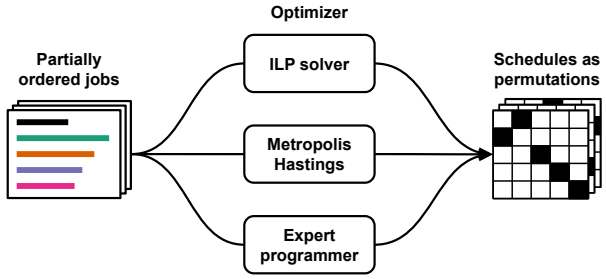


Figure 2: Automated data generation process. A computationally expensive scheduler processes large datasets of job sequences offline, producing a ground-truth schedule dataset to be imitated.

(POCS) algorithm.

$$\underset{\hat{x}}{\text{minimize}} \ \frac{1}{2}||x-\hat{x}||_2^2 \tag{11}$$

$$\text{subject to } A\hat{x} \leq b \tag{12}$$

$$\implies \hat{x} = \text{POCS}_{A,b}(x) \tag{13}$$

The POCS method is a generalization of alternating projection schemes that find a feasible point in a convex polyhedron proximal to an initial unbounded point [22].

Alternating projections with fixed constraints have been used in prior learning-to-rank work. The Sinkhorn-Knopp algorithm [23] is an alternating normalization scheme that alternates between enforcing constraint (6) and (7) on an arbitrary matrix. That is, the Sinkhorn-Knopp algorithm alternatively normalizes rows and columns of the matrix, mapping rows and columns onto the simplex P^n. Given a positive matrix, in the limit of iterations, the Sinkhorn-Knopp algorithm converges to a doubly stochastic matrix. By truncating the alternating normalization optimization and applying differentiable softmax normalizations, the so-called *incomplete* Sinkhorn iteration has been used for end-to-end learning of rankings and matchings [14, 15]. The Sinkhorn iteration can be seen as a projection onto the Birkhoff polytope in a KL-minimizing sense.

Similarly, in the limit of iterations, variants of the projection onto convex sets algorithm will converge to a point in the intersection, enforcing desired constraints. In particular, the space of feasible schedules is often a convex set. In our relaxation, $\mathscr{B}_n^{\prec} \subseteq \mathscr{B}_n \subset [0,1]^{n\times n}$. $\mathscr{B}_n^{\prec}$ is an intersection of the Birkhoff polytope $\mathscr{B}_n$ with a finite number of half-spaces corresponding to constraint (8), so $\mathscr{B}_n^{\prec}$ is a bounded convex set of feasible relaxed schedules.

In Algorithm 1, we define the EPOCS (end-to-end differentiable projection onto convex set) operator that incorporates incomplete POCS optimization into a network's architecture via iterative orthogonal projection. The constraints that define the feasible region for the EPOCS operator may vary between inputs, unlike existing reparameterizations and the Sinkhorn iteration. With an automatic differentiation library, we can backpropagate through such half-space projections for end-to-end learning of a model including EPOCS as a layer. As constraints can be customized, the EPOCS operator defines a general procedure for enforcing constraints on a neural network.

Algorithm 1: EPOCS provides a differentiable projective reparameterization of general dynamic constraints.

```
function EPOCS (G, A, b⃗);
  Input  : G ∈ ℝ^{n×n}, A ∈ ℝ^{m×n²}, b⃗ ∈ ℝ^m
  Output : Ĝ ∈ ℝ^{n×n} such that vec(Ĝ)^T A^T ≤ b⃗
ĝ = vec(G);
for it = 1 to MAXITERS do
    for c = 1 to m do
        if ⟨a⃗_c, ĝ⟩ > b_c then
            // Project onto affine half-space boundary
            u⃗_c = a⃗_c * b_c / ⟨a⃗, a⃗⟩;
            ĝ = u⃗_c + proj_{a⃗_c}(ĝ − u⃗_c);
        end
    end
end
return vec^{-1}(ĝ);
```

3.3 Specializing EPOCS to partial orders

For many problems, the constraint matrix $\mathbf{A}$ in Algorithm 1 is highly sparse. For example, when a partial order is violated, only two columns of the matrix are changed by the orthogonal projection. Exploiting this, we present Algorithm 2, the POPOCS operator, a specialization of EPOCS to map real valued matrices onto the partial order preserving doubly stochastic matrix polytope $\mathscr{B}_n^{\prec}$.

Algorithm 2, POPOCS, extends the Sinkhorn operator to include projections onto the half-spaces defined by constraint (8). While POPOCS can be applied for arbitrarily many iterations, in practice a moderate number of iterations is sufficient to correct partial order violations and approximate a doubly stochastic matrix, whereby computational cost and correctness can be traded off (Section 4).

Algorithm 2: POPOCS provides a sparse, differentiable projective reparameterization of the partial order preserving doubly stochastic matrix polytope $\mathscr{B}_n^{\prec}$.

function POPOCS $(\mathbf{G}, \vec{a}, \vec{b})$;
Input : $\mathbf{G} \in \mathbb{R}^{n \times n}$, $\vec{\alpha} \in \mathbb{R}^m$, $\vec{\beta} \in \mathbb{R}^m$ where $x_{\alpha_i} \prec x_{\beta_i}$
Output : $\mathbf{P} \in \mathbb{R}^{n \times n}$ such that $\mathbf{P} \in \mathscr{B}_n^{\prec}$
$\mathbf{P} = \mathbf{G}$;
for *it* $= 1$ *to* MAXITERS **do**
  $\mathbf{P} = \text{ROWNORMALIZE}(\mathbf{P})$;
    // $\mathbf{P}_{ij} = \mathbf{P}_{ij} - \frac{1}{n}\sum_{j'} \mathbf{P}_{ij'} + \frac{1}{n}$
  $\mathbf{P} = \text{COLNORMALIZE}(\mathbf{P})$;
    // $\mathbf{P}_{ij} = \mathbf{P}_{ij} - \frac{1}{n}\sum_{i'} \mathbf{P}_{i'j} + \frac{1}{n}$
  for $i = 1$ *to* m **do**
    if $\langle \mathbf{P}^T_{\beta_i}, v \rangle - \langle \mathbf{P}^T_{\alpha_i}, v \rangle < 1$ **then**
      // PO violation. Project onto affine space
      $A_{ij} = \begin{cases} -1 & i = \alpha_i \\ 1 & i = \beta_i \\ 0 & \text{otherwise} \end{cases}$
      $\vec{u} = \frac{\text{vec}(A)}{\langle A, A \rangle_H}$;
      $\vec{p} = \vec{u} + \text{proj}_{\text{vec}(A)}(p - \vec{u})$;
      $\mathbf{P} = \text{vec}^{-1}(\vec{p})$;
    end
  end
end
return $\text{vec}^{-1}(\hat{g})$;

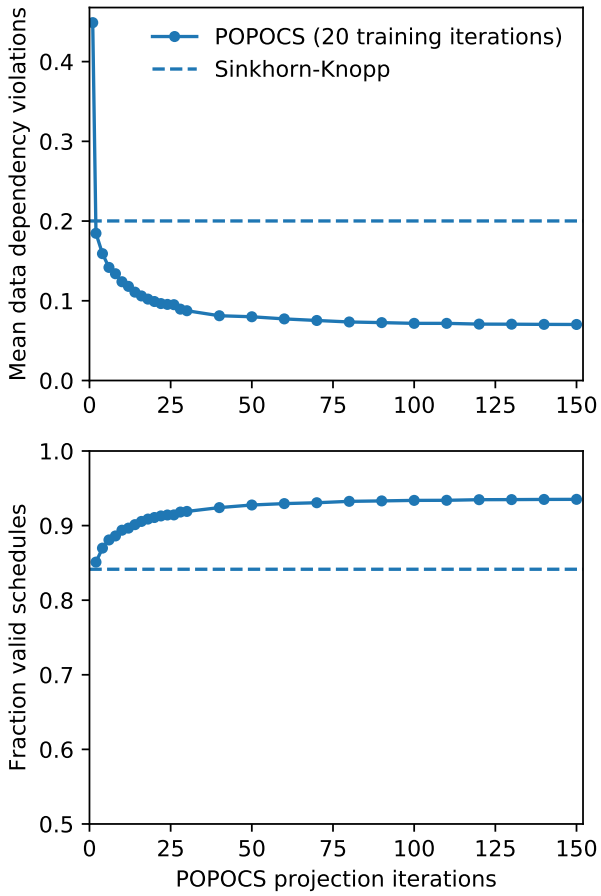


Figure 3: Data dependency violations in schedules predicted by POCSNET decay as more POPOCS iterations are applied at test time. During training, only 20 iterations are applied for efficiency. In contrast, the baseline Sinkhorn auto-scheduler produces invalid schedules at a high rate.

4. EVALUATION

4.1 Scheduling architecture

In Figure 1, we present the POCSNET neural network architecture, an end-to-end supervised order predicting auto-scheduler that applies POPOCS to enforce ordering constraints. The scheduler is non-greedy, selecting a schedule for the entire set of jobs. Such an architecture is applicable to instruction scheduling, text generation, packet switching, and information retrieval. For our instruction scheduling application, each instruction in an input basic block is embedded into a 256-dimensional space by the job embedding LSTM, consisting of a single cell. A multilayer perceptron (MLP) with 2 hidden layers of 256 hidden units flattens the n instruction embeddings and proposes an $n \times n$-dimensional unconstrained matrix. The MLP applies batch normalization and ReLU activations after each hidden layer. Finally, the non-parametric POPOCS operator projects the proposal matrix into the feasible set with data dependencies inferred from the source block.

As a baseline architecture, the POPOCS operator is replaced with 20 iterations of Sinkhorn normalization, enforcing double stochasticity constraints (6) and (7).

4.2 Dataset and training

We train the network to imitate the instruction schedules produced by GCC 4.9.4 on optimization level `-O3` on a dataset of 77,202 basic blocks, each consisting of 5 instructions. These blocks are a subset of those used by [9], extracted from compiled binaries for the SPEC2006 and SPEC2017 datasets. During training, the dataset is augmented by randomly sampling a topological sort of the instructions in each

Table 1: POCSNET metrics on 5 instruction basic blocks

	Accuracy	Kendall tau	Violations	Percent valid schedules
Sinkhorn iteration	0.356	0.238	0.200	84.14%
POPOCS @ 20	**0.397**	**0.222**	0.099	91.08%
POPOCS @ 40	0.396	0.226	0.081	92.41%
POPOCS @ 60	0.393	0.228	0.077	92.95%
POPOCS @ 150	0.394	0.229	**0.070**	**93.52%**

block such that no data dependencies are violated in the input. Metrics are computed on a validation set of 8,579 similarly shuffled basic blocks. Note that not all data dependencies are currently modeled, including those involving status flags. While we train POCSNET to imitate the compiler's behavior, in practice, the training set of basic blocks may be exhaustively scheduled or optimized with stochastic search as it is created offline, as in Figure 2. POCSNET is trained with cross-entropy loss between ground-truth permutation matrices and the predicted, POPOCS-corrected doubly stochastic matrices. The proposed network is optimized with SGD and Nesterov accelerated gradients with 20 alternating projection iterations, and our baseline Sinkhorn network is trained with the Adam optimizer. Both networks are trained for 24 epochs (90,000 iterations) at a batch size of 20 and learning rate of 0.01.

4.3 Results

In Table 1, we show metrics for instruction schedulers learned with a baseline Sinkhorn iteration and the proposed POPOCS operator for various iteration counts. Applying POPOCS iterations significantly reduces the average number of data dependency violations per basic block (PO violations) in predicted schedules while increasing the fraction of schedules that perfectly reconstruct the ground truth block (accuracy). The normalized Kendall tau distance measures similarity between the predicted and ground truth schedules, indicating the fraction of the $\frac{1}{2}n*(n-1)$ instruction pairs that are transposed in the prediction (lower is better).

While POCSNET is trained with 20 projective iterations, additional iterations may be performed at test time to better correct violations with minimal accuracy loss. Figure 3 shows the decay in data dependency violations in predicted schedules as iterations are increased. POCSNET still produces a small number of invalid schedules due to the finite number of corrective iterations and the relaxation introduced in (9). Due to use of the bipartite matching operator (10), constraint (8) may not be satisfied in the discretized solution even if it holds in the relaxed solution. These violations can be corrected via a heuristic, or a greedy scheduler may be applied instead of POCSNET when a data dependency violation is detected.

Schedules predicted by POCSNET have comparable performance to those produced by GCC 4.9.4. In Table 2, we show the mean cycles per block as predicted by the LLVM Machine Code Analyzer (`llvm-mca`) [8] on 7,814 validation blocks with feasible predicted schedules. We evaluate latency with the `llvm-mca` model of Intel X86 Haswell processors, averaged over 100 simulated iterations. As a comparison, we evaluate the predicted latency of the blocks input to the network, which undergo a random topological sort prior to scheduling. In the Appendix, we provide qualitative examples of scheduled basic blocks.

Table 2: Validation basic block latencies from llvm-mca

	Cycles per iteration (mean)
GCC 4.9.4 schedules	2.3396
POCSNet, POPOCS @ 20	2.3430
Random input schedules	2.3464

5. RELATED WORK

Instruction scheduling.

Instruction scheduling is a compiler optimization applied during code generation to maximize instruction level parallelism available in a program. The instruction scheduling literature has seen a host of heuristics for *local* and *global* instruction scheduling, where instructions can move within and between basic blocks (i.e. across control flow), respectively [24]. These approaches include greedy algorithms that select one instruction at a time to dispatch (list scheduling). [1] motivate and [2] develop an integer linear program to optimally and simultaneously select the ordering of instructions in a binary and allocate registers. However, ILP solvers are too computationally expensive for practical use in production compilers. To accelerate scheduling without resorting to greedy heuristics, we develop a learning approach to full basic block instruction scheduling, and apply projective reparameterizations on the output of the network to satisfy ordering restrictions stemming from data dependencies between instructions.

Kernel computation scheduling.

For numerical computing and image processing applications, Halide [25, 26] and Tiramisu [27] provide programming environments that allow programmers to separately specify an algorithm and an execution order (schedule) at a high level of abstraction. These systems originally offered limited auto-scheduling capability. Auto-tuning applies black-box optimization techniques to tune program parameters and has been used for compiler pass ordering, analogous to instruction scheduling [28]. Tensor Comprehensions [29] extends Halide with black-box genetic algorithms that optimize the generated kernel, though this is costly and requires many executions of the kernel itself. Halide now employs heuristics for auto-scheduling [30].

Learning approaches to permutation selection.

The Park system evaluated reinforcement learning techniques for predicting matchings in packet switching; such a matching is parameterized as a permutation [31]. However, their policy only applies to small switch configurations (3 input and output ports) without routing constraints. Training penalties allow permutation matrices to be learned as *parameters* [16], though such penalties cannot guarantee network *outputs* are valid. The information retrieval literature has seen a host of pointwise [32] and pairwise [33, 34, 35] ranking schemes, where items or pairs of items are scored and sorted. For a listwise approach, relaxations of the matching operator through incomplete Sinkhorn normalization [23] reparameterize the output of a network to guarantee double stochasticity and can learn complete list rankings end-to-end [12, 15, 14, 13]. Unlike our work, such approaches have not supported additional constraints such as partial orders. A listwise approach is important for scheduling, as multiple jobs influence the latency of individual jobs.

Continuous relaxations with constraints.

Bayesian network structure learning, a combinatorial DAG selection problem, has been framed as an equality constrained continuous optimization problem (continuous ECP) solvable with gradient based methods [17]. We initially developed a similar approach to solve instruction scheduling via continuous optimization. This requires a pretrained, differentiable objective that captures cost under different orderings such as [9]. In other settings, our supervised learning approach allows optimal or black-box schedulers to be imitated in the absence of a cost model.

6. CONCLUSION

In this work, we propose a general EPOCS operator that imposes constraints on neural network outputs in an end-to-end trainable model. Based on a relaxed formulation of order-based auto-scheduling and a specialization of EPOCS termed POPOCS, we design an auto-scheduler with partial order constraints, trained with supervision from ground-truth schedules. Beyond order-based auto-scheduling, constraint-aware learners are a promising direction of research for the systems community as they enable algorithms that are both correct and data-driven.

7. ACKNOWLEDGEMENTS

We would like to thank Alex Renda and Charith Mendis for providing the basic block dataset used in the evaluation of this work, and for fruitful discussions about our approach. We also thank Paras Jain for discussions and feedback on a draft of this paper. This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

8. REFERENCES

[1] R. Motwani, K. V. Palem, V. Sarkar, and S. Reyen, "Combining Register Allocation and Instruction Scheduling," 1995.

[2] R. C. Lozano, M. Carlsson, G. H. Blindell, and C. Schulte, "Register Allocation and Instruction Scheduling in Unison," in *Proceedings of the 25th International Conference on Compiler Construction*, CC 2016, (New York, NY, USA), pp. 263–264, ACM, 2016. event-place: Barcelona, Spain.

[3] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st ed., 1994.

[4] A. McGovern and J. E. B. Moss, "Scheduling Straight-Line Code Using Reinforcement Learning and Rollouts," in *NIPS*, 1998.

[5] J. E. B. Moss, P. E. Utgoff, J. Cavazos, D. Precup, D. Stefanovic, C. E. Brodley, and D. Scheeff, "Learning to schedule straight-line code," in *Advances in Neural Information Processing Systems*, pp. 929–935, 1998.

[6] T. Russell, "Learning Instruction Scheduling Heuristics from Optimal Data," Master's thesis, University of Waterloo, 2006.

[7] "Intel Architecture Code Analyzer (IACA)."

[8] "LLVM Machine Code Analyzer (llvm-mca)."

[9] C. Mendis, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks," *arXiv:1808.07412 [cs, stat]*, Aug. 2018. arXiv: 1808.07412.

[10] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic Program Optimization," *Commun. ACM*, vol. 59, pp. 114–122, Jan. 2016.

[11] G. Birkhoff, "Tres observaciones sobre el algebra lineal," 1946.

[12] R. P. Adams and R. S. Zemel, "Ranking via Sinkhorn Propagation," *arXiv:1106.1925 [cs, stat]*, June 2011. arXiv: 1106.1925.

[13] S. W. Linderman, G. E. Mena, H. Cooper, L. Paninski, and J. P. Cunningham, "Reparameterizing the Birkhoff Polytope for Variational Permutation Inference," *arXiv:1710.09508 [stat]*, Oct. 2017. arXiv: 1710.09508.

[14] R. S. Cruz, B. Fernando, A. Cherian, and S. Gould, "DeepPermNet: Visual Permutation Learning," *arXiv:1704.02729 [cs]*, Apr. 2017. arXiv: 1704.02729.

[15] G. Mena, D. Belanger, S. Linderman, and J. Snoek, "Learning Latent Permutations with Gumbel-Sinkhorn Networks," *arXiv:1802.08665 [cs, stat]*, Feb. 2018. arXiv: 1802.08665.

[16] J. Lyu, S. Zhang, Y. Qi, and J. Xin, "AutoShuffleNet: Learning Permutation Matrices via an Exact Lipschitz Continuous Penalty in Deep Convolutional Neural Networks," p. 13, Jan. 2019.

[17] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing, "DAGs with NO TEARS: Continuous Optimization for Structure Learning," *arXiv:1803.01422 [cs, stat]*, Mar. 2018. arXiv: 1803.01422.

[18] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, Mar. 1955.

[19] J. Munkres, "Algorithms for the assignment and transportation problems," p. 7, Mar. 1957.

[20] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 136–145, JMLR. org, 2017.

[21] B. Amos, *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University, 2019.

[22] H. H. Bauschke and J. M. Borwein, "On projection algorithms for solving convex feasibility problems," *SIAM review*, vol. 38, no. 3, pp. 367–426, 1996.

[23] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, pp. 343–348, May 1967.

[24] D. Bernstein and M. Rodeh, "Global instruction scheduling for superscalar machines," in *ACM SIGPLAN Notices*, vol. 26, pp. 241–255, ACM, 1991.

[25] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines," p. 12, 2012.

[26] J. Ragan-Kelley, C. Barnes, and A. Adams, "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines," p. 12, 2013.

[27] R. Baghdadi, J. Ray, M. B. Romdhane, E. Del Sozzo, A. Akkas, Y. Zhang, P. Suriana, S. Kamil, and S. Amarasinghe, "Tiramisu: A Polyhedral Compiler for Expressing Fast and Portable Code," *arXiv:1804.10694 [cs]*, Apr. 2018. arXiv: 1804.10694.

[28] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "OpenTuner: An Extensible Framework for Program Autotuning," in *International Conference on Parallel Architectures and Compilation Techniques*, (Edmonton, Canada), Aug. 2014.

[29] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions," *arXiv:1802.04730 [cs]*, Feb. 2018. arXiv: 1802.04730.

[30] R. T. Mullapudi, A. Adams, D. Sharlet, J. Ragan-Kelley, and K. Fatahalian, "Automatically scheduling halide image processing pipelines," *ACM Transactions on Graphics*, vol. 35, pp. 1–11, July 2016.

[31] H. Mao, A. Narayan, P. Negi, H. Wang, J. Yang, H. Wang, M. Khani, S. He, R. Addanki, R. Marcus, F. Cangialosi, W.-H. Weng, S. Han, T. Kraska, and M. Alizadeh, "Park: An Open Platform for Learning Augmented Computer Systems," p. 12, 2019.

[32] W. S. Cooper, F. C. Gey, and D. P. Dabney, "Probabilistic Retrieval Based on Staged Logistic Regression," in *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '92, (New York, NY, USA), pp. 198–210, ACM, 1992. event-place: Copenhagen, Denmark.

[33] C. J. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," in *Advances in neural information processing systems*, pp. 193–200, 2007.

[34] C. J. Burges, "From RankNet to LambdaRank to LambdaMART: An Overview," Tech. Rep. MSR-TR-2010-82, June 2010.

[35] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli, "SortNet: Learning to Rank by a Neural Preference Function," *IEEE Transactions on Neural Networks*, vol. 22, pp. 1368–1380, Sept. 2011.

9. APPENDIX: PREDICTED SCHEDULES

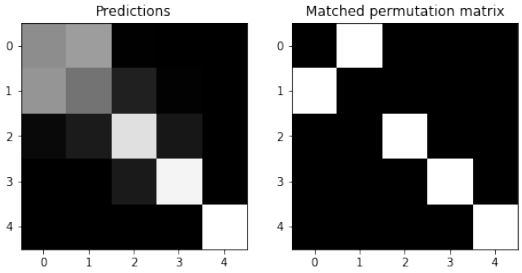
In this section, we present basic blocks extracted from the compiled SPEC benchmark suite where POCSNET and GCC 4.9.4 yield the same schedules. POCSNET is trained and evaluated with 20 iterations of POPOCS, and GCC 4.9.4 schedules the blocks with optimization level `-O3` targetting Haswell processors. These schedules are represented by permutation matrices that specify a transformation to the shuffled block.

Latency is measured in cycles per iteration and is predicted by the `llvm-mca` tool distributed with LLVM 8.0.0. `llvm-mca` is invoked with flags `-x86-asm-syntax=intel -mcpu=haswell`. Cycles are averaged over 100 simulated iterations of execution to expose steady-state processor behavior.

Block emitted by GCC 4.9.4, POCSNET
Cycles per iteration: 14.2

```
movsd   xmm5, qword ptr [rsp+0x20]
movsd   xmm3, qword ptr [r13+0xa0]
subsd   xmm3, xmm5
divsd   xmm3, xmm8
ucomisd xmm2, xmm3
```

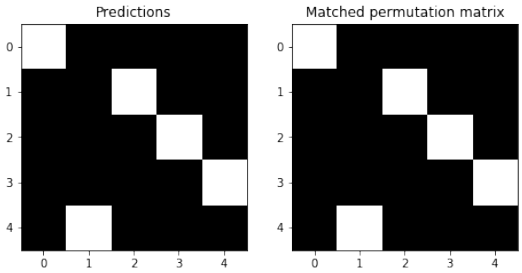
POCSNET proposes a transposition of the first two `movsd` instructions, with low confidence prior to matching due to the similarity of the instructions. Read after write (RAW) data dependencies are preserved.



Block emitted by GCC 4.9.4, POCSNET
Cycles per iteration: 1.28

```
xor     esi, esi
mov     r8d, 0x0000000b
mov     ecx, 0x00000001
lea     rdi, [rsp+0x00017720]
mov     edx, 0x00000001
```

As this basic block contains no data dependencies, only row and column normalizations are performed by POPOCS to produce the prediction (left matrix).



Block emitted by GCC 4.9.4, POCSNET
Cycles per iteration: 13.03

```
add     rsp, 0x18
mov     rsi, rbx
mov     rdi, rbp
pop     rbx
pop     rbp
```

POCSNET proposes a transposition of the first and third instruction in its input block (omitted), yielding the block above and matching GCC.