Differentiable Combinatorial Scheduling at Scale

Mingju Liu [*1] **Yingjie Li** [*1] **Jiaqi Yin** [1] **Zhiru Zhang** [2] **Cunxi Yu** [1]

Abstract

This paper addresses the complex issue of resource-constrained scheduling, an NP-hard problem that spans critical areas including chip design and high-performance computing. Traditional scheduling methods often stumble over scalability and applicability challenges. We propose a novel approach using a differentiable combinatorial scheduling framework, utilizing Gumbel-Softmax differentiable sampling technique. This new technical allows for a fully differentiable formulation of linear programming (LP) based scheduling, extending its application to a broader range of LP formulations. To encode inequality constraints for scheduling tasks, we introduce *constrained Gumbel Trick*, which adeptly encodes arbitrary inequality constraints. Consequently, our method facilitates an efficient and scalable scheduling via gradient descent without the need for training data. Comparative evaluations on both synthetic and real-world benchmarks highlight our capability to significantly improve the optimization efficiency of scheduling, surpassing state-of-the-art solutions offered by commercial and open-source solvers such as CPLEX, Gurobi, and CP-SAT in the majority of the designs.

1. Introduction

Nowadays, the computer-aided scheduling techniques have been widely used in various tasks, such as computing (Cong & Zhang, 2006; Floudas & Lin, 2005; Davis & Burns, 2011; Dhall & Liu, 1978; Steiner et al., 2022; Kathail, 2020; Babu et al., 2021), operations research (Kolisch & Sprecher, 1997; Laborie et al., 2018; Hartmann & Briskorn, 2022), automated systems (Booth et al., 2016a;b; Schmitt & Stuetz, 2016; Tran et al., 2017), transportation (Cappart & Schaus, 2017; Gedik et al., 2017; Kinable et al., 2016). Scheduling plays a crucial role in optimizing time, resources, and productivity, leading to better outcomes and improved efficiency. For example, in the context of computing systems, scheduling is a critical step in computing systems, ensuring optimal performance in hardware synthesis by efficiently allocating resources and timing, and in compilers by determining the sequence of operations to optimize code execution and resource usage.

However, resource- or time-constrained scheduling is a known *NP-hard* problem. Despite an extensive body of prior research and development on either exact or heuristic-based scheduling methods, contemporary scheduling approaches still have major limitations:

(1) **Unfavorable speed-quality trade-off:** Many constrained scheduling problems can be solved exactly using integer linear programming (ILP) (Hwang et al., 1991; Floudas & Lin, 2005; Steiner et al., 2022; Yin et al., 2022), satisfiability (SAT) (Steiner, 2010; Zhang et al., 2004; Coelho & Vanhoucke, 2011), or constraint programming (CP) formulations (Christofides et al., 1987; Laborie et al., 2018; Baptiste et al., 2001; Cesta et al., 2002). However, these approaches suffer from limited scalability. Conversely, popular heuristic methods (Ahn et al., 2020; Paulin & Knight, 1989; Graham, 1969; Blum & Roli, 2003; Brucker et al., 1998) often yield suboptimal results while achieving feasible run times. Notably, a heuristic method based on system of difference constraints (SDC) provides an efficient formulation to encode a rich set of scheduling constraints in SDC and expresses the optimization objective in a linear function that can be solved as an LP problem (Cong & Zhang, 2006; Dai et al., 2018).

(2) **Insufficient utilization of modern parallel computing devices:** Existing scheduling algorithms and solvers are primarily designed for single-threaded CPU execution and are unable to exploit modern parallel computing devices like GPUs (Sanders & Kandrot, 2010) and TPUs (Jouppi et al., 2017).

Recently, machine learning (ML) has been used for combinatorial scheduling for compiler and hardware synthesis to improve its runtime efficiency and explore the expanded decision space (Bengio et al., 2021; Yu et al., 2018; Yu & Zhang, 2019; Neto et al., 2022; Wu et al., 2023). There are

[*]Equal contribution [1]University of Maryland, College Park [2]Cornell University. Correspondence to: Yingjie Li <yingjiel@umd.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

mainly two categories: *imitation learning* (Baltean-Lugojan et al., 2018; Gasse et al., 2019; Gagrani et al., 2022; Wang et al., 2023), where the policy is learned through supervised targets while suffering from difficult data collection and poor model generalizability; *reinforcement learning* (Mascia et al., 2014; Karapetyan et al., 2017; Chen & Shen, 2019; Yin et al., 2023; Yin & Yu, 2023; Yu, 2020; Neto et al., 2022), where the policy is learned from the rewards and potential to outperform the current policy with new discoveries while suffering from limited problem scalability and significant runtime overhead.

In this work, we introduce a scalable approach to differentiable combinatorial scheduling based on SDC formulations employing Gumbel-Softmax (Jang et al., 2016) for the differentiation of scheduling variables and crafting constraints as differentiable distributions for variable discretization. In contrast to existing learning-based approaches, this allows for the customization of objective functions, as well as models the optimization problem of scheduling as a stochastic optimization problem that can be optimized without training and labeled data collection. As a result, our approach introduces an auto-differentiation process for solving combinatorial scheduling without model training. This new approach distinguishes itself from conventional methods by its ability to scale global optimization through parallel computing resources. Moreover, the proposed technique seamlessly integrates with existing ML frameworks like PyTorch, ensuring fast and practical implementation. Our experimental results demonstrate significant improvements in optimization efficiency over state-of-the-art (SOTA) methods solved with commercial solvers CPLEX (IBM, 2023), Gurobi (Gurobi Optimization, LLC, 2023) and open-source CP-SAT solver (Perron & Didier; Perron et al., 2023). Our experimental setups and implementations are available at `https://github.com/Yu-Maryland/Differentiable_Scheduler_ICML24`.

2. Preliminary

2.1. Scheduling and Problem Formulation

Scheduling is one of the most extensively studied combinatorial problems with a wide range of real-world applications. This work focuses on scheduling a dataflow graph, with the input represented as a directed acyclic graph (DAG) $G(V, E)$. In the domain of computing systems, these graphs consist of nodes V, representing tasks that execute specific computations such as arithmetic, logical operations, or ML operators. The edges E represent the flow of data between these nodes. Additional cost metrics can be associated with the nodes and/or edges of the graph in the form of weights. Moreover, a set of scheduling constraints, such as timing constraints and resource constraints, are often specified as part of the formulation, depending on the target scheduling problem. The goal of the optimization is to generate a schedule $S = s_0, s_1, ...s_i$, $i \leq |V|$, where s_i represents the scheduled stage of node v_i, in order to satisfy the given constraints while minimizing or maximizing an objective which is a function of S.

The targeted scheduling in this work is defined as follows: **Given** a DAG $G(V, E)$, where V is the list of nodes to be scheduled, each associated with a per-node resource cost, and E are weighted edges capturing dependency constraints and edge costs. Latency L is the time-to-completion for the entire graph, representing the time between the initiation and completion of the computational task captured by the DAG. The **objective** is to optimize the schedule w.r.t the dependency constraints under a given latency L while minimizing the cost. In other words, we are solving a latency-constrained min-resource scheduling.

System of Difference Constraint (SDC) – An SDC is a system of difference constraints in the integer difference form, denoted as $x_i - x_j \leq c_{ij}$, where c_{ij} is an integer constant, and x_i and x_j are discrete variables. SDC scheduling has been deployed in multiple commercial and open-source high-level synthesis (HLS) tools, such as AMD Xilinx Vivado/Vitis HLS (Kathail, 2020; Cong et al., 2011) and Google XLS (Babu et al., 2021). An SDC is feasible if there exists a solution that satisfies all inequalities in the system. Due to the restrictive form of these constraints, the underlying constraint matrix of SDC is totally unimodular (Camion, 1965), enabling the problem (feasibility checks or optimization) to be solvable in polynomial time with LP solving while ensuring integral solutions. These constraints can be incorporated with a linear objective to formulate an optimization problem, which is leveraged in this work to handle the dependency constraints (Section 3).

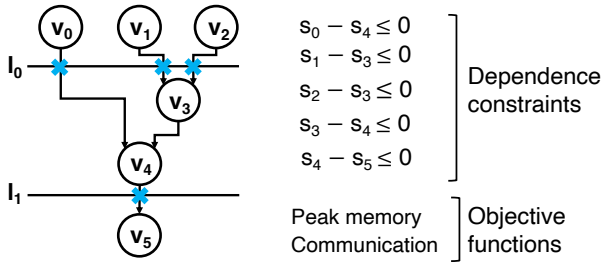


Figure 1: Example of SDC-based scheduling — (left) A DFG with two schedule stages l_0 and l_1 with latency $L = 3$; (right) Dependence constraints and objective functions including peak memory minimization and inter-stage communication minimization (the blue crosses)

We illustrate the SDC-based scheduling formulation with a simple data flow graph (DFG) in Figure 1. To manage the dependencies, SDC establishes a difference constraint for each data edge from operation i to operation j within

the DFG, denoted as $s_i - s_j \leq 0$. In our example, since there is an edge from node v_0 to node v_4, SDC introduces the difference constraint $s_0 - s_4 \leq 0$, ensuring that v_4 is scheduled no earlier than v_0. Similar constraints are formulated for other data-dependent edges. In this work, we leverage SDC formulation with a new technique that implements a fully differentiable SDC to handle dependency constraints in scheduling.

Constraint Programming (CP) – CP is a paradigm for solving combinatorial problems and is an effective method for addressing scheduling problems by allowing both discrete variables and non-linear constraints (Laborie et al., 2018). Unlike LP, which focuses on optimizing a linear objective function and requires constraints to be linear, CP is based on feasibility (finding a feasible solution) rather than optimization (finding an optimal solution). It focuses on the constraints and variables rather than the objective function, which leads to its superiority in managing complex and logical constraints. This makes it ideal for loosely constrained discrete sequencing problems with disjunctive constraints. For example, CP is used to solve the problem of execution time minimization of compute graphs subject to a memory budget (Laborie et al., 2018; Bartan et al., 2023). However, while CP provides significant flexibility and powerful constraint satisfaction capabilities, it can also face challenges with scalability and efficiency.

Learning-based Scheduling – ML approaches have been explored for combinatorial scheduling, particularly in compiler optimization and hardware synthesis, to enhance the Pareto frontier of runtime and quality. Topoformer (Gagrani et al., 2022) introduces a novel attention-based graph neural network architecture for topological ordering, focusing on learning embeddings for graph nodes. While Topoformer has provided significant insights and demonstrated potential in leveraging ML for scheduling, its generalizability and scalability heavily depend on the availability and volume of data. Conversely, reinforcement learning (RL) with graph learning-based schedulers (Chen & Shen, 2019; Yin et al., 2023; Yin & Yu, 2023) aims to improve scalability and generalizability by learning from action rewards, thus eliminating the need for extensive data collection and model generalization required by supervised learning. However, these RL-based approaches still face challenges related to problem scalability, generalizability, and substantial runtime overhead in training.

Heuristic scheduling algorithms Heuristic scheduling algorithms (Ahn et al., 2020; Graham, 1969; Paulin & Knight, 1989; Blum & Roli, 2003) play a critical role in scheduling as well. Notable examples include list scheduling (Graham, 1969), a greedy algorithm that prioritizes tasks based on a predefined order, and force-directed scheduling (Paulin & Knight, 1989), which aims to balance tasks and resources iteratively to achieve latency-constrained, minimum-resource scheduling. In addition, stochastic heuristic methods such as evolutionary algorithms (Blum & Roli, 2003; Wall, 1996) and simulated annealing (Van Laarhoven et al., 1992) are particularly effective in escaping local optima in complex scheduling spaces. While heuristic approaches mostly focus on finding feasible solutions at low runtime costs, they often fall short of reaching the optimal solution.

2.2. Gumbel-Softmax

Gumbel-Softmax is a continuous distribution on the simplex which can be used to approximate discrete samples (Maddison et al., 2016; Jang et al., 2016; Gumbel, 1954). With Gumbel-Softmax, discrete samples can be differentiable and their parameter gradients can be easily computed with standard backpropagation. Let z be the discrete sample with one-hot representation with k dimensions and its class probabilities are defined as $p_1, p_2, ..., p_k$. Then, according to the Gumbel-Max trick proposed by (Gumbel, 1954), the discrete sample z can be presented by:

$$z = \texttt{one\_hot}(\underset{i}{\text{argmax}}[g_i + logp_i]) \tag{1}$$

where g_i are i.i.d samples drawn from Gumbel(0, 1). Then, we can use the differentiable approximation `Softmax` to approximate the one-hot representation for z, i.e., $\nabla_p z \approx \nabla_p y$:

$$y_i = \frac{\exp((\log(p_i) + g_i)/\tau)}{\sum_{i=1}^{k} \exp((\log(p_i) + g_i)/\tau)} \tag{2}$$

where $i = 1, 2, ..., k$. The softmax temperature τ is introduced to modify the distributions over discrete levels. Softmax distributions will become more discrete and identical to one-hot encoded discrete distribution as $\tau \to 0$, while at higher temperatures, the distribution becomes more uniform as $\tau \to \infty$ (Jang et al., 2016). Gumbel-Softmax distributions have a well-defined gradient $\frac{\partial y}{\partial p}$ w.r.t the class probability p. When we replace discrete levels with Gumbel-Softmax distribution depending on its class probability, we are able to use backpropagation to compute gradients.

Gumbel-Softmax provides solutions for the differentiation of discrete scheduling and discrete design space explorations in neural architecture search and quantization tasks (Wu et al., 2019; 2018; He et al., 2020; Fu et al., 2021a;b; Baevski et al., 2019). For instance, (Wang et al., 2023) leverages the Gumbel trick as well as Sinkhorn iterations for combinatorial optimization and utilizes Sinkhorn to implement the problem constraints. However, the study of constrained discrete search and optimization through sampling methods, such as the Gumbel-Softmax, has not been extensively explored, which is particularly critical in scheduling and many other combinatorial optimization problems.

3. Approach

We propose a novel differentiable approach that *compactly encodes* our targeted scheduling problem defined in Section 2.1, which can also be applied to a variety of important scheduling problems on dataflow graphs. Specifically, our method is capable of modeling (1) scheduling constraints in the SDC form and (2) an objective function for resource/cost minimization, both in a differentiable manner. We further introduce a novel *constrained Gumbel Trick*, enabling highly parallelizable scheduling optimization through a sampling-based process with gradient descent.

The remainder of this section will describe the formulation of the targeted scheduling problem in the SDC form, detailing the (1) definition of the search space, (2) modeling of dependencies, and (3) cost metrics (optimization objectives). Afterward, we will present our differentiable approach, aligning it with these three key components.

3.1. Differentiable SDC

With our latency-constrained min-resource scheduling problem, we intend to schedule the node set V on L scheduling stages while minimizing the cost objectives defined in Section 3.2. Note that we allow node chaining, which means two dependent nodes can be scheduled in the same stage, but a node cannot be scheduled earlier than its predecessor.

As mentioned earlier, a critical aspect of scheduling is honoring the dependencies between nodes, which can be specified using SDC. Specifically, these dependencies are translated into integer linear inequalities, which ensure that the resulting schedule adheres to the necessary precedence and resource constraints, maintaining the integrity of the data flow. Specifically, the inequality constraints can be summarized w.r.t the edges E,

$$\forall e(i,j) \in E : s_i - s_j \leq c_{ij} \tag{3}$$

where $e(i,j)$ denotes an edge that connects node i to node j. The term s_i and s_j are the schedule variables for nodes i and j, respectively. Given that we operate under a latency constraint L, all schedule variables follow constraint $\leq L$. To fully encode the scheduling problem as a differentiable model, our approach first addresses the vectorization of the search space, i.e., the vectorization of SDC variables, and then handles the integer inequality constraints with differentiable modeling.

3.1.1. Search Space Vectorization

Given latency constraint L, a vector p in $\mathbb{R}^L$ represents the probability vector of the scheduling decision $\overrightarrow{s}$ for a given node, and the sampled decision $\overrightarrow{s}$ is generated via hard Gumbel-Softmax $\overrightarrow{s}$ = GS(p). As Equation 1 indicates, $\overrightarrow{s}$ is a one-hot vector, which contains a single '1' in its t^{th} coordinate and zeros elsewhere, indicating the variable is scheduled at the scheduling stage t, while p represents the probability distribution of t falling into $[0, L-1]$. Therefore, for any scheduled variable vectorized as p in $\mathbb{R}^L$, its corresponding integer solution space can be defined as $t \in [0, L-1]$ and L is the latency upper bound.

Therefore, in the context of an SDC-encoded schedule, the solution values for each variable can be defined within its vector representation, i.e., $\overrightarrow{s_i} \in \mathbb{R}^L$, with argmax($\overrightarrow{s_i}$) $\in [0, L-1]$. The search space can be fully vectorized by defining all the schedule variables in SDC forms and capturing their dependencies in SDC. Given a DAG $G(V,E)$, our differentiable approach will first define the schedule variables in vector representation, for all $\overrightarrow{s_i} \in \mathbb{R}^L$, $i \in V$. This vectorization establishes a bijection between each integer value and its corresponding one-hot vector. Considering all schedule variables, the search space can be effectively represented by the tensor product of these one-hot vectors. As a result, the total optimizable parameters are in $\mathbb{R}^{|V| \times L}$.

3.1.2. Differentiable Modeling of Inequality Constraints

The next critical step is to ensure the dependency constraints are met using the proposed approach. Specifically, our differentiable scheduling aims to incorporate the dependency constraints defined in E as input, following the integer inequalities constraints in SDC, shown in Equation 3. To encode these inequalities in a differentiable manner, we utilize the cumulative sum (`cumsum`) function. For a schedule variable represented as a one-hot vector $\overrightarrow{s_i}$, the transformation using `cumsum` yields `cumsum`($\overrightarrow{s_i}$), converting $\overrightarrow{s_i}$ into its cumulative sum representation. Generally, the cumulative sum of a vector $v = [v_0, v_1, ..., v_n]$ is $v' = [v_0, v_0 + v_1, ..., \sum_{i=0}^{n} v_i]$. For one-hot vectors, this transformation indicates the feasible space for subsequent Gumbel-softmax sampling operations.

Given an integer inequality constraint $s_i - s_j \leq c_{ij}$, we express it in vector form as $\overrightarrow{s_i} \xrightarrow{c_{ij}} \overrightarrow{s_j}$, where $\overrightarrow{s_i}$ is the sampled discrete solution of s_i, and we define a transformation $T_{\leq} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ for the "$\leq$" constraint, where $\hat{+}$ is an operator that rolls the '1' to the right in the one-hot vector by $|c_{ij}|$ position(s):

$$\mathbf{T}_{\leq}(\overrightarrow{s_i}, c_{ij}) = \texttt{cumsum}(\overrightarrow{s_i} \hat{+} |c_{ij}|) \tag{4}$$

This transformation $\mathbf{T}_{\leq}(\overrightarrow{s_i}, c_{ij})$ effectively constrains the solution space for s_j, represented as $\overrightarrow{s_j}$. Note that non-zero c_{ij} can be used to capture additional timing constraints.

Example – We illustrate the differentiable integer inequality modeling using the constraint $s_0 - s_1 \leq 0$, with $s_0, s_1 \in \mathbb{R}^3$. Let the initial sampling of $\overrightarrow{s_0}$ be $[0,1,0]$, such that s_0 evaluates to '1'. Then, $T_{\leq}$ evaluates to $[0,1,1]$, which implies that if $s_0 = 1$, s_1 can only be sampled as 1 or 2,

which can be confirmed by the original integer inequality constraint. If additional timing constraint is given, e.g., $c_{ij} = -1$, $T_{\leq}$ will be evaluated to $[0, 0, 1]$ by rolling $T_{\leq}$ to the right by one position, which results in $s_1 = 2$ to satisfies the additional timing constraint.

Therefore, we introduce the *constrained Gumbel Trick*, enabling our model to handle inequality constraints. We use $T_{\leq}$ in conjunction with Gumbel distribution sampling to ensure that the sampling always satisfies the constraints:

$$y'_i = \frac{\exp((\log(p_i) + g_i)/\tau)}{\sum_j \exp((\log(p_j) + g_j)/\tau)} \cdot \mathbf{T}_{\leq}, \quad g \sim \text{Gumbel}(0, 1) \tag{5}$$

Lemma 3.1. *For the inequalities $s_i - s_j \leq c_{ij}$, the transformation $T_{\leq}$ ensures any sampled vector space for s_j satisfies the inequality.*

Proof. Consider any arbitrary constraint $s_i - s_j \leq c_{ij}$. The transformation $T_{\leq}$ applied to $\overrightarrow{s_i}$ restricts the feasible vector space for $\overrightarrow{s_j}$. Any vector $\overrightarrow{s_j}$ sampled from this space will satisfy the inequality when converted back to integer values via the bijection. □

3.2. Optimization Process and Objectives

Targeting latency-constrained min-resource scheduling, the optimization process aims to search for the best possible scheduling solutions with the given latency and dependency constraints, guided by a loss function that minimizes resource costs.

3.2.1. Optimization Process

As discussed in Section 3.1, the latency constraint is modeled via our vectorization process of schedule variables, and the dependency constraints are ensured via differentiable inequality modeling. The core of this process is the constrained Gumbel Trick, where we employ vectorized representations of scheduling decisions. Firstly, we calculate the logits for each scheduling decision, then we incorporate these logits into the GS function with our constraint transformations to obtain the sampling probabilities. The process can be mathematically formulated as follows:

Let $\overrightarrow{s}$ be the vector representation of a schedule variable. We make use of the GS function with the constrained Gumbel Trick in Equation 5 to obtain the sampling probability:

$$P = \text{GS}(\mathbf{T}_{\leq}(\overrightarrow{s}, c); \tau), \tag{6}$$

where $P \in \mathbb{R}^L$ and $P^i \in [0, 1]$ for each $i \in [0, L-1]$ with L being the scheduling depth upper bound. Here, $\mathbf{T}_{\leq}$ represents the transformation for inequality constraints and $\text{GS}(\cdot; \tau)$ is the GS function with temperature τ.

The probability of selecting a scheduling solution can be calculated by considering both the conditional probabilities under the constraints and the overall probability of the solution being feasible. For a scheduling solution k, the probability $P(i)$ for a node i can be computed as:

$$P(i) = P(i|k) \cdot P(k) \tag{7}$$

$$= P(i|k) \cdot p(cl(i)) \tag{8}$$

where $P(i|k)$ is the conditional probability of choosing node i given the scheduling solution k, and $p(cl(i))$ represents the probability of the scheduling class $cl(i)$ being feasible under the given constraints.

3.2.2. Differentiable Cost Models

Finally, we introduce a differentiable loss function that integrates the target objectives. As discussed in Section 2.1, we target the scheduling problem of minimizing two cost objectives associated with the nodes and edges of the graph in the form of weights: 1) maximum memory resource utilization calculated with weights of the nodes, and 2) cross-stage communication cost with the weights of the edges.

Specifically, we illustrate the two targeted optimization objectives and metrics using the example in Figure 1. Considering a schedule where v_0, v_1, and v_2 execute in the first stage, while v_5 executes in the last stage, the communication cost is then calculated as the sum of all data transferred between stages I_0 and I_1. As for the memory cost, peak memory refers to the maximum memory used across all stages. Note that the cost metrics can be assessed uniquely w.r.t a given schedule.

To enable parallelizable optimization using gradient descent w.r.t the target objectives, we integrate a differentiable cost function $\mathcal{L}$ based on the scheduling result. To minimize memory usage under a latency constraint, we define the memory loss function $\mathcal{L}_e$, which includes the entropy of scheduled nodes over L stages:

$$\mathcal{L}_e = -\sum_{i=0}^{L-1} \frac{N_i}{M} \log \frac{N_i}{M} \tag{9}$$

where N_i represents the memory of all nodes at the i-th stage, and M is the total memory of all nodes. Assuming uniform memory requirements for each node, N_i is equivalent to the number of nodes at the i-th stage, and $M = |V|$, the total node count. Minimizing $\mathcal{L}_e$ aims to evenly distribute the required memory across stages, which correlates to minimizing the peak memory resource cost. The effectiveness of this entropy-based approach for scheduling resource minimization is originally proven in (Wang et al., 2010).

Furthermore, to account for the minimization of communication cost, we add $\mathcal{L}_c$ into the loss functions. In this

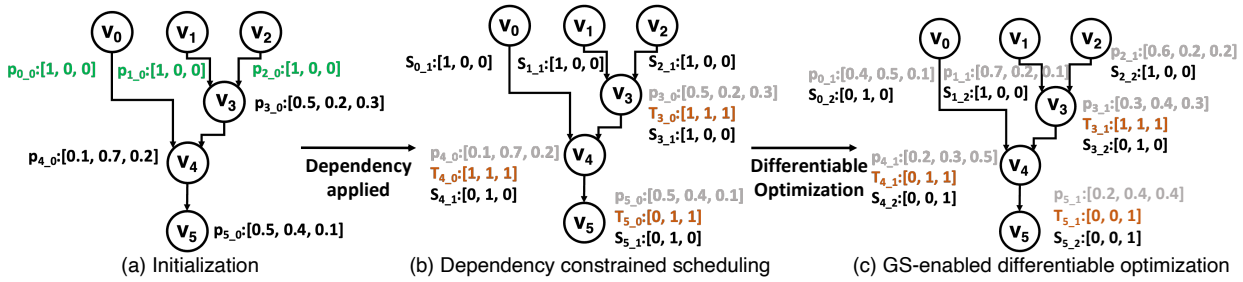


Figure 2: The implementation overview of our differentiable scheduling. p indicates the probability vector for each node during GS-enabled differentiable optimization; T indicates the dependency constraints from the predecessors S indicates the scheduled stage for the node. (a) The search space vectorization and GS initialization. (b) The legal scheduling after applying the dependency constraints. (c) The scheduling is optimized with GS-enabled differentiable optimization.

context, we simply formulate the communication cost as the mean of total cost over all the inter-stages, i.e.,

$$\mathcal{L}_c = \frac{1}{\sum_{b=0}^{|E|-1} c_b} \sum_{i=0}^{L-2} m_i \tag{10}$$

where m_i is the accumulated communication cost for all edges on each inter-stage i, c_b is the communication cost introduced by edge e_b. The final loss is then defined as

$$\mathcal{L} = \lambda \mathcal{L}_e + \mathcal{L}_c \tag{11}$$

where λ is a customizable input used to adjust the optimization ratio between the two objectives.

Implementation The overview of the differentiable scheduling optimization is shown in Figure 2. First, each schedule variable (per node) is initialized with the vectorized search space. More concretely, given 3 available stages, we use 3-dimensional vectors that are made differentiable with GS (Figure 2(a)). Note that, the primary input nodes (v_0, v_1 and v_2) are initialized with the distribution biased to the first stages in GS. Then, the constraints T are applied to the initialized p to produce the legal one-hot scheduling S. For example, for v_5, even though the first stage shows the highest probability in p_{5_0}, its feasible search space is constrained by its predecessors v_4 and v_3 with $T_{5\_0} = [0, 1, 1]$ and the legal scheduling is then selected within the feasible space with the highest probability, i.e., the second stage. In each optimization iteration, in the forward path, the loss function is computed by the legal one-hot scheduling S to guarantee its legalization, and the scheduling is optimized with the backward propagation through the probability vector p in GS to make it differentiable. By updating the probability vector p, the one-hot vector $\overrightarrow{s}$ for variable scheduling given by $\overrightarrow{s}$ = GS(p) will be updated accordingly in the next iteration to realize the differentiable iterative optimization.

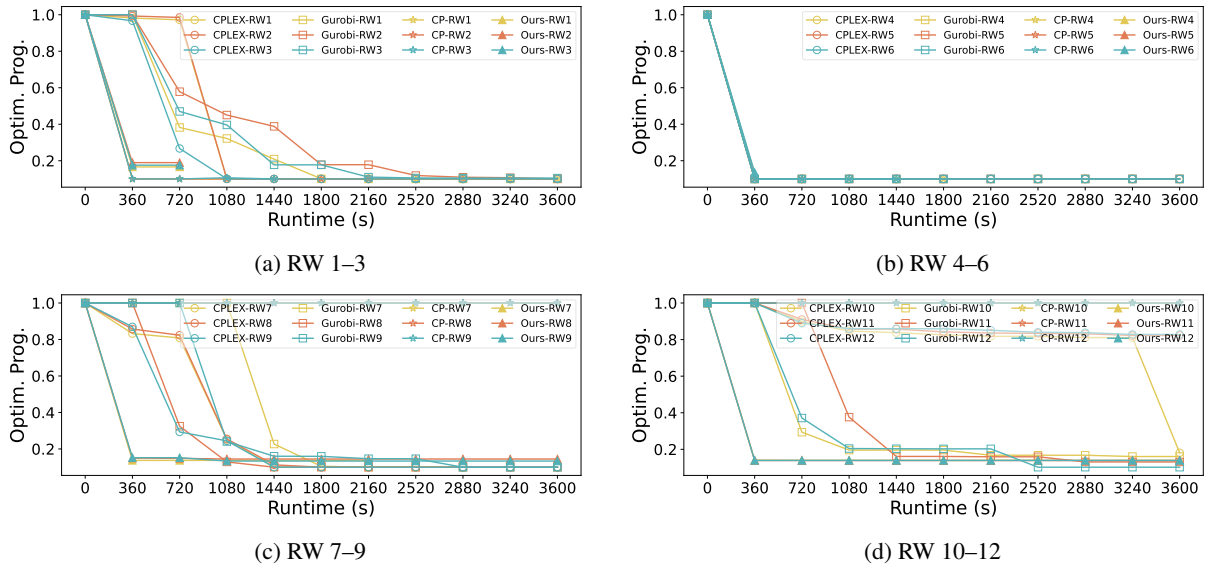
We illustrate the training process of our approach in Algorithm 1 for a given graph $G(V, E)$ with a latency bound L. Initially, the weights W_n are initialized for each node $n \in V$ and collectively stored in W. The algorithm proceeds for a specified number of epochs. In each epoch, an empty schedule S^i is created. The nodes are processed in a topological order. For each node t, a Gumbel-Softmax distribution is computed using the current weights W_t^i. Constraints specific to the edges E_t which start from node t are applied to this distribution by *constrained Gumbel Trick*, and a one-hot encoded vector is extracted to represent the schedule of node t, which is then added to the schedule S^i. The loss $\mathcal{L}$ is computed based on the current schedule S^i, and the gradient ∇W^i of the loss w.r.t. the weights is calculated. Finally, the weights are updated using the computed gradient and a learning rate η, resulting in updated weights W^{i+1} for the next epoch. After training all epochs, a final schedule S is output by the algorithm.

Algorithm 1 Differentiable Scheduling

Require: Graph $G(V, E)$; Targeted latency L
for each $n \in V$ **do**
  $W_n \leftarrow$ Initialize(L)
  $W \leftarrow W \cup \{W_n\}$
end for
for $i = 1$ to *num_epochs* **do**
  $S^i \leftarrow [\,]$
  for each node t in TopologicalOrder(V) **do**
    $GS \leftarrow$ GumbelSoftmax(W_t^i)
    $GS_{E_t} \leftarrow$ ApplyConstraints(GS, E_t)
    $S_t^i \leftarrow$ ExtractOneHot(GS_{E_t})
    $S^i \leftarrow S^i \cup \{S_t^i\}$
  end for
  $\mathcal{L} \leftarrow \mathcal{L}(S^i)$; $\nabla W^i \leftarrow \nabla \mathcal{L}(W^i)$
  $W^{i+1} \leftarrow$ UpdateParameters($W^i, \nabla W^i, \eta$)
end for

4. Experiment

Benchmarks and baselines Our experiments are conducted on specialized scheduling problems in GPU-based circuit



(a) RW 1–3

(b) RW 4–6

(c) RW 7–9

(d) RW 10–12

Figure 3: Performance comparisons with random workloads. Baseline results are SDC scheduling solved by commercial SOTA CPLEX, Gurobi, and CP-SAT solvers.

simulation (Zhang et al., 2022). Due to the nature of Boolean circuits and GPU runtime, the efficiency of simulation workload runtime is highly correlated with data transfer and GPU memory usage. Specifically, (Zhang et al., 2022) pointed out that the critical challenge in scheduling these workloads is to minimize communication overhead and avoid memory bandwidth bottlenecks. While the proposed approach can be evaluated with classic high-level synthesis benchmarks (e.g., MachSuite (Reagen et al., 2014) and Rosseta (Zhou et al., 2018)) or neural network computation graphs (Yin & Yu, 2023; Yin et al., 2023; Steiner et al., 2022), we did not explicitly evaluate those benchmarks due to the fact they are trivial in reaching the optimal using the existing SOTA solvers[1]. Our GPU workloads/graphs are derived using six designs from the EPFL Benchmark Suite (Amarú et al., 2015), alongside baseline SDC+LP formulation solved by the SOTA commercial solvers, CPLEX (IBM, 2023) and Gurobi (Gurobi Optimization, LLC, 2023), as well as open-source tool CP-SAT solver (Perron & Didier). Note that these are non-traditional GPU workloads in scheduling. We extend the benchmarks by adding derived GPU computational graphs of simulated technology-mapped designs using 7nm ASAP technology library (Xu et al., 2017). Additionally, we add twelve synthetic random workloads (RW), all summarized in Table 1. We predefined the ratio of LP and the factor of our method, setting $\mathcal{R} = \lambda = 100$ for all EPFL designs and $\mathcal{R} = \lambda = 10$ for all synthetic workloads, respectively. We set the targeted latency to be $L = 10$ for the experiments. All experiments were conducted using an an Intel®Xeon®Gold 6418H CPU and NVIDIA RTX™4090 GPU. A timeout of 3600 seconds was enforced for all designs across all methods.

[1]See our discussion on limitations in Section 4.3.

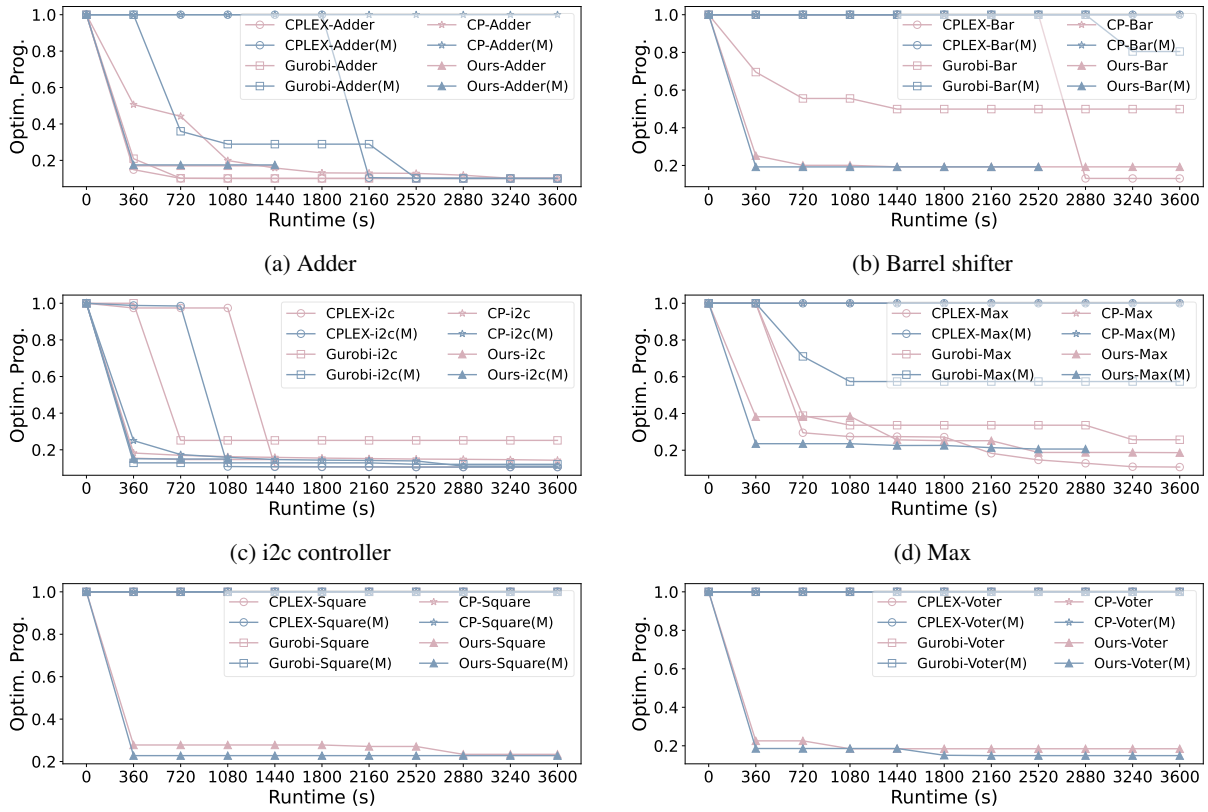
Table 1: Number of nodes and depths for selected designs. **RW*: Random workload. (M): Mapped.

Design	$\|V\|$	#Depth	Design	$\|V\|$	#Depth
Adder	1661	258	RW1	949	15
Adder (M)	1830	89	RW2	941	16
Barrel shifter	3734	15	RW3	929	16
Barrel shifter (M)	2265	10	RW4	810	9
i2c controller	1793	23	RW5	819	8
i2c controller (M)	1221	11	RW6	829	8
Max	4019	290	RW7	4087	10
Max (M)	3493	97	RW8	4063	9
Square	18742	253	RW9	4086	8
Square (M)	18389	96	RW10	8058	9
Voter	15761	73	RW11	8192	11
Voter (M)	20058	39	RW12	8193	9

4.1. Performance Comparison

We established 11 result sampling points at intervals of 360 seconds, ranging from 0 to 3600 seconds, to implement a consistent timeout across all designs. To ensure a fair comparison among the three methods, we employed optimization progress as a normalized factor. Initially, the objective value for all methods is set to 1.0. During optimization, as the tentative objective value decreases, we capture the current best objective value at each sampling point. This value is then expressed as a ratio relative to the initial point, providing a normalized measure of progress.

Evaluation on Synthetic benchmarks As illustrated in Figure 3, we grouped three workloads with similar setups into one subfigure to highlight the similarity in optimization



(a) Adder (b) Barrel shifter

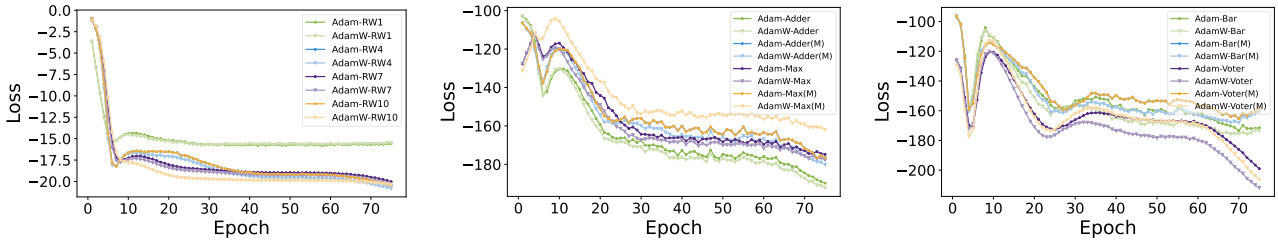
(c) i2c controller (d) Max

(e) Square (f) Voter

Figure 4: Performance comparisons with workloads built for EPFL benchmarks. Baseline results are SDC scheduling solved by commercial SOTA CPLEX, Gurobi, and CP-SAT solvers.

behavior. We observe that our method is particularly effective in optimizing the scheduling solution at early stages, regardless of graph size and density. For instance, as shown in Figures 3a and 3b, our method converges within the initial 360 seconds for both denser workloads (RW 1-3) and less dense workloads (RW 4-6). In contrast, CPLEX and Gurobi barely initiate the optimization process for denser workloads and only achieve comparable performance for sparser workloads. Additionally, Figures 3c and 3d show that our method maintains a similar convergence speed within the first 360 seconds, whereas CPLEX and Gurobi experience nearly a 3× decrease in performance when the workload scale increases from 5000 to 10000. Meanwhile, CP-SAT shows its superiority in Figure 3a, but its performance degrades sharply and fails to produce any optimization when solving larger and denser workloads, as shown in Figure 3c and 3d. This demonstrates the robustness and stability of our method in scalability.

Evaluation on EPFL Benchmarks Following the observations from the synthetic workloads, we extended our verification to real-world designs. Similar to our previous approach, we grouped the original design and its corresponding mapped design in a single subfigure, due to their shared graph characteristics. Consistent with our initial observations, our method consistently exhibits a similar convergence trend between the original and mapped designs, across a range of graph densities. For instance, as demonstrated in Figure 4b, while CPLEX and Gurobi manage to perform adequately on the original (sparse) designs, they exhibit nearly a 2× performance degradation on the mapped (dense) designs during most stages of the total timeout period. Furthermore, with very large designs such as those depicted in Figures 4e and 4f for Square and Voter respectively, CPLEX and Gurobi barely initiate optimization within the 3600 seconds timeout, whereas our method achieves convergence to desirable points early on (within a 360 seconds timeout). As for CP-SAT, it only achieves comparable performance on very limited designs, as shown in Figure 4a and 4c, which are both represented as graphs with smaller and sparser scales. However, it fails to initiate the optimization for the rest of the tested designs. This underscores the advantages of our method in handling large and complex scheduling cases.



(a) Selected Random workloads Loss (b) Selected EPFL designs Loss (c) Selected EPFL designs Loss

Figure 5: Loss Function Evaluation with selected Random workloads and EPFL designs.

4.2. Loss Function Evaluation

Considering that our method demonstrates consistent early convergence across a diverse range of workloads and designs, we focus our investigation on the initial 75 epochs to illustrate the effectiveness of the formulated loss function. Given the similarities among the workloads, only a representative subset has been selected to accommodate page constraints while ensuring a comprehensive analysis.

As depicted in Figure 5a, for the selected synthetic workloads, fast convergence of the loss function is observed during the first 30 epochs, independent of graph size. This rapid convergence indicates the robustness of our method across varying data sizes. Similarly, for the selected EPFL designs, as illustrated in Figures 5b and 5c, the convergence pattern remains consistent, showcasing rapid convergence within the first 75 epochs. An initial instability observed during the first 10 epochs can be attributed to the necessity of a warm-up phase for the Gumbel-Softmax mechanism employed in discrete sampling by our method. This brief period of instability is typical and expected as the model adjusts to the discrete nature of the data. Furthermore, we explored the impact of different weight decay settings by transitioning from the Adam optimizer to AdamW. Upon examining Figure 5, no consistent advantage is observed in the loss convergence trends between the two optimizers. This observation suggests that the optimal choice of weight decay may be design-specific, indicating further investigation.

4.3. Limitations and Discussion

While our method exhibits rapid convergence across a variety of designs, our analysis has unveiled certain limitations for future directions. First, we note that in some specific cases, the optimization quality achieved by our approach falls short when compared to the benchmarks set by established solvers such as CPLEX, Gurobi, and CP-SAT. This discrepancy is particularly evident in designs of a smaller scale or those characterized by sparse graph structures. This observation suggests that our method might benefit from enhanced strategies tailored to these specific problem characteristics, possibly through refined optimization techniques or algorithmic adjustments that better leverage the properties of sparsity and scale.

Second, we observe that while the problem is less complex (trivial), i.e., the problems can be solved very effectively by solving SDC+LP models using LP/CP solvers, our approach does not offer much advantage. Furthermore, we empirically observe that the complexity of the targeted problem is highly associated with the density of the graph $G(V, E)$. As the density of $G(V, E)$ increases, our proposed approach will advance further in the runtime-quality frontier.

Thirdly, our empirical findings indicate instances of 'overfitting' in the loss convergence beyond certain epochs, as illustrated in Figure 5c, which could limit the ability to optimize the objectives. Addressing this challenge might involve exploring advanced regularization methods or adaptive stopping criteria that preemptively halt training before overfitting occurs. Furthermore, these limitations not only highlight areas for improvement but also underscore the importance of developing more nuanced optimization methods capable of adapting to varying problem scales and complexities. Future work could focus on devising novel approaches that explicitly account for the structural characteristics of the problem domain, thereby enhancing the robustness and effectiveness of the method across a broader spectrum of scheduling application scenarios.

5. Conclusion

In this work, we propose an end-to-end differentiable formulation for combinatorial and scalable scheduling, utilizing model-free dataless auto-differentiation with customized objective functions. By harnessing GPU capabilities, our experimental results demonstrate substantial performance improvements over SOTA methods solved with commercial and open-source solvers such as CPLEX, Gurobi, and CP-SAT. Our limitations point towards the necessity for further refinement, particularly in achieving more competitive optimization outcomes and addressing overfitting issues in loss convergence. It potentially broadens the applicability and efficiency of our method in solving complex scheduling problems and other discrete combinatorial problems.

Acknowledgement

This work is supported in part by National Science Foundation (NSF) awards #2047176, #2019306, #2019336, #2008144, #2229562, #2403134, #2403135, and ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

Impact Statement

This paper aims to advance the field of Machine Learning through the development of a novel differentiable combinatorial scheduling algorithm. Our work has many potential societal implications, including enhanced efficiency over industrial SOTA methods. Finally, this work does not raise any ethical concerns that need to be addressed at this time.

References

Ahn, B. H., Lee, J., Lin, J. M., Cheng, H.-P., Hou, J., and Esmaeilzadeh, H. Ordering chaos: Memory-aware scheduling of irregularly wired neural networks for edge devices. *Proceedings of Machine Learning and Systems*, 2:44–57, 2020.

Amarú, L., Gaillardon, P.-E., and De Micheli, G. The epfl combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, number CONF, 2015.

Babu, A., Wang, C., Tjandra, A., Lakhotia, K., Xu, Q., Goyal, N., Singh, K., von Platen, P., Saraf, Y., Pino, J., et al. Xls-r: Self-supervised cross-lingual speech representation learning at scale. *arXiv preprint arXiv:2111.09296*, 2021.

Baevski, A., Schneider, S., and Auli, M. vq-wav2vec: Self-supervised learning of discrete speech representations. *arXiv preprint arXiv:1910.05453*, 2019.

Baltean-Lugojan, R., Misener, R., Bonami, P., and Tramontani, A. Strong sparse cut selection via trained neural nets for quadratic semidefinite outer-approximations. *Imperial College, London, Tech. Rep*, 2018.

Baptiste, P., Le Pape, C., and Nuijten, W. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2001.

Bartan, B., Li, H., Teague, H., Lott, C., and Dilkina, B. Moccasin: Efficient tensor rematerialization for neural networks, 2023.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Blum, C. and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

Booth, K. E., Nejat, G., and Beck, J. C. A constraint programming approach to multi-robot task allocation and scheduling in retirement homes. In *Principles and Practice of Constraint Programming: 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings 22*, pp. 539–555. Springer, 2016a.

Booth, K. E., Tran, T. T., Nejat, G., and Beck, J. C. Mixed-integer and constraint programming techniques for mobile robot task planning. *IEEE Robotics and Automation Letters*, 1(1):500–507, 2016b.

Brucker, P., Knust, S., Schoo, A., and Thiele, O. A branch and bound algorithm for the resource-constrained project scheduling problem. *European journal of operational research*, 107(2):272–288, 1998.

Camion, P. Characterization of totally unimodular matrices. *Proceedings of the American Mathematical Society*, 16 (5):1068–1073, 1965.

Cappart, Q. and Schaus, P. Rescheduling railway traffic on real time situations using time-interval variables. In *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14*, pp. 312–327. Springer, 2017.

Cesta, A., Oddi, A., and Smith, S. F. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8:109–136, 2002.

Chen, H. and Shen, M. A deep-reinforcement-learning-based scheduler for fpga hls. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019. doi: 10.1109/ICCAD45719.2019.8942126.

Christofides, N., Alvarez-Valdés, R., and Tamarit, J. M. Project scheduling with resource constraints: A branch and bound approach. *European journal of operational research*, 29(3):262–273, 1987.

Coelho, J. and Vanhoucke, M. Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. *European Journal of Operational Research*, 213(1):73–82, 2011.

Cong, J. and Zhang, Z. An efficient and versatile scheduling algorithm based on sdc formulation. In *Proceedings of the 43rd annual Design Automation Conference*, pp. 433–438, 2006.

Cong, J., Liu, B., Neuendorffer, S., Noguera, J., Vissers, K., and Zhang, Z. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4): 473–491, 2011.

Dai, S., Liu, G., and Zhang, Z. A scalable approach to exact resource-constrained scheduling based on a joint sdc and sat formulation. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 137–146, 2018.

Davis, R. I. and Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):1–44, 2011.

Dhall, S. K. and Liu, C. L. On a real-time scheduling problem. *Operations research*, 26(1):127–140, 1978.

Floudas, C. A. and Lin, X. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139:131–162, 2005.

Fu, Y., Zhang, Y., Li, C., Yu, Z., and Lin, Y. A3c-s: Automated agent accelerator co-search towards efficient deep reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 13–18. IEEE, 2021a.

Fu, Y., Zhang, Y., Zhang, Y., Cox, D., and Lin, Y. Auto-nba: Efficient and effective search over the joint space of networks, bitwidths, and accelerators. In *International Conference on Machine Learning*, pp. 3505–3517. PMLR, 2021b.

Gagrani, M., Rainone, C., Yang, Y., Teague, H., Jeon, W., Hoof, H. V., Zeng, W. W., Zappi, P., Lott, C., and Bondesan, R. Neural topological ordering for computation graphs, 2022.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Gedik, R., Kirac, E., Milburn, A. B., and Rainwater, C. A constraint programming approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, 107:178–195, 2017.

Graham, R. L. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

Gumbel, E. J. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Hartmann, S. and Briskorn, D. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 297(1):1–14, 2022.

He, C., Ye, H., Shen, L., and Zhang, T. Milenas: Efficient neural architecture search via mixed-level reformulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11993–12002, 2020.

Hwang, C.-T., Lee, J.-H., and Hsu, Y.-C. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):464–475, 1991.

IBM, I. I. Ibm(r) ilog(r) cplex(r) interactive optimizer 22.1.1.0. 2023.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

Karapetyan, D., Punnen, A. P., and Parkes, A. J. Markov chain methods for the bipartite boolean quadratic programming problem. *European Journal of Operational Research*, 260(2):494–506, 2017.

Kathail, V. Xilinx vitis unified software platform. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 173–174, 2020.

Kinable, J., van Hoeve, W.-J., and Smith, S. F. Optimization models for a real-world snow plow routing problem. In *Integration of AI and OR Techniques in Constraint Programming: 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29-June 1, 2016, Proceedings 13*, pp. 229–245. Springer, 2016.

Kolisch, R. and Sprecher, A. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European journal of operational research*, 96(1):205–216, 1997.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. Ibm ilog cp optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23:210–250, 2018.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., and Stützle, T. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & operations research*, 51: 190–199, 2014.

Neto, W. L., Li, Y., Gaillardon, P.-E., and Yu, C. End-to-end automatic logic optimization exploration via domain-specific multi-armed bandit. *arXiv preprint arXiv:2202.07721*, 2022.

Paulin, P. G. and Knight, J. P. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(6):661–679, 1989.

Perron, L. and Didier, F. Cp-sat. URL https://developers.google.com/optimization/cp/cp_solver/.

Perron, L., Didier, F., and Gay, S. The cp-sat-lp solver. In Yap, R. H. C. (ed.), *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 3:1–3:2, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-300-3. doi: 10.4230/LIPIcs.CP.2023.3. URL https://drops.dagstuhl.de/opus/volltexte/2023/19040.

Reagen, B., Adolf, R., Shao, Y. S., Wei, G.-Y., and Brooks, D. Machsuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 110–119. IEEE, 2014.

Sanders, J. and Kandrot, E. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

Schmitt, M. and Stuetz, P. Perception-oriented cooperation for multiple uavs in a perception management framework: System concept and first results. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pp. 1–10. IEEE, 2016.

Steiner, B., Elhoushi, M., Kahn, J., and Hegarty, J. Olla: Decreasing the memory usage of neural networks by optimizing the lifetime and location of arrays. *arXiv preprint arXiv:2210.12924*, 2022.

Steiner, W. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *2010 31st IEEE Real-Time Systems Symposium*, pp. 375–384. IEEE, 2010.

Tran, T. T., Vaquero, T., Nejat, G., and Beck, J. C. Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots. *Journal of Artificial Intelligence Research*, 58:523–590, 2017.

Van Laarhoven, P. J., Aarts, E. H., and Lenstra, J. K. Job shop scheduling by simulated annealing. *Operations research*, 40(1):113–125, 1992.

Wall, M. B. *A genetic algorithm for resource-constrained scheduling*. PhD thesis, Massachusetts Institute of Technology, 1996.

Wang, R., Shen, L., Chen, Y., Yang, X., and Yan, J. Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case. In *ICLR*, 2023.

Wang, X., Gao, L., Zhang, C., and Shao, X. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 51:757–767, 2010.

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., and Keutzer, K. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

Wu, N., Li, Y., Hao, C., Dai, S., Yu, C., and Xie, Y. Gamora: Graph learning based symbolic reasoning for large-scale boolean networks. *Design Automation Conference (DAC'23)*, 2023.

Xu, X., Shah, N., Evans, A., Sinha, S., Cline, B., and Yeric, G. Standard cell library design and optimization methodology for asap7 pdk. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 999–1004. IEEE, 2017.

Yin, J. and Yu, C. Accelerating exact combinatorial optimization via rl-based initialization – a case study in scheduling, 2023.

Yin, J., Zhang, Z., and Yu, C. Exact memory-and communication-aware scheduling of dnns on pipelined edge tpus. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pp. 203–215. IEEE, 2022.

Yin, J., Li, Y., Robinson, D., and Yu, C. Respect: Reinforcement learning based edge scheduling on pipelined coral edge tpus. *arXiv preprint arXiv:2304.04716*, 2023.

Yu, C. Flowtune: Practical multi-armed bandits in boolean optimization. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.

Yu, C. and Zhang, Z. Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

Yu, C., Xiao, H., and De Micheli, G. Developing synthesis flows without human knowledge. In *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

Zhang, H., Li, D., and Shen, H. A sat based scheduler for tournament schedules. In *SAT*, 2004.

Zhang, Y., Ren, H., Sridharan, A., and Khailany, B. Gatspi: Gpu accelerated gate-level simulation for power improvement. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1231–1236, 2022.

Zhou, Y., Gupta, U., Dai, S., Zhao, R., Srivastava, N., Jin, H., Featherston, J., Lai, Y.-H., Liu, G., Velasquez, G. A., et al. Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 269–278, 2018.