Generalized Conflict-directed Search for Optimal Ordering Problems

Jingkai Chen, Yuening Zhang, Cheng Fang, Brian C. Williams
Massachusetts Institute of Technology
jkchen@csail.mit.edu, zhangyn@mit.edu, cfang@mit.edu, williams@csail.mit.edu

Abstract

Solving planning and scheduling problems for multiple tasks with highly coupled state and temporal constraints is notoriously challenging. An appealing approach to effectively decouple the problem is to judiciously order the events such that decisions can be made over sequences of tasks. As many problems encountered in practice are over-constrained, we must instead find relaxed solutions in which certain requirements are dropped. This motivates a formulation of optimality with respect to the costs of relaxing constraints and the problem of finding an optimal ordering under which this relaxing cost is minimum. In this paper, we present Generalized Conflict-directed Ordering (GCDO), a branch-and-bound ordering method that generates an optimal total order of events by leveraging the generalized conflicts of both inconsistency and suboptimality from sub-solvers for cost estimation and solution space pruning. Due to its ability to reason over generalized conflicts, GCDO is much more efficient in finding high-quality total orders than the previous conflict-directed approach CDITO. We demonstrate this by benchmarking on temporal network configuration problems, which involves managing networks over time and makes necessary tradeoffs between network flows against CDITO and Mixed Integer-Linear Programing (MILP). Our algorithm is able to solve two orders of magnitude more benchmark problems to optimality and twice the problems compared to CDITO and MILP within a runtime limit, respectively.

1 Introduction

In order to plan for many real-world problems, autonomous systems are required to take into account the requirements over timing and system states. This category of problems ranges from the classical job shop scheduling problems (Manne 1960) to hybrid planning problems for multiple tasks with coupled state and temporal constraints (Wang and Williams 2015). The key to this body of work has been to abstract the tasks, which are then ordered and checked against state and temporal requirements. With a consistent total order, these abstracted tasks are then refined into more concrete courses of actions by resource managers or schedulers. The choice of ordering algorithms is particularly important. A good ordering algorithm should prune unhelpful orderings as much as possible to avoid the computationally expensive checks of the state and temporal consistency. Recent work demonstrates how generalizing inconsistent orderings of events through the interaction with sub-solvers can greatly accelerate this ordering procedure while exploring a special total order tree (Wang 2015; Chen et al. 2019).

However, the problems specified by these abstract tasks or non-practitioner users are often over-constrained and contain requirements drawing on competing resources. A key challenge to solve such problems is to provide high-quality relaxed solutions in which some requirements are dropped in order to meet hard constraints representing the environment characteristics or other higher-priority requirements. This motivates a notion of optimality with respect to constraint relaxation, as an extension to previous approaches that only considered orderings for constraint satisfaction (Wang 2015; Chen et al. 2019). While recent work has addressed optimal constraint relaxation problems purely for temporal constraints (Yu and Williams 2013), we aim to develop an algorithm that can interact with various underlying solvers such that optimal relaxation problems with tightly-coupled state and temporal constraints can be tackled by solving an optimal ordering problem.

In this paper, we introduce Generalized Conflict-directed Ordering (GCDO), an ordering algorithm that generates optimal total orders of the start and end events of abstract tasks. The optimality is defined with respect to a set of constraints with relaxing costs. GCDO first starts with a total order of these events and then incrementally changes partial orders in a branch-and-bound (B&B) manner, during which inconsistency or suboptimality discovered by sub-solvers are summarized as bounding constraints to render cost estimation and solution space pruning.

Our optimal ordering algorithm is based on the well-known B&B search method (Lawler and Wood 1966). The basic idea of B&B is implicitly enumerating all the solutions and pruning suboptimal subtrees along the way. With different branching and bounding rules, B&B has been used to solve a wide range of optimization problems. From the perspective of B&B, as we arrange all the total orders in a special tree (Ono and Nakano 2005), we branch by exploring subtrees whose total orders share some common partial orders. Then, we bound and prune the subtrees whose costs are provably suboptimal, given the shared partial orders.

To estimate the cost of total orders for pruning suboptimal ones, we draw inspiration from the idea of Conflict-directed Incremental Total Ordering (CDITO), which extracts conflicting partial orders from sub-solvers to guide the search (Chen et al. 2019). We further extend these conflicting partial orders to account for costs, and thus the search is able to estimate total order costs without using sub-solvers and directly jump over suboptimal subtrees. The idea of interacting with sub-solvers is also similar in spirit to Satisfiability Module Theories (SMT) solvers such as Z3 (De Moura and Bjørner 2008) or Optimization Modulo Theories (OMT) solvers such as OptiMathSAT (Sebastiani and Trentin 2020) while we are using sub-solvers to find relaxations with respect to suboptimality instead of satisfiability.

2 Motivating Example

Consider a network configuration problem in which we need to schedule and route four network flows with different priorities and allocate bandwidth resources of a network. In this network, the links have different characteristics of loss, delay, and bandwidth capacity, as shown in Figure 1.

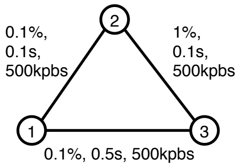

Figure 1: Network topology and statistics.

All these four flows have source node 1 and destination node 2, and Table 1 gives their mission specifications on maximal loss, maximal delay, and minimal required throughput. A detailed description of these specifications can be found in (Chen et al. 2018). The allowable duration lengths for all the flows are [30, 60] seconds. There are also priorities associated with these flows. As Flow-A and Flow-D are very important, they must be transferred. The costs for dropping Flow-B and Flow-C are 3 and 5, respectively. The mission also has temporal requirements: (1) Flow-B and Flow-C should start at the same time, and their ends should be at least 20 seconds apart; (2) Flow-A and Flow-D should start and end at the same times; (3) Flow-B and Flow-C should finish before Flow-A and Flow-D end; (4) we prefer the whole mission to finish in 70 seconds; if the mission takes longer than required, cost 1 is incurred.

Flow	Cost	Loss	Delay	Throughput
A	∞	0.5%	1s	200kbps
B	5	3%	1s	360kbps
C	3	3%	0.3s	360kbps
D	∞	3%	1s	360kbps

Table 1: Mission specifications of flows.

A total order of the flow starts and ends with the minimal cost and its corresponding routes are given in Figure 2. We also show the temporal constraints of the example in Figure 2. It can be easily verified that such a plan only violates the fourth temporal requirement with cost 1. Now we prove this plan is optimal. Given the limited bandwidth constraints, a network link can transfer at most one flow simultaneously. By checking the loss and delay, we know the only feasible route of Flow-A and Flow-C is Path 1-2, while Flow-B and Flow-D can take either Path 1-2 or Path 1-3-2. From the temporal requirements, we know Flow-B and Flow-C must be concurrent, and Flow-A and Flow-D must be concurrent. Furthermore, these two clusters cannot be concurrent, given the limited network capacity, which leads to an 80-second horizon and violates the fourth requirement. Therefore, there is no plan to satisfy all the requirements, and the plan in Figure 2 that only violates one constraint with cost 1 is optimal.

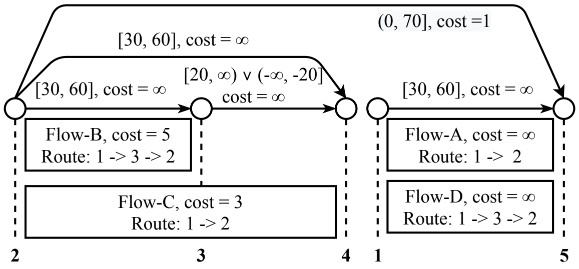

Figure 2: Optimal solution of the motivating example.

As we can see, this problem involves managing network flows with different priorities and multiple characteristics, which is hard, especially when multiple flows are considered over a large network (Chen et al. 2018). The problem becomes much harder when an exponential number of relaxation choices are considered to minimize total costs. We show that our ordering method can efficiently find an optimal solution for this problem by interacting with proper sub-solvers.

3 Problem Formulation

Based on the definitions of general ordering problems (Chen et al. 2019), we associate each constraint with a real-valued cost or an infinite cost, which is similar to the valued constraint satisfaction problems (Schiex et al. 1995), and the optimal ordering problem is defined as a tuple $\langle E, \Phi, w, h\rangle$:

- E is a set of n events represented by the natural numbers $\{1, 2, .., n\}$.
- Φ is a set of constraints, consisting of a set of ordering constraints Φ^+ and a set of theory constraints Φ^*. An ordering constraint $\phi^+ \in \Phi^+$ is a disjunction of partial orders. A partial order $(a \prec b)$ constrains $a \in E$ to precede $b \in E$. A theory constraint is a user-defined state or temporal requirements across a set of events over time.
- $w : \Phi \rightarrow \mathbb{R}_+ \cup \{\infty\}$ is a function that maps a constraint $\phi \in \Phi$ to a positive cost value $g(\phi)$. If $w(\phi) = \infty$, ϕ is a hard constraint; otherwise, ϕ is a soft constraint.
- $h : \mathcal{L} \times 2^\Phi \rightarrow \{\top, \bot\}$ is a function that maps a total order $\mathcal{L}$ of E and a set of constraints $\Phi' \subseteq \Phi$ to a Boolean value indicating the consistency of $\mathcal{L}$ with respect to Φ'.

A candidate solution of this ordering problem is a total order $\mathcal{L}$ that is a sequence of all the events of E. Note that we do not allow events to co-occur, and thus we require a strict ordering such that $\neg(a \prec b) = (b \prec a)$.

To define the solution consistency and optimality, we first introduce the notions of constraint relaxation and the cost of total orders with respect to its constraint relaxation. A set of constraints $\Phi' \subseteq \Phi$ is a relaxation of Φ under total order $\mathcal{L}$ if $h(\mathcal{L}, \Phi/\Phi') = \top$ (i.e., $\mathcal{L}$ is consistent with the rest of the constraints). A trivial relaxation under any total order is Φ, which means suspending all the constraints. A relaxation Φ^* under $\mathcal{L}$ is an optimal relaxation if $\sum_{\phi \in \Phi^*} w(\phi) \leq \sum_{\phi \in \Phi'} g(\phi)$ holds for any relaxation Φ' under $\mathcal{L}$ (i.e., the cost sum of optimal relaxations is minimum). Formally, the cost of $\mathcal{L}$ is denoted as $g(\mathcal{L})$ and defined as

$$g(\mathcal{L}) = \min_{(\Phi' \subseteq \Phi) \wedge h(\mathcal{L}, \Phi/\Phi')} \sum_{\phi \in \Phi'} w(\phi).$$

Note that, $g(\mathcal{L})$ is in the form of $k\infty + c$, where k is a non-negative integer representing the total number of the relaxed hard constraints and c is a real-valued number representing the cost sum of the relaxed soft constraints. We treat $(k\infty + c)$ as finite if $k = 0$.

A candidate solution $\mathcal{L}$ is a solution if and only if its cost $g(\mathcal{L})$ is finite. A solution $\mathcal{L}$ is an optimal solution if and only if $g(\mathcal{L}) \leq g(\mathcal{L}')$ holds for any solution $\mathcal{L}'$.

We assume the cost evaluation function g is provided, which can return the cost of a total order in terms of its optimal relaxation of Φ given consistency function h.

Our motivating example can be formulated as follows:

- $E = \{1, 2, 3, 4, 5\}$ as shown in Figure 2.
- $\Phi^+ = \{o_1, o_2, o_3, o_4, o_5\}$: $o_1 = (1 \prec 5)$, $o_2 = (2 \prec 3)$, and $o_3 = (2 \prec 4)$ constrain each flow's start to precede its end; $o_4 = (3 \prec 5)$ and $o_5 = (4 \prec 5)$ captures temporal requirement (3). All of these constraints are hard constraints with cost ∞.
- $\Phi^* = \{t_1, t_2, t_3, t_4, t_5\} \cup \{s_1, s_2, s_3, s_4\}$: t_1, t_2 and t_3 constrain the duration of each flow to be within $[30, 60]$; t_4 and t_5 captures temporal requirements (1) and (4), respectively; $\{s_1, s_2, s_3, s_4\}$ represents the state constraints on routing, loss, delay, and bandwidth of each flow. All of these constraints are hard constraints except $w(t_5) = 1$, $w(s_2) = 5$, and $w(s_3) = 3$.
- h is able to take as input a total order and determine whether there exists a valid plan, which specifies flow routing, bandwidth allocation, and schedules, satisfies the given set of constraints and respects this total order. To implement g for this domain, we use the network configuration manager in (Chen et al. 2018) and the optimal temporal network relaxation solver in (Yu and Williams 2013) to optimize the cost with respect to the state and temporal constraints, respectively.

One optimal solution of the above problem is 23415 with cost 1 by relaxing temporal constraint t_5.

4 Approach

In this section, we present the design and implementation of the GCDO algorithm, which adopts the well-known branch-and-bound (B&B) search to systematically explore all the total orders of events and find an optimal ordering solution in an anytime manner. One core idea of B&B is to bound and prune suboptimal solutions, which requires (1) estimating the objective bounds of subsets of solution candidates; and (2) pruning the suboptimal subsets given these estimations.

We leverage the following two ideas to pave the way for cost estimation and pruning: (1) a well-defined tree structure for enumerating the set of total orders (Ono and Nakano 2005), such that total orders in a subtree have partial orders in common; (2) bounding constraints that summarize the cost of satisfying a set of partial orders and constraints. With this tree structure and the bounding constraints, we can estimate the cost bound of a total order or all the total orders in a subtree by summing up the costs of their manifested bounding constraints. By using bounding constraints to estimate costs, GCDO avoids many expensive queries about the exact costs of total orders from sub-solvers. Moreover, given the cost estimation of subtrees, we can safely prune inconsistent or suboptimal ones by directly jumping over them.

By using GCDO, the optimal total order of our motivating example can be found in the third iteration as given in Figure 3. GCDO starts with total order 12345, in which all the flows transfer concurrently and leads to total order cost 8. Then, it identifies that by moving event 1 after event 3, the next total order 23145 resolves this concurrency and reduces the cost by 8. GCDO then checks 23145 and finds the concurrency of Flow-A, Flow-C, and Flow-D leads to cost 3. By moving 1 after 4, this concurrency is resolved, and we end up with the optimal solution 23415 with cost 1, which only needs to relax the overall makespan constraint t_5. These concurrencies are examples of bounding constraints, which are partial orders and constraints that impose a certain cost when satisfied. Then, GCDO takes another thirteen iterations to prove this total order is optimal, during which GCDO mainly uses the bounding constraints to estimate costs and only calls cost evaluation function g once.

GCDO (Algorithm 1) takes as input the total number of events n, a set of ordering constraints Φ^+, a cost evaluation function g, and a bounding constraint extraction function f. GCDO outputs either $(\{\}, \infty)$ or an optimal total order $\mathcal{L}^*$ along with its cost γ^* with respect to g (Line 18). Total order $\mathcal{L}$ is initialized as the root total order $(1, 2, .., n)$, and the search status l_c is set to 0 (Line 1). Then, Line 2 initializes the incumbent total order $\mathcal{L}^*$, the incumbent cost γ^*, and the extracted bounding constraints Θ. Starting from $(1, 2, .., n)$, the algorithm explores all the total orders in a systematic way and updates the incumbents when better solutions are found (Line 8). We provide a high-level explanation to the pseudo-code in the following four paragraphs and their implementation details are introduced in the rest of Section 4.

Total Order Search Our algorithm follows a systematic search strategy in the total order tree introduced by (Ono and Nakano 2005). In each iteration, GCDO only maintains one total order $\mathcal{L}$ and its search status l_c, which is the level

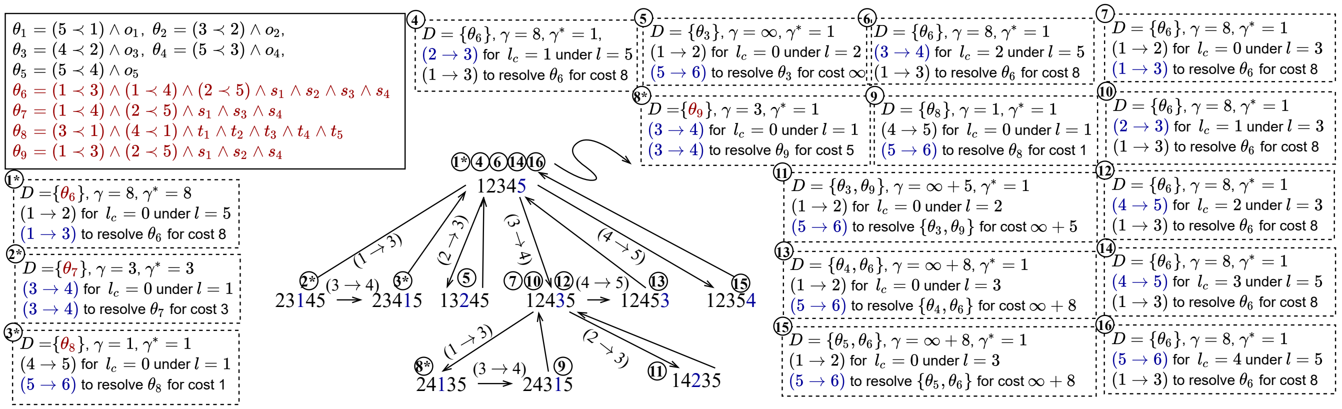
$\theta_1 = (5 \prec 1) \wedge o_1,\ \theta_2 = (3 \prec 2) \wedge o_2,$
$\theta_3 = (4 \prec 2) \wedge o_3,\ \theta_4 = (5 \prec 3) \wedge o_4,$
$\theta_5 = (5 \prec 4) \wedge o_5$
$\theta_6 = (1 \prec 3) \wedge (1 \prec 4) \wedge (2 \prec 5) \wedge s_1 \wedge s_2 \wedge s_3 \wedge s_4$
$\theta_7 = (1 \prec 4) \wedge (2 \prec 5) \wedge s_1 \wedge s_3 \wedge s_4$
$\theta_8 = (3 \prec 1) \wedge (4 \prec 1) \wedge t_1 \wedge t_2 \wedge t_3 \wedge t_4 \wedge t_5$
$\theta_9 = (1 \prec 3) \wedge (2 \prec 5) \wedge s_1 \wedge s_2 \wedge s_4$
1* $D = \{\theta_6\}, \gamma = 8, \gamma^* = 8$
$(1 \to 2)$ for $l_c = 0$ under $l = 5$
$(1 \to 3)$ to resolve θ_6 for cost 8
2* $D = \{\theta_7\}, \gamma = 3, \gamma^* = 3$
$(3 \to 4)$ for $l_c = 0$ under $l = 1$
$(3 \to 4)$ to resolve θ_7 for cost 3
3* $D = \{\theta_8\}, \gamma = 1, \gamma^* = 1$
$(4 \to 5)$ for $l_c = 0$ under $l = 1$
$(5 \to 6)$ to resolve θ_8 for cost 1
4 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1,$
$(2 \to 3)$ for $l_c = 1$ under $l = 5$
$(1 \to 3)$ to resolve θ_6 for cost 8
5 $D = \{\theta_3\}, \gamma = \infty, \gamma^* = 1$
$(1 \to 2)$ for $l_c = 0$ under $l = 2$
$(5 \to 6)$ to resolve θ_3 for cost ∞
6 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1$
$(3 \to 4)$ for $l_c = 2$ under $l = 5$
$(1 \to 3)$ to resolve θ_6 for cost 8
7 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1$
$(1 \to 2)$ for $l_c = 0$ under $l = 3$
$(1 \to 3)$ to resolve θ_6 for cost 8
8* $D = \{\theta_9\}, \gamma = 3, \gamma^* = 1$
$(3 \to 4)$ for $l_c = 0$ under $l = 1$
$(3 \to 4)$ to resolve θ_9 for cost 5
9 $D = \{\theta_8\}, \gamma = 1, \gamma^* = 1$
$(4 \to 5)$ for $l_c = 0$ under $l = 1$
$(5 \to 6)$ to resolve θ_8 for cost 1
10 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1$
$(2 \to 3)$ for $l_c = 1$ under $l = 3$
$(1 \to 3)$ to resolve θ_6 for cost 8
11 $D = \{\theta_3, \theta_9\}, \gamma = \infty + 5, \gamma^* = 1$
$(1 \to 2)$ for $l_c = 0$ under $l = 2$
$(5 \to 6)$ to resolve $\{\theta_3, \theta_9\}$ for cost $\infty + 5$
12 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1$
$(4 \to 5)$ for $l_c = 2$ under $l = 3$
$(1 \to 3)$ to resolve θ_6 for cost 8
13 $D = \{\theta_4, \theta_6\}, \gamma = \infty + 8, \gamma^* = 1$
$(1 \to 2)$ for $l_c = 0$ under $l = 3$
$(5 \to 6)$ to resolve $\{\theta_4, \theta_6\}$ for cost $\infty + 8$
14 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1$
$(4 \to 5)$ for $l_c = 3$ under $l = 5$
$(1 \to 3)$ to resolve θ_6 for cost 8
15 $D = \{\theta_5, \theta_6\}, \gamma = \infty + 8, \gamma^* = 1$
$(1 \to 2)$ for $l_c = 0$ under $l = 3$
$(5 \to 6)$ to resolve $\{\theta_5, \theta_6\}$ for cost $\infty + 8$
16 $D = \{\theta_6\}, \gamma = 8, \gamma^* = 1$
$(5 \to 6)$ for $l_c = 4$ under $l = 5$
$(1 \to 3)$ to resolve θ_6 for cost 8
1* 4 6 14 16
12345
$(1 \to 3)$
$(2 \to 3)$
$(3 \to 4)$
$(4 \to 5)$
2* 23145
$(3 \to 4)$
3* 23415
5 13245
7 10 12 12435
$(4 \to 5)$
13 12453
15 12354
$(1 \to 3)$
$(2 \to 3)$
8* 24135
$(3 \to 4)$
9 24315
11 14235

Figure 3: The explored total orders when solving the motivating example by using GCDO. The levels of total orders and the chosen next order move are in blue; all the bounding constraints are in the solid-line box; the bounding constraints that are extracted by f during the search are in red; the manifested disjoint bounding constraints D, estimation cost γ, incumbent cost γ^*, the standard order move, and the first reducing order move at each iteration are given in the dotted-line boxes; we add $*$ to the iteration number if the exact cost is queried from the sub-solvers in that iteration.

of the latest visited children of $\mathcal{L}$ in the previous iterations. Intuitively, the level of a total order is the first event that is not in the right place compared to the same place as the root total order. For example, 12435 has level 3. Line 9 calculates the standard order move to the next total order, which is uniquely determined by $\mathcal{L}$ and l_c. Here an order move is an operation that right shifts the position of an event in a total order. Then, GCDO either moves to the next total order (Lines 13-14) or backtracks (Lines 16-17). The function NEXTMOVE is implemented by Equation 1 in Section 4.1

Extracting Bounding Constraints The algorithm extracts a set of bounding constraints and adds them to Θ (Line 7) when the true cost of a total order is queried (Line 6). Each bounding constraint is a set of partial orders and constraints that impose a certain amount of cost when satisfied. An example bounding constraint is $\theta_7 = (1 \prec 4) \wedge (2 \prec 5) \wedge s_1 \wedge s_3 \wedge s_4$ with cost 3. As partial orders $(1 \prec 4) \wedge (2 \prec 5)$ force the concurrency of Flow-A, Flow-C, and Flow-D, which cannot be transferred together, any total orders that imply these partial orders will have to at least relax constraint $s3$, which imposes cost 3. With bounding constraints Θ, we can estimate total order costs without using g (Line 4) and jump further than the standard move to prune inconsistent or suboptimal total orders (Lines 10-11). The implementation of function INITCB and the method construct bounding constraint extraction function f are introduced in Section 4.2.

Estimating Total Order Costs At each iteration, we start by calculating an optimistic cost estimation of $\mathcal{L}$ by using bounding constraints Θ (Line 4). Our estimation method combines multiple manifested bounding constraints of $\mathcal{L}$ to determine an informative lower bound estimation while being optimistic. As using g to calculate the true cost of $\mathcal{L}$ is computationally expensive (Line 6), we evaluate $g(\mathcal{L})$ only when the estimation is better than the incumbent cost (Line 5). The function ESTIMATECOST is implemented by Equation 4 in Section 4.3.

Algorithm 1: GCDO

```
   Input: $\langle n, \Phi^+, g, f \rangle$
   Output: $(\mathcal{L}^*, \gamma^*)$
1  $(\mathcal{L}, l_c) \leftarrow ((1, 2, .., n), 0)$;
2  $(\mathcal{L}^*, \gamma^*, \Theta) \leftarrow (\{\}, \infty, \textsc{InitBC}(\Phi^+)$;
3  while $\mathcal{L} \mathrel{!=} \{\}$ do
4      $\gamma \leftarrow \textsc{EstimateCost}(\Theta, \mathcal{L})$ ;
5      if $\gamma < \gamma^*$ then
6          $\gamma \leftarrow g(\mathcal{L})$ ;
7          $\Theta \leftarrow \Theta \cup f(\mathcal{L})$ ;
8          if $\gamma < \gamma^*$ then $(\mathcal{L}^*, \gamma^*) \leftarrow (\mathcal{L}, \gamma)$ ;
9      $(i' \to j') \leftarrow \textsc{NextMove}(\mathcal{L}, l_c)$ ;
10     $(i^\dagger \to j^\dagger) \leftarrow \textsc{FirstReducing}(\mathcal{L}, \Theta, \gamma^*)$ ;
11     if $ni' + j' < ni^\dagger + j^\dagger$ then $(i' \to j') \leftarrow (i^\dagger \to j^\dagger)$;
12     if $i' < n$ then
13         $l_c \leftarrow 0$ ;
14         $\mathcal{L} \leftarrow \mathcal{L} \oplus (i' \to j')$
15     else
16         $l_c \leftarrow \text{PLV}(\mathcal{L})$ // position of level event
17         $\mathcal{L} \leftarrow \textsc{Parent}(\mathcal{L})$ ;
18 return $(\mathcal{L}^*, \gamma^*)$;
```

Reduction-directed Order Moves With bounding constraints Θ, the GCDO algorithm computes the first reducing move, which is the first order move that jumps over the inconsistent or suboptimal total orders with respect to the incumbent γ^* (Line 10). The order move is then used to update the standard order move to jump further (Line 11). Finding such order moves is based on the observation that some partial orders, which manifest a set of bounding constraints with a total cost larger than the incumbent, can be persistent in some subtrees. Thus, these subtrees and total orders can be pruned without impairing completeness or suboptimality. The function FIRSTREDUCING is implemented by Equations 6-7 in Section 4.4.

4.1 Total Order Search

Now we introduce the total order tree (Ono and Nakano 2005) and a depth-first search strategy in this tree (Wang 2015). In the tree of events $E = \{1, 2, .., n\}$, nodes are total orders of E, and an edge is an operation of altering partial orders of a total order. This tree is rooted at the root total order $(1, 2, ..n)$ and constructed by expanding all the children of each total order. The tree expansion uses the notions of levels and order moves, which are defined as follows:

Definition 1 (Level)**.** The level of a total order $\mathcal{L} = (p_1, p_2, .., p_n) \neq (1, 2, .., n)$ is the minimal integer l such that $p_l \neq l$. The level of $(1, 2, .., n)$ is n.

Definition 2 (Order Move)**.** An order move $(i \rightarrow j)$ deletes p_i from a total order $\mathcal{L} = (p_1, p_2, .., p_n)$ and inserts it right after p_j to obtain a total order $\mathcal{L}'$. This operation is denoted as $\mathcal{L}' = \mathcal{L} \oplus (i \rightarrow j)$.

In this tree, we generate a child by right shifting an event that is less than its parent level. To generate all the children of a total order $\mathcal{L}$ with level, we apply $\mathcal{L} \oplus (i \rightarrow j)$ for every $i < l$ and $i < j \leq n$. This tree exactly includes all the total orders of a set of events, which is proved in (Ono and Nakano 2005). The total order tree of $E = \{1, 2, 3, 4\}$ is given as an example in Figure 4.

As we can see, the total order level decreases from parents to children. Meanwhile, since the total order tree constrains the feasible order moves with respect to the total orders' level, a portion of partial orders can be persistent in all the total orders in a subtree, which is summarized as Lemma 1:

Lemma 1. *For a total order with level l, order move can only right shift $i < l$ in its subtree, and the partial orders between events $\{l, l+1, .., n\}$ remain in its descendants.*

Based on this property, as long as we know the partial orders that lead to inconsistency or suboptimality, we can identify the subtrees that always have these partial orders, which can be safely pruned.

To search this tree, we follow the depth-first order: (1) from a total order, the algorithm first visits its children and then its siblings with the same level; (2) when its children and these siblings are exhausted, the algorithm backtracks to its parent; (3) The group of children with the lowest level are generated first, and within each group, children are generated in the order of right shifting children's level events until the right end; (4) the same-level siblings are also generated by right shifting their level events until the right end. The order of visiting all the total orders of four events is given in Figure 4. Formally, from a total order $\mathcal{L} = (p_1, p_2, .., p_n)$, when its latest visited child has level l_c, the next move $\textsc{NextMove}(i, j, \mathcal{L})$ is calculated as follows:

$$\begin{cases} (l_c + 1 \rightarrow l_c + 2) & (l_c < l - 1) \\ (\mathrm{PLV}(\mathcal{L}) \rightarrow \mathrm{PLV}(\mathcal{L}) + 1) & (l_c = l - 1) \end{cases}, \tag{1}$$

where $\mathrm{PLV}(\mathcal{L})$ is the position of the level of $\mathcal{L}$ in itself. Note that the feasible order moves under these two conditions lead to the children of $\mathcal{L}$ and its siblings with level l, respectively. By following Equation 1, $(i \rightarrow j)$ with lower $(ni + j)$ is taken first from a total order and $(ni+j) \leq (nl+n)$ holds for the feasible order moves. Feasible order moves are simply applied as Lines 13-14. When the returned move is $(n \rightarrow n + 1)$, which is infeasible and means all the children and same-level siblings are exhausted, the algorithm backtracks to the parent of $\mathcal{L}$ (Lines 16-17).

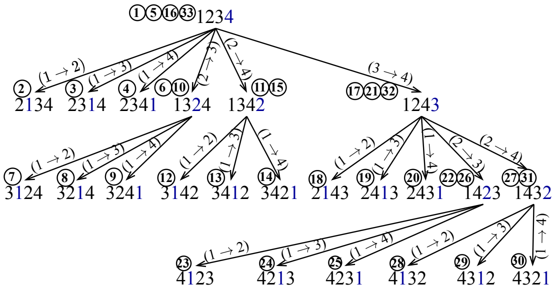

Figure 4: Total order tree of $E = \{1, 2, 3, 4\}$. The levels of total orders are in blue. The orderings of visiting these total orders by following Equation 1 are given in their upper left.

4.2 Extracting Bounding Constraints

In this section, we introduce the formal definition of bounding constraints and their extraction method. Intuitively, the bounding constraints are a set of partial orders along with the constraint that imposes a certain amount of cost when satisfied, which are the generalization of ordering conflicts to account for costs (Chen et al. 2019). They are also similar in spirit to valued nogoods (Dago and Verfaillie 1996) and bounding conflicts (Timmons and Williams 2020) by associating partial assignments with objective value bounds, but the bounding constraints are tailored to the ordering problems by replacing partial assignments with partial orders.

In our motivating example, Flow-A, Flow-C, and Flow-D cannot be transferred concurrently because of the limited bandwidth capacity of the two available paths. When a total order forces them to be concurrent, dropping Flow-C is the optimal relaxation since the other flows have higher priorities and thus higher costs to drop. As Flow-C starts at 2 and ends at 4, and Flow-A and Flow-D start at 1 and end at 5, this concurrency can be summarized as the partial orders $(1 \prec 4) \wedge (2 \prec 5)$, which compactly capture all the combinations of this concurrency: 1245, 1254, 2145, and 2154. As the state constraints of these flows are $\{s_1, s_2, s_3\}$, the fact that this concurrency imposes at least cost 3 can be summarized as a bounding constraint $\theta_7 = (1 \prec 4) \wedge (2 \prec 5) \wedge s_1 \wedge s_3 \wedge s_4$ with cost 3. Another bounding constraint example for state constraints is $\theta_6 = \mathrm{PO}(\theta_6) = (1 \prec 3) \wedge (1 \prec 4) \wedge (2 \prec 5) \wedge s_1 \wedge s_2 \wedge s_3 \wedge s_4$ with cost 1, which summarizes that the concurrency of all the flows has cost 8 by relaxing Flow-B and Flow-C.

In addition to the bounding constraints for state constraints, a bounding constraint example of temporal constraints is $\theta_8 = (3 \prec 1) \wedge (4 \prec 1) \wedge t_1 \wedge t_2 \wedge \wedge t_3 \wedge t_4 \wedge t_5$ with cost 1, which is manifested by total order 23415 as in Figure 2. As the partial orders $(3 \prec 1) \wedge (4 \prec 1)$ force both Flow-B and Flow-C to end before Flow-A and Flow-D start, the mission takes at least 80 seconds given temporal constraints $\{t_1, t_2, t_3, t_4\}$, which contradicts with time limit 70 seconds specified by t_5. As relaxing t_5 will remove this time limit with cost 1, which is the optimal relaxation since the other involved constraints are hard constraints, this bounding constraint has cost 1.

The third kind of bounding constraints is for ordering constraints, which is simpler than those for state and temporal constraints. For example, total order 12453 includes partial order $(5 \prec 3)$ and violates the ordering constraint $o_4 = (3 \prec 5)$ with cost ∞, which can be summarized as a bounding constraint θ_4 with cost ∞. Formally, we define the bounding constraints as follows:

Definition 3 (Bounding Constraints)**.** Let Φ be a set of constraints. A bounding constraint θ associated with cost $\textsc{cost}(\theta)$ is a conjunction of a set of partial orders $\text{PO}(\theta)$ and a set of constraints $\text{CS}(\theta) \subseteq \Phi$ such that $\textsc{cost}(\theta)$ is the cost sum of the optimal relaxation of $\text{CS}(\theta)$ under $\text{PO}(\theta)$. We say a total order $\mathcal{L}$ manifests θ if $\mathcal{L}$ implies $\text{PO}(\theta)$.

For ordering constraint $\phi_r^+ = \vee q_{rs}$, where q_{rs} is a partial order, we can obtain its corresponding bounding constraint θ_r by having $\text{PO}(\theta_r) = \wedge q_{rs}$, $\text{CS}(\theta_r) = \{\phi_r^+\}$, and $\textsc{cost}(\theta_r) = w(o_r)$. As this procedure is easy, we extract the bounding constraints for all the ordering constraints Φ^+ before search, which is implemented as function $\textsc{InitBC}$.

Now we introduce our method to construct bounding extraction function f, which finds PO, CS, and $\textsc{cost}$ of bounding constraints for state and temporal constraints. First, we extract PO for state and temporal constraints by using the same approach to extract ordering conflicts as (Chen et al. 2019): we represent the concurrency of state constraints as a polynomial number of partial orders; and we obtain the partial orders that are inconsistent with a set of temporal constraints by collecting the imposed partial orders in their negative cycle (Dechter, Meiri, and Pearl 1991). They are given in Equation 2 and Equation 3, respectively.

The partial orders PO^s to present the concurrency of multiple state constraints are:

$$\text{PO}^s = \underset{i,j}{\wedge} R^s_{ij} = \underset{i,j}{\wedge} (x_i^\vdash \prec x_j^\dashv). \tag{2}$$

where each $R^s_{ij} = (x_i^\vdash \prec x_j^\dashv) \wedge (x_j^\vdash \prec x_i^\dashv)$ represents the concurrency of two tasks (i.e., state constraints), and $x_i^\vdash$ and $x_i^\dashv$ are the start and end events of the i^{th} task.

When a negative cycle in a simple temporal network is found, the corresponding partial orders PO^t are:

$$\text{PO}^t = \underset{i}{\wedge} R^t_i = \underset{i}{\wedge} (x_i^- \prec x_i^+), \tag{3}$$

where $R^t_i = (x_i^- \prec x_i^+)$ represents the from event and to event of a temporal constraint that is involved in the cycle and added because of total ordering.

Then, CS in these two cases are the concurrent state constraints and all the temporal constraints involved in the negative cycle, respectively. Lastly, $\textsc{cost}$ can be obtained by associating the state and temporal constraints with costs and solving an optimization problem for an optimal relaxation. In our example, we use the solvers in (Chen et al. 2018) and (Yu and Williams 2013) to find the optimal relaxation with respect to the state and temporal constraints, respectively, and $\textsc{cost}$ is the cost sum of the relaxed constraints.

4.3 Estimating Total Order Costs

Given a set of bounding constraints Θ manifested by a total order $\mathcal{L}$, we can calculate γ^Θ, an optimistic, informative cost estimation of $\mathcal{L}$ by using Θ instead of evaluating the exact cost $g(\mathcal{L})$. The latter usually requires solving complex optimization problems, which is more computationally expensive and should be avoided as much as possible. For example, without using g, we know the costs of total orders that manifest θ_6 with $\text{PO}(\theta_6) = (1 \prec 3) \wedge (1 \prec 4) \wedge (2 \prec 5)$ such as 12435 and 12453 are at least 8.

An informative cost estimation of a total order should be as large as possible to lower bound its true cost. To be informative, we want to incorporate the information of multiple bounding constraints. For example, total order 12453 manifests both bounding constraints θ_5 with cost ∞ and θ_6 with cost 8. Therefore, the cost estimation of 12435 is lower bounded by $(\infty + 8)$ by summing their costs together.

Meanwhile, we still require the estimation to be optimistic when considering multiple bounding constraints. An optimistic cost estimation is a lower bound of the true cost. Thus, the constraint sets of the used bounding constraints should neither count the costs of unnecessary relaxations nor double count same relaxations. For example, total order 12435 manifests both bounding constraints θ_7 with cost 3 and θ_6 with cost 8. An optimistic cost estimation of 12435 is 8, which is determined by θ_6 instead of counting both θ_7 and θ_6. This is because they share s_3 (i.e., Flow-C) in their optimal relaxation, and its cost should be counted only once. We choose θ_6 with a higher cost to get a closer bound to its true cost.

Disjoint Bounding Constraints Formally, when we use the cost sum of multiple bounding constraints to obtain an informative estimation, the key to being optimistic is to use a set of disjoint bounding constraints, which is formally defined as follows:

Definition 4 (Disjoint Bounding Constraints)**.** Two bounding constraints θ_i and θ_j are disjoint if the intersection of their constraint sets $\text{CS}(\theta_i) \cap \text{CS}(\theta_j)$ only include hard constraints.

Given a set of disjoint bounding constraints D manifested by total order $\mathcal{L}$, we can estimate the cost of $\mathcal{L}$ as $\sum_{\theta_r \in D} \textsc{cost}(\theta_r)$. Based on Definition 4, we conclude the optimism of this estimation in Lemma 2.

Lemma 2. *Let $\mathcal{L}$ be a total order with cost γ and D be a set of disjoint bounding constraints manifested by $\mathcal{L}$. Let $\sum_{\theta_r \in D} \textsc{cost}(\theta_r) = k\infty + c$ be a cost estimation of $\mathcal{L}$. If $k = 0$, we have $c \leq \gamma$; otherwise, $\gamma \geq \infty$.*

Proof. When $k = 0$ for $k\infty + c$, the estimation is optimistic because: given any two disjoint bounding constraints θ_i and θ_j, we know the intersection of their constraint sets $\text{CS}(\theta_i) \cap \text{CS}(\theta_j)$ have only hard constraints, and thus soft constraints can not appear in the intersection of their relaxations. Therefore, the choice of the optimal relaxation for each bounding constraint over a set of soft constraints is independent and thus remains optimal. When $k > 0$ for $k\infty + c$, there must be a bounding constraint with cost ∞, which is manifested by total order $\mathcal{L}$, and thus $\mathcal{L}$ must be inconsistent and has at lest cost ∞. □

Finding Informative Optimistic Estimation Given a set of bounding constraints Θ manifested by total order $\mathcal{L}$, we calculate an informative estimation γ^{Θ} by choosing a set of disjoint bounding constraints D with the maximal cost sum:

$$\gamma^{\Theta} = \max_{(D\subseteq\Theta)\wedge(D \text{ is disjoint})} \sum_{\theta_r\in D} \text{COST}(\theta_r) \qquad (4)$$

As an implementation of ESTIMATECOST, we find such disjoint bounding orderings $D \subseteq \Theta$ and its cost γ^{Θ} by solving a weighted maximum clique problem (Bomze et al. 1999), which is NP-hard but can be solved very efficiently in practice (Cai and Lin 2016; Balas and Xue 1996). We first construct an undirected graph: the vertices are bounding constraint Θ, the weight of each vertex $\theta \in \Theta$ is $\text{COST}(\theta)$, and there is an undirected edge between two vertices $\theta_i, \theta_j \in \Theta$ if and only if they are disjoint as given in Definition 4. A clique D is a subset of vertices such that every two distinct vertices in this clique are adjacent, and a maximum clique is not a subset of any other clique. In the graph, each maximum clique D represents a set of disjoint bounding constraints. To be informative, we choose the clique with the maximum cost sum to have a tight bound.

4.4 Reduction-directed Order Moves

In this section, we introduce the method to find the next order move that jumps over inconsistent or suboptimal total orders given the incumbent cost. We first introduce our method to find the first resolving move of a bounding constraint θ, before which any total order still manifests θ and thus has at least cost $\text{COST}(\theta)$. By using these first resolving moves, we identify the first reducing move of a desired cost reduction, before which any total order still manifests a set of bounding constraints and is inconsistent or suboptimal.

First Resolving Move To find the first resolving move of a bounding constraint, we use the same method as resolving ordering conflicts in (Chen et al. 2019), which is based on Lemma 1 and finds the first order move that leads to a total order negating at least one partial order in its partial orders.

Consider total order 12453 that manifests bounding constraints $\{\theta_4, \theta_6\}$ in the thirteen iteration. We consider finding the first resolving move of one of its bounding constraints, θ_6, as an example. From 12453, the standard next move calculated by Equation 1 is $(1 \to 2)$ and leads to 21435, which still implies $\text{PO}(\theta_6) = (1 \prec 3) \wedge (1 \prec 4) \wedge (2 \prec 5)$. To resolve θ_6, we should jump over 21453 and directly move to 24153 through order move $(1 \to 3)$, which negates $(1 \prec 4)$ in $\text{PO}(\theta_6)$ and can reduce $\text{COST}(\theta_6)$. Thus, $(1 \to 3)$ is the first resolving move of θ from 12453. While any order move before $(1 \to 3)$ is nonhelpful, the order moves after it may also resolve θ_6 . For example, $(1 \to 4)$ moves to 24513, which negates both $(1 \prec 3)$ and $(1 \prec 4)$. Another example is $(2 \to 3)$. Even though the new total order 14253 still implies $\text{PO}(\theta_6)$, its subtree may contain total orders that resolve θ_6 such as taking $(1 \to 2)$ in subsequent.

We return infeasible moves $(n \to n+1)$ when there is no total order that resolves the bounding constraint among the descendants of the current total order, its same-level siblings, and the descendants of these siblings. Consider the other bounding constraint θ_4 manifested by 12453. As the level of 12453 is 3, the feasible order moves from 12453 can only right shift events 1 or 2 to generate its children or right shift 3 to generate its siblings. However, 3 has reached the right end, and right shifting 1 or 2 does not change partial order $\text{PO}(\theta_4) = (5 \prec 3)$. Moreover, the levels of the new total orders obtained by shifting 1 or 2 are no more than 2, and thus there is no chance to negate $(5 \prec 3)$ in the following moves by Lemma 1. Thus, the first resolving move of θ_4 is $(5 \to 6)$, which means backtrack.

Formally, consider a bounding constraint θ_r with partial orders $\text{PO}(\theta_r) = \bigwedge_s q_{rs}$ manifested by total order $\mathcal{L} = (p_1, p_2, ..., p_n)$ with level l. The first resolving order move of θ_r from $\mathcal{L}$ is

$$(i_r^{\dagger} \to j_r^{\dagger}) = \operatorname*{argmin}_{(i'\to j')\in\Pi_r} (ni' + j'), \qquad (5)$$

where $\Pi_r = \{(i' \to j') \,|\, ((p_{i'} \prec p_{j'}) \in \text{PO}(\theta_r)) \wedge (p_{i'} \leq l)\} \cup \{(n \to n+1)\}$. Order move $(n, n+1)$ is infeasible and means GCDO should backtrack as in Equation 1.

First Reducing Move Let $\Delta = \gamma^D - \gamma^*$ be the gap between an optimistic cost estimation γ^D given disjoint bounding constraints D and the incumbent cost γ^*. We need to find the first order move that resolves a set of bounding constraints $D' \subseteq D$ with enough cost reduction $\sum_{\theta\in D'} \text{COST}(\theta)$ such that $\sum_{\theta\in D'} \text{COST}(\theta) > \Delta$. The first order move to achieve this reduction Δ is called the first reducing move of Δ from $\mathcal{L}$ given D. From another perspective, any order move that is before this move must lead to the total orders whose costs are at least γ^* and thus not worth exploring.

In our previous example, we consider total order 12453 with manifested disjoint bounding constraints $\{\theta_4, \theta_6\}$ and incumbent cost 1 in the thirteen iteration. The first resolving move of θ_4 and θ_6 are $(1 \to 3)$ and $(5 \to 6)$, respectively. Recall that any order move before the first resolving move of a bounding constraint will lead to a subtree where this bounding constraint remains. Thus, as we identify reducing cost from $\infty + 8$ to be less than incumbent $\gamma^* = 1$ requires resolving both constraints, we choose the order move that jumps furthest, which is $(5 \to 6)$ in this example. Thus, the search can safely backtrack by considering these two bounding constraints together.

To identify the first reducing move of incumbent cost γ^* from $\mathcal{L}$ as an implementation of function FIRSTREDUCING, we start by calculating the first resolving move $(i_r^{\dagger} \to i_r^{\dagger})$ of each bounding constraint $\theta_r \in D$. Then, we sort all these resolving moves in the order of increasing $(ni_r^{\dagger} + i_r^{\dagger})$, which is the exploration order of the order moves as given in Equation 1. We also associate every move with a cost estimation G_r^D, which is as follows:

$$G_r^D = \sum_{k\in\{r+1,r+2,..R\}} \text{COST}(\theta_k), \qquad (6)$$

where R is the number of all the bounding constraints D. This cost estimation G_r^D is the cost sum of all the unresolved bounding constraints of $(i_r^{\dagger} \to i_r^{\dagger})$ from the current total order, which is optimistic following Lemma 2. Therefore,

#flows	GCDO					CDITO					MILP			
	t_1	γ_1	γ	η	ζ	t_1	γ_1	γ	η	ζ	t_1	γ_1	γ	η
5	0.029	2.30	**0**	**100**	0.087	**0.020**	2.30	**0**	**100**	0.763	0.062	4	**0**	**100**
10	0.043	2.87	**0.43**	**81**	0.086	**0.036**	2.87	1.52	9	0.751	0.273	8	0.73	74
15	0.096	3.96	**0.41**	**80**	0.043	**0.089**	3.96	3.04	3	0.727	1.055	12	1.14	41
20	0.172	5.18	**0.47**	**77**	0.008	**0.153**	5.18	4.27	0	0.706	1.791	16	1.09	52
25	0.301	6.12	**0.49**	**73**	0.006	**0.229**	6.12	5.24	0	0.681	3.516	20	1.33	47
30	0.460	7.24	**1.53**	**57**	0.004	**0.317**	7.24	6.54	0	0.676	4.949	24	1.95	19

Table 2: Experimental results. t_1: the average runtime to find the first solution; γ_1: the average cost of the first solution; γ: the average cost of the returned solutions. η: the number of the problems whose returned solutions are optimal. ζ: the ratio of the average times of calls to g to the average number of the explored total orders. We highlight the best results of t_1, γ, and η.

if an order move is before $(i_r^\dagger \to j_r^\dagger)$ and $G_{r-1}^D > \gamma^*$, it must lead to a total order or a subtree that is inconsistent or suboptimal, which can be safely pruned. Given the current incumbent cost γ^*, we identify the next resolving move to reduce the cost to be under γ^* as follows:

$$(i^\dagger \to j^\dagger) = \underset{(i_r^\dagger \to i_r^\dagger)\wedge G_r^D<\gamma^*}{\arg\min} (ni_r^\dagger + j_r^\dagger). \tag{7}$$

5 Experimental Results

In order to evaluate the effects of using bounding constraints in GCDO, we benchmarked GCDO on the optimal temporal network configuration problems similar to our motivating example, with different complexities and sizes. These problems involve routing flows and allocating bandwidth resources with respect to requirements on loss, delay, bandwidth against CDITO (Chen et al. 2019) and Mixed Linear Inter Programming (MILP) approach by using Gurobi (Gurobi Optimization 2021). CDITO can interact with a rich class of sub-solvers as our algorithm does. As CDITO is only aware of hard constraints to resolve ordering conflicts, we modified CDITO to keep exploring the solution space after finding the first solution and record the best solution as the incumbent. For both CDITO and GCDO, we use (Chen et al. 2019) and (Yu and Williams 2013) as sub-solvers for state and temporal constraints, respectively. The MILP encoding uses Boolean variables to indicate the precedence relations of events, and constraints and violating costs are conditional on these variables.

We use the same communication network simulator in (Chen et al. 2018) to generates network flow requirements on a meshed network. The major difference is that we associate each flow with a priority to be its relaxing cost. While one-fifth of the flows must be transferred, the relaxing costs of the others are 1. The other setup is as follows: (1) the mission horizon is 300s; (2) the meshed network has 6 nodes; (3) the loss, delay, and bandwidth of each link are uniformly generated from intervals [0.1,0.3]%, [0.1,0.3]s, and [500,1000]kbps; (4) the loss, delay, throughput, minimum duration of each network flow are uniformly generated from [0.1,0.3]%, [0.1,0.3]s, [600,1000]kbps, and [20,80]s; (5) the generator adds temporal constraints between randomly chosen events with a duration (0,100], and the number of temporal constraints is one-fifth of the number of flows;

We tested six scenarios of 5, 10, 15, 20, 25, and 30 flows with 100 trials each. The timeout was 30 seconds. We only include the trials in which consistent solutions exist.

Table 2 shows the experimental results. We observe that all GCDO and CDITO can find consistent solutions ten times faster than MILP. As GCDO reduces to CDITO before an incumbent solution is found except recording bounding constraints, their first solutions are exactly the same, and GCDO spends slightly more time on recording these constraints. In the first solutions, MILP basically drops all the optional flows. It can be seen that while GCDO then reduces at least 80% costs for most of the scenarios, CDITO can only achieve good final cost γ when $\#$flow ≤ 10 and fails to achieving more than 20% reduction for the other scenarios in 30s. This demonstrates that GCDO is capable of using suboptimality to efficiently guide the search for high-quality solutions. Meanwhile, the costs of the solutions returned by GCDO is at lest half of that of MILP for most cases. We also report η , the number of problems in which an optimal solution is proved, which requires the algorithms to exhaust the solution space. As GCDO is able to prove the optimality of the returned solutions for a large portion of problems in 30 seconds, CDITO fails to complete this task for most problems and MILP only performs well when $\#$flow ≤ 10. The calls to the sub-solvers dominate the runtime for both CDITO and GCDO given our observation that more than 95% runtime is spent on these calls. The major reason for GCDO being efficient is the use bounding constraints to estimate costs and jump over suboptimal total orders without explicitly calling sub-solvers. This can be seen from that the ratios ζ of GCDO and CDITO differ by two or three orders of magnitude, which means GCDO avoids a large number of unnecessary calls to the evaluation function and instead focuses on thoroughly exploring the solution space.

6 Conclusion

In this paper, we presented GCDO, a generalized conflict-directed search algorithm that efficiently solves the optimal ordering problems with tightly coupled temporal and state constraints. GCDO adopts the branch-and-bound to search a special total order tree and generalizes the ordering conflicts in CDITO to bounding constraints, which can summarize both inconsistency and suboptimality. Thus, GCDO is able to efficiently prune the inconsistent or suboptimal total orders and thus avoids expensive and unnecessary calls to the sub-solvers. In our experiments on optimal temporal network configuration problems generated by a communication network simulator, we empirically demonstrate the efficiency of GCDO over CDITO and a MILP encoding.

Acknowledgments

This work at Massachusetts Institute of Technology was supported by Kawasaki Heavy Industries, Ltd (KHI) under grant number 030118-00001. This article solely reflects the opinions and conclusions of its authors and not KHI or any other Kawasaki entity.

References

Balas, E.; and Xue, J. 1996. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Algorithmica* 15(5): 397–412.

Bomze, I. M.; Budinich, M.; Pardalos, P. M.; and Pelillo, M. 1999. The maximum clique problem. In *Handbook of combinatorial optimization*, 1–74. Springer.

Cai, S.; and Lin, J. 2016. Fast Solving Maximum Weight Clique Problem in Massive Graphs. In *IJCAI*, 568–574.

Chen, J.; Fang, C.; Muise, C.; Shrobe, H.; Williams, B. C.; and Yu, P. 2018. RADMAX: Risk And Deadline Aware Planning for Maximum Utility. In *AAAI Workshop on Artificial Intelligence for Cyber Security (AICS'18)*.

Chen, J.; Fang, C.; Wang, D.; Wang, A.; and Williams, B. 2019. Efficiently Exploring Ordering Problems through Conflict-Directed Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 97–105.

Dago, P.; and Verfaillie, G. 1996. Nogood recording for valued constraint satisfaction problems. In *Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence*, 132–139. IEEE.

De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1-3): 61–95.

Gurobi Optimization, I. 2021. Gurobi optimizer reference manual. http://www.gurobi.com. Accessed: 2021-2-17.

Lawler, E. L.; and Wood, D. E. 1966. Branch-and-bound methods: A survey. *Operations research* 14(4): 699–719.

Manne, A. S. 1960. On the job-shop scheduling problem. *Operations Research* 8(2): 219–223.

Ono, A.; and Nakano, S.-i. 2005. Constant time generation of linear extensions. In *FCT*, 445–453. Springer.

Schiex, T.; Fargier, H.; Verfaillie, G.; et al. 1995. Valued constraint satisfaction problems: Hard and easy problems. *IJCAI (1)* 95: 631–639.

Sebastiani, R.; and Trentin, P. 2020. OptiMathSAT: A tool for optimization modulo theories. *Journal of Automated Reasoning* 64(3): 423–460.

Timmons, E. M.; and Williams, B. C. 2020. Best-First Enumeration Based on Bounding Conflicts, and its Application to Large-scale Hybrid Estimation. *Journal of Artificial Intelligence Research* 67: 1–34.

Wang, D. 2015. *A Factored Planner for the Temporal Coordination of Autonomous Systems*. Ph.D. thesis, Massachusetts Institute of Technology.

Wang, D.; and Williams, B. 2015. tBurton: A Divide and Conquer Temporal Planner. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Yu, P.; and Williams, B. C. 2013. Continuously relaxing overconstrained conditional temporal problems through generalized conflict learning and resolution. In *Twenty-Third International Joint Conference on Artificial Intelligence*.