Network Flow Models for Robust Binary Optimization with Selective Adaptability

Merve Bodur
School of Mathematics, University of Edinburgh, Edinburgh EH9 3FD, UK, merve.bodur@ed.ac.uk

Timothy C. Y. Chan
Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario M5S 3G8, Canada, tcychan@mie.utoronto.ca

Ian Yihang Zhu
NUS Business School, National University of Singapore, Singapore 119245, ianyzhu@nus.edu.sg

Adaptive robust optimization problems have received significant attention in recent years, but remain notoriously difficult to solve when recourse decisions are discrete in nature. In this paper, we propose new reformulation techniques for adaptive robust binary optimization (ARBO) problems with objective uncertainty. Without loss of generality, we focus on ARBO problems with "selective adaptability", a term we coin to describe a common class of linking constraints between first-stage and second-stage solutions. Our main contribution revolves around a collection of exact and approximate network flow reformulations for the ARBO problem, which we develop by building upon ideas from the decision diagram literature. Our proposed models can generate feasible solutions, primal bounds and dual bounds, while their size and approximation quality can be precisely controlled through user-specified parameters. Furthermore, and in contrast with existing solution methods, these models are easy to implement and can be solved directly with standard off-the-shelf solvers. Through an extensive set of computational experiments, we show that our models can generate high-quality solutions and dual bounds in significantly less time than popular benchmark methods, often by orders of magnitude.

Key words: Robust Optimization, Mixed Integer Optimization, Decision Diagrams, Network Flow Models

1. Introduction

Robust optimization (RO) has become a well-established paradigm for modeling and solving decision-making problems under uncertainty. It has found diverse applications across a wide range of problem domains, and is particularly well-suited for environments where the distribution of parameters in a model are difficult to characterize or when there is a need to consider worst-case outcomes. While early research has predominantly focused on static RO models, where a single robust solution is generated for all possible parameter realizations (Bertsimas et al. 2013), there has been significant recent interest in *adaptive* robust optimization, where *recourse* decisions can be made once additional information is revealed (Yanıkoğlu et al. 2019).

Adaptive robust optimization problems, particularly those with discrete recourse variables, have numerous applications. Common examples include, but are not limited to, routing (Eufinger et al.

2020), scheduling (Yan and Kung 2018), facility location (Hanasusanto et al. 2015), network design (Álvarez-Miranda et al. 2015), batching (Bayram et al. 2022), assignment (Daş et al. 2020) and matching problems (McElfresh et al. 2019). Despite their importance, adaptive robust optimization models can be significantly more challenging to solve relative to their static counterpart. While static models can typically be reformulated and solved as tractable mixed-integer linear programming (MILP) problems, these reformulation techniques do not extend to adaptive models, where optimal discrete recourse decisions must be defined for each and every parameter value lying within an uncertainty set. Few solution methods exist for solving these models, and most require elaborate and carefully-tuned iterative algorithms (Zeng and Zhao 2013, Kämmerling and Kurtz 2020, Arslan and Detienne 2022). Significant attention has instead been placed on approximation techniques, particularly ones that yield MILP formulations as these can be easily implemented through standard commercial solvers (e.g., Hanasusanto et al. 2015).

In this paper, we consider adaptive robust binary optimization (ARBO) problems with objective uncertainty. We focus, without loss of generality, on ARBO problems with *selective adaptability*, a term that we coin to describe a class of linking constraints where first-stage variables fix the values of some recourse variables without restricting the rest; that is, the remaining recourse variables have the ability to adapt to new information. This distinct structure underlies a variety of planning and sequential decision-making tasks, and we highlight several examples in Section 3. More importantly, the particular structure of these linking constraints motivates the design of new reformulation techniques and solution approaches. In particular, we employ ideas from the decision diagram community (Castro et al. 2022) to reformulate these ARBO problems into various exact and approximate *constrained network flow models*. These models are flexible, easy to implement, can be solved directly using standard commercial solvers, and offer a number of computational advantages compared to existing methods.

While many common problems naturally fit the description of ARBO with selective adaptability, we emphasize that our focus on these problems comes without loss of generality. Specifically, we will show that any ARBO problem can be reformulated into one that has this property through the use of auxiliary variables. Thus, all reformulation techniques and models presented in this paper are relevant to the general class of adaptive robust binary problems with objective uncertainty.

A concise list of our main contributions are as follows:

1. We introduce ARBO models with selective adaptability, and show how the structure of the linking constraints can be exploited when we have a description of the convex hull of the recourse feasible space. By highlighting the connection between decision diagrams and this convex hull description, we show that the ARBO models can be reformulated as constrained network flow problems, where recourse decisions are represented by continuous flow along

certain links in a large capacitated network, and where capacity constraints are given by the values of first-stage decisions. We show that these models have MILP formulations and can thus be directly solved using standard commercial solvers.

2. We introduce three approximation techniques to generate smaller and more compact network flow models for large ARBO problems. The first two techniques utilize approximate decision diagrams to formulate inner and outer approximations of the recourse feasible space, and thus generate primal and dual bounds for a given ARBO problem, respectively. The third technique, which we term as a generalized multi-network flow model, pools together a collection of multiple constraints and networks to generate a dual bound. We also outline new methods for specifying the size and quality of these approximation models.
3. We examine the performance of our exact and approximate network flow formulations in two sets of numerical experiments spanning a robust project investment problem and a robust assignment problem. For smaller ARBO problems, we find that the exact flow formulations are small in size and can be solved efficiently. For larger ARBO problems, we show that approximate network flow formulations are substantially smaller and easier to generate. More importantly, these approximate formulations consistently generate near-optimal solutions and high-quality dual bounds in solution times that are orders of magnitude lower than exact formulations and benchmark models. Furthermore, by adjusting the size of the approximate models, we show that the complexity and solution times of the approximate models can be reduced substantially while sacrificing little in terms of solution quality, making these methods highly scalable for large ARBO problems.

The rest of the paper unfolds as follows. In Section 2, we review the relevant literature on robust optimization, decision diagrams, and network flow models. In Section 3, we introduce the ARBO model and the concept of selective adaptability. In Section 4, we present the exact network flow reformulation, while Section 5 presents a series of approximation techniques that result in more compact and tractable network flow formulations. Section 6 examines the performance of the network flow models across various numerical experiments. We conclude in Section 7.

We summarize key notation used in this paper. All vectors and matrices are in bold letters, while sets are denoted using calligraphic letters. Let $\mathcal{S}$ be a feasible set described using linear and integrality constraints. We use $\text{Relax}(\mathcal{S})$ to describe the relaxation of $\mathcal{S}$ obtained by removing integrality constraints, and $\text{Conv}(\mathcal{S})$ to denote the convex hull of $\mathcal{S}$; by definition, $\text{Conv}(\mathcal{S}) \subseteq \text{Relax}(\mathcal{S})$. The set of extreme points of any polyhedral set $\mathcal{S}$ is denoted by $\text{Ext}(\mathcal{S})$.

2. Literature Review

We first review the relevant literature for robust binary optimization with objective uncertainty. We then review the relevant literature on decision diagrams and network flow models. We refer

readers interested in a general overview of robust optimization to the comprehensive surveys by Bertsimas et al. (2013) and Yanıkoğlu et al. (2019).

2.1. Adaptive robust binary optimization

Static robust binary optimization (RBO) problems with objective uncertainty are prevalent and have been well-studied in the literature, and we refer to Kasperski and Zieliński (2017) and Buchheim and Kurtz (2018) for comprehensive surveys of these problems. To summarize, RBO problems are challenging, and even robust formulations of polynomially-solvable combinatorial optimization problems can be NP-hard (Buchheim and Kurtz 2018). Nonetheless, they can be formulated and solved as MILP models by reformulating the inner adversarial problem using standard optimality conditions. However, these reformulations do not extend to the adaptive setting, where recourse decisions must be defined for each and every parameter vector within the uncertainty set.

Few algorithms exist for solving general two-stage RBO problems with objective uncertainty. The first and most well-known approach is the nested constraint-and-column generation method proposed by Zeng and Zhao (2013). This algorithm iterates between a master problem and a subproblem, the latter iteratively adding variables and constraints into the master problem. The subproblem is itself a max-min problem, and is commonly solved using a decomposition algorithm (e.g., constraint-and-column generation) for each outer iteration. A second approach is the branch-and-price algorithm developed by Arslan and Detienne (2022). Within each node of a branch-and-bound search tree, the algorithm solves a series of iterative pricing problems to add columns that correspond to feasible second-stage solutions for a fixed first-stage solution generated by a master problem defined within each given node. Finally, Kämmerling and Kurtz (2020) define a specialized branch-and-bound algorithm that branches over first-stage solutions while using another algorithm to iteratively refine the dual bound in each node of the branch-and-bound tree. The first approach of nested constraint-and-column generation is known to scale poorly to larger instances, because each iteration of the algorithm adds a full copy of all variables and constraints of the recourse problem to the master problem (e.g., Dumouchelle et al. 2023). The algorithm also may not converge in a finite number of iterations, because binding constraints may exist in the interior of the uncertainty set (Arslan and Detienne 2022). On the other hand, the latter two approaches are iterative algorithms that require carefully-tuned intermediary steps and may be confined to specific software; for example, many commercial solvers limit the degree of user customization of specific branch-and-bound processes.

The challenges that surround these exact solution methods have motivated much interest in approximation techniques (e.g., Vayanos et al. (2011), Bertsimas and Georghiou (2015), Postek and Hertog (2016); see Arslan and Detienne (2022) for a comprehensive review). For example,

Bertsimas and Georghiou (2015) and Bertsimas and Dunning (2016) consider the use of piecewise constant decision rules to represent the binary recourse variables. On the other hand, Hanasusanto et al. (2015) and Subramanyam et al. (2020) propose the K-adaptability approximation method which pre-identifies K recourse solutions to implement in the second stage. The K-adaptability model, which can be solved as a MILP model, implicitly partitions the uncertainty set into a finite number of subsets for which a single recourse decision is assigned to each. In recent years, this MILP formulation has become a commonly used approximation method, particularly for the problem class that we will examine (e.g., Dumouchelle et al. (2023), Arslan and Detienne (2022)).

In this paper, we present a new set of exact and approximate MILP reformulations for ARBO problems with objective uncertainty. These formulations, which are presented in the form of constrained network flow problems, are flexible, easy to formulate and implement in standard solvers, and generate high-quality solutions and bounds in times that can be significant lower than alternative solution methods.

2.2. Decision diagrams and network flow models

Decision diagrams (DDs) are increasingly used to generate solutions and bounds for discrete optimization problems. With few exceptions, this literature focuses on deterministic problems (Bergman et al. 2016). We refer to Castro et al. (2022) for a comprehensive survey of this literature.

Several works have examined the use of decision diagrams for two-stage stochastic programming with binary recourse decisions. Lozano and Smith (2018), Guo et al. (2021), and MacNeil and Bodur (2023) consider various two-stage problems for which the recourse feasible space under each scenario can be represented as a decision diagram, making the two-stage problem amenable to standard Benders decomposition techniques that would otherwise only apply to problems with continuous recourse (Rahmaniani et al. 2017). In their problems, first-stage decision variables serve as capacity constraints in the diagrams, and given a first-stage decision, a shortest path algorithm can be applied on the diagram to generate a cut for the master problem. On the other hand, Serra et al. (2019) consider a monolithic formulation of a two-stage scheduling problem where a subset of constraints are replaced with DD-based network flow constraints. Integrality of the flows across these networks are then enforced using binary variables.

To our knowledge, there is only one application of decision diagrams for robust optimization. Lozano et al. (2022) consider the reformulation of static robust binary optimization models as DD-based constrained shortest path problems for which specialized algorithms can be employed.

In contrast to these works, we consider the first use of DD-based network flow models for adaptive robust optimization. We show that in our problem setting, the complete set of optimal recourse decisions given any first-stage solution can be represented as a continuous flow across a network

(i.e., rather than a flow along a single path). We extend the analysis by proposing several primal and dual bounding techniques using the concept of approximate decision diagrams (Castro et al. 2022). These are the first application of DD-based approximation schemes for robust optimization, and we show that they can lead to highly tractable and effective models in our numerical experiments.

Finally, we remark that decision diagrams are also closely related to the state transition graphs found in the dynamic programming literature; we refer to Hooker (2013) and Castro et al. (2022) for detailed comparisons of these two research areas. de Lima et al. (2022) provide a survey on dynamic programming-based network flow formulations for deterministic optimization problems. As the authors point out, one of the main advantages of these formulations is that they can be solved directly using standard MILP solvers, overcoming the necessity of more elaborate iterative methods. We consider exact and approximate DD-based network flow formulations for similar reasons, and show that they are also highly effective for our robust optimization problems.

3. Robust Binary Optimization with Selective Adaptability

In this section, we define the ARBO problem of interest, introduce the concept of selective adaptability, and derive several preliminary insights based on the structure of the ARBO problem.

3.1. Problem definition

Consider an ARBO problem of the form

$$\min_{\mathbf{x}\in\mathcal{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathcal{Y}\cap\mathcal{S}(\mathbf{x})} \mathbf{c}^\top\mathbf{x}+\boldsymbol{\xi}^\top\mathbf{y}, \tag{1}$$

where $\mathcal{X}\subseteq\{0,1\}^m$ defines the feasible set of binary first-stage decisions, $\Xi := \{\boldsymbol{\xi}\,|\,\boldsymbol{T}\boldsymbol{\xi}\leq\mathbf{d}\}$ is a bounded polyhedral uncertainty set, and $\mathcal{Y}\cap\mathcal{S}(\mathbf{x})\subseteq\{0,1\}^n$ defines the feasible set of binary recourse decisions where the set $\mathcal{Y} := \{\mathbf{y}\in\{0,1\}^n\,|\,\boldsymbol{G}\mathbf{y}\geq\mathbf{h}\}$ captures the constraints that do not depend on $\mathbf{x}$, while $\mathcal{S}(\mathbf{x})\subseteq\mathbb{R}^n$ models the linking constraints. In this paper, we focus on problems where $\mathcal{S}(\mathbf{x})$ satisfies a condition that we term *selective adaptability*.

Definition 1 (Selective Adaptability) *A linking constraint between first-stage and second-stage decision variables is defined to be selectively adaptive if it can be expressed as $y_i\leq x_j$, $y_i=x_j$ or $y_i\geq x_j$, for some $i\in\{1,\dots,n\}$ and $j\in\{1,\dots,m\}$. The ARBO problem (1) is said to have selective adaptability if all constraints defining $\mathcal{S}(\mathbf{x})$ are selectively adaptive, i.e.,*

$$\mathcal{S}(\mathbf{x}) = \left\{\mathbf{y}\in\mathbb{R}^n \;\middle|\; \begin{array}{ll} y_i\leq x_j & \forall(i,j)\in\mathcal{U}_1 \\ y_i=x_j & \forall(i,j)\in\mathcal{U}_2 \\ y_i\geq x_j & \forall(i,j)\in\mathcal{U}_3 \end{array}\right\}, \tag{2}$$

where $\mathcal{U}_1,\mathcal{U}_2,\mathcal{U}_3$ are disjoint sets, and $\mathcal{U}_1\cup\mathcal{U}_2\cup\mathcal{U}_3\subseteq\{1,\dots,n\}\times\{1,\dots,m\}$.

An ARBO problem with selective adaptability implies that the value of each binary recourse variable $y_i \in \{0,1\}$ is either (i) fixed by the values of first-stage decisions $\mathbf{x} \in \mathcal{X}$ (due to constraints corresponding to $\mathcal{U}_2$, and those associated with $\mathcal{U}_1$ and $\mathcal{U}_3$ if x_j is 0 or 1, respectively), or (ii) not restricted by the values of first-stage decisions $\mathbf{x} \in \mathcal{X}$ (since the constraints associated with y_i become redundant when $\mathcal{U}_2 = \emptyset$, and when $x_j = 1$ for all $(i,j) \in \mathcal{U}_1$ and $x_j = 0$ for all $(i,j) \in \mathcal{U}_3$). Practically, this implies that individual recourse variables not impacted by the first-stage decisions can be used to "adapt" to new information.

3.2. Models with selective adaptability

We give a few examples of ARBO problems with selective adaptability and then show that our examination of this problem structure comes without loss of generality. We begin with two examples of sequential decision-making problems that naturally exhibit this linking constraint structure:

- In a facility location problem, a planner can only make an assignment (or shipping) decision between a potential location j and an individual customer l if a facility (e.g., hub, warehouse, factory) has been placed at location j (Kämmerling and Kurtz 2020). Assuming that customer demand is given by $\boldsymbol{\xi}$, the sequential nature of this decision-making process can be represented by linking constraints of the form $y_{lj} \leq x_j$, where y_{lj} is a particular location-customer assignment decision and x_j determines whether a facility is placed in location j.
- In a project investment problem, an investor with a limited budget aims to maximize investment returns across two decision stages by investing in different projects where early-stage investments come with risk but higher return rates (Arslan and Detienne 2022). For each project $i \in \{1,\ldots,n\}$, the linking constraint $y_i \geq x_i$ captures early-stage commitment decisions. We revisit this problem in our numerical experiments in Section 6.

The concept of selective adaptability can also be used to as a framework for introducing limited degrees of flexibility into static robust binary optimization problems. These models can be used to narrow down the set of choices to consider before model parameters are known exactly. Specifically, given the static robust problem

$$\min_{\mathbf{y} \in \mathcal{Y}} \max_{\boldsymbol{\xi} \in \Xi} \quad \boldsymbol{\xi}^\top \mathbf{y},$$

we can define extensions where we have a *budget* on the level of adaptability. For example, consider problem (1) with $\mathcal{S}(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^{n=m} \,|\, \mathbf{y} \leq \mathbf{x}\}$, $\mathbf{c} = \mathbf{0}$, and $\mathcal{X} = \{\mathbf{x} \in \{0,1\}^m \,|\, \mathbf{w}^\top \mathbf{x} \leq \beta\}$ where $\mathbf{w}$ and β are non-negative. In this setting, β controls the budget on adaptability, where higher values imply that more recourse decisions are "available" in the second stage. For example, consider a daily route generation problem in a transportation context. If $\mathbf{w} = \mathbf{1}$ and $\beta = 0.1n$, where n represents the number of links in a road network, the corresponding model would identify the most important

10% of links (denoted by $\mathbf{x}$) for constructing real-time vehicle routes (denoted by $\mathbf{y} \in \mathcal{Y} \cap \mathcal{S}(\mathbf{x})$). Highlighting such subnetworks may be important for regular planning, communication, or training purposes. A similar problem is examined in the numerical experiments in Section 6.

Finally, we emphasize that any ARBO problem without selective adaptability can be reformulated into one that has this property by introducing additional auxiliary variables into the recourse problem. Specifically, let $\hat{\mathbf{F}}\mathbf{x} + \hat{\mathbf{G}}\mathbf{y} \leq \hat{\mathbf{h}}$ be a set of linking constraints that are not in the form of equation (2) and let $\hat{\mathcal{I}} \subseteq \{1, \ldots, m\}$ be the indices of the $\mathbf{x}$ variables that appear in these constraints. Now, let $\boldsymbol{y}^{\text{aux}} \in \{0,1\}^{|\hat{\mathcal{I}}|}$ be a new set of auxiliary recourse variables. For each $j \in \hat{\mathcal{I}}$, we can replace x_j in $\hat{\mathbf{F}}\mathbf{x} + \hat{\mathbf{G}}\mathbf{y} \leq \hat{\mathbf{h}}$ with the corresponding y_j^{aux} variable, making these constraints a part of $\mathcal{Y}$, and redefine the linking constraints as the set $\mathcal{S}(\mathbf{x}) = \{y_j^{\text{aux}} = x_j, \forall j \in \hat{\mathcal{I}}\}$. This reformulation technique is inspired by Arslan and Detienne (2022), and implies that all results derived in our paper are applicable to any ARBO problem, with or without selective adaptability.

3.3. Preliminary model insights

Problem (1), which is a min-max-min optimization problem, can be rewritten as the infinite-dimensional MILP model

$$\min_{v,\mathbf{x},\mathbf{y}^\xi} \quad \mathbf{c}^\top \mathbf{x} + v \tag{3a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}^\top \mathbf{y}^\xi \leq v, \quad \forall \boldsymbol{\xi} \in \Xi \tag{3b}$$

$$\mathbf{y}^\xi \in \mathcal{Y}, \quad \forall \boldsymbol{\xi} \in \Xi \tag{3c}$$

$$\mathbf{y}^\xi \in \mathcal{S}(\mathbf{x}), \quad \forall \boldsymbol{\xi} \in \Xi \tag{3d}$$

$$\mathbf{x} \in \mathcal{X}, \tag{3e}$$

where v represents the worst-case recourse objective value, and where a set of variables ($\mathbf{y}^\xi$) and constraints ((3b) - (3d)) must be defined for every $\boldsymbol{\xi} \in \Xi$.

Zeng and Zhao (2013) proposed a nested constraint-and-column generation to iteratively refine an approximation of this infinite-dimensional MILP formulation. In their approach, the set Ξ in model (3) is replaced with a finite set $\hat{\Xi}$ which is iteratively enlarged by solving a subproblem that computes points in Ξ to add to the set $\hat{\Xi}$. However, since model (3) is a complex combinatorial problem even for very small sets $\hat{\Xi} \subset \Xi$, the approach, which adds an additional copy of variables and constraints to the model at each iteration, can quickly become intractable. Furthermore, as discussed in Section 2.1, the nested constraint-and-column generation algorithm may be challenging to implement, and finite convergence may be neither quick nor guaranteed.

In this paper, we begin by showing that model (3) permits a more tractable MILP reformulation when $\mathcal{S}(\mathbf{x})$ satisfies selective adaptability. This reformulation is shown in the next proposition, which relies on the following lemma. All proofs can be found in the Electronic Companion.

Lemma 1 $\text{Conv}(\mathcal{Y} \cap \mathcal{S}(\mathbf{x})) = \text{Conv}(\mathcal{Y}) \cap \mathcal{S}(\mathbf{x}), \quad \forall \mathbf{x} \in \{0,1\}^m$.

Proposition 1 *If Problem (1) has selective adaptability, then model (3) can be reformulated as shown, where* $\mathbf{y}$ *becomes a vector of continuous variables:*

$$\min_{v,\mathbf{x},\mathbf{y}} \quad \mathbf{c}^\top \mathbf{x} + v \tag{4a}$$
$$\text{s.t.} \quad \boldsymbol{\xi}^\top \mathbf{y} \leq v, \quad \forall \boldsymbol{\xi} \in \Xi \tag{4b}$$
$$\mathbf{y} \in \text{Conv}(\mathcal{Y}) \tag{4c}$$
$$\mathbf{y} \in \mathcal{S}(\mathbf{x}) \tag{4d}$$
$$\mathbf{x} \in \mathcal{X}. \tag{4e}$$

Proposition 1 is important for two main reasons, and motivates the rest of this paper. First, model (4) is equivalent to a static RO problem, i.e., where both $\mathbf{x} \in \{0,1\}^m$ and $\mathbf{y} \in \mathbb{R}^n$ represent "first-stage" decision variables. Practically, this means that we can solve the original ARBO problem as a MILP model if we have a polyhedral representation of $\text{Conv}(\mathcal{Y})$. Section 4 focuses on the use of network flow constraints in an extended space to describe $\text{Conv}(\mathcal{Y})$. Second, Proposition 1 provides intuition for developing approximation schemes. In particular, model (4) implies that replacing $\text{Conv}(\mathcal{Y})$ with an inner-(or outer-) approximation will generate a valid primal (or dual) bound on the optimal value of (4). For example, replacing constraint (4c) with the continuous relaxation of $\mathcal{Y}$, defined as $\text{Relax}(\mathcal{Y}) := \{\mathbf{y} \in [0,1]^n \,|\, \boldsymbol{G}\mathbf{y} \geq \mathbf{h}\}$, generates a lower bound on the optimal value of model (4). Approximation schemes will be the focus of Section 5.

Finally, there may naturally exist problems where the relaxation of $\mathcal{Y}$ is an integral polytope, i.e., $\text{Relax}(\mathcal{Y}) = \text{Conv}(\mathcal{Y})$. One example is the budgeted adaptive routing problem mentioned in Section 3.2, if we assume that the routing problem is a shortest path problem. In this setting, the adaptive robust problem can solved directly as a monolithic MILP model.

Proposition 2 *If* $\text{Relax}(\mathcal{Y}) = \text{Conv}(\mathcal{Y})$*, then model (3) is equivalent to*

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y},\boldsymbol{\lambda}} \quad & \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \boldsymbol{\lambda} \\ \text{s.t.} \quad & \boldsymbol{T}^\top \boldsymbol{\lambda} = \mathbf{y} \\ & \boldsymbol{G}\mathbf{y} \geq \mathbf{h} \\ & \mathbf{y} \in \mathcal{S}(\mathbf{x}) \\ & \mathbf{x} \in \mathcal{X}, \quad \mathbf{y} \in [0,1]^n, \quad \boldsymbol{\lambda} \geq \mathbf{0} \end{aligned} \tag{5}$$

by reformulating the uncertainty set using duality conditions (Gorissen et al. 2015).

We conclude this section with two remarks on the generalizability of the results in this paper when certain modeling assumptions are relaxed.

Remark 1 *First-stage decisions can be mixed-integer, i.e.,* $\mathcal{X} \subseteq \{0,1\}^{m_1} \times \mathbb{Z}^{m_2} \times \mathbb{R}^{m_3}$*, as long as only the binary variables defining* $\mathcal{X}$ *appear in the linking constraints* $\mathcal{S}(\mathbf{x})$*.*

Remark 2 *The uncertainty set* Ξ *can be any convex set. For example, if* Ξ *is an ellipsoidal uncertainty set, then the MILP formulations we derive will become mixed-integer quadratic programs.*

4. Constrained Network Flow Reformulations

In this section, we present a MILP formulation for model (4) when $\text{Relax}(\mathcal{Y}) \neq \text{Conv}(\mathcal{Y})$. Specifically, in Section 4.1 we introduce decision diagrams and the corresponding network flow formulation used to obtain a polyhedral description of $\text{Conv}(\mathcal{Y})$. Then, in Section 4.2, we integrate the formulation with first-stage decisions.

4.1. Reformulating the recourse feasible space

We first describe a general procedure for obtaining a decision diagram encoding of the feasible solutions in $\mathcal{Y}$, then represent this diagram using a network flow formulation.

4.1.1. Decision diagram formulation. A binary decision diagram (BDD) is a graphical structure that can be used to encode the feasible space of a binary optimization problem (Bergman et al. 2016). Specifically, a BDD is a directed acyclic graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ with nodes $\mathcal{V}$ and arcs $\mathcal{A}$. The nodes are partitioned into $n+1$ non-empty layers $\mathcal{V} = (\mathcal{V}_1, \dots, \mathcal{V}_{n+1})$, while the directed arcs connect nodes in adjacent layers from $\mathcal{V}_i$ to $\mathcal{V}_{i+1}$ for $i \in \{1, \dots, n\}$. The sets $\mathcal{V}_1 = \{\mathtt{r}\}$ and $\mathcal{V}_{n+1} = \{\mathtt{t}\}$ are each composed of a single node, defined as the root node and terminal node, respectively. Each arc in the network has a label of either zero or one, and $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ define the subset of zero and one arcs leaving nodes in $\mathcal{V}_i$, respectively. A decision diagram $\mathcal{D}$ is a valid representation of a feasible space $\mathcal{Y}$ if each path from root node $\mathtt{r}$ to terminal node $\mathtt{t}$ in $\mathcal{D}$ can be mapped to a solution $\mathbf{y} \in \mathcal{Y}$, and vice versa. This mapping of path to solution is defined by the zero-one label on each arc in the path. Specifically, for any arc j in the path, if $j \in \mathcal{A}_i^0$ then $y_i = 0$, or if $j \in \mathcal{A}_i^1$ then $y_i = 1$. For example, a decision diagram of $\mathcal{Y} := \{\mathbf{y} \in \{0,1\}^5 \,|\, y_1 + y_2 + 2y_3 + 2y_4 + 3y_5 \leq 4\}$ is given in Figure 1. Finally, we note that the definition of BDDs in the literature generally includes arc weights that correspond to the value of objective coefficients of an optimization problem. Since these coefficients are not deterministic in our problem, we instead consider only "unweighted" decision diagrams.

Decision diagrams are closely related to the state-transition graph in the dynamic programming literature (Hooker 2013). In particular, nodes and arcs in the decision diagram can be mapped to "states" and feasible "actions" of a recursive formulation where decisions are made sequentially. Many binary optimization problem structures admit simple recursive formulations that can be used to obtain decision diagrams; we refer to Bergman et al. (2016), Bergman et al. (2022) and de Lima et al. (2022) for a comprehensive summary of recursive formulations for a variety of problem

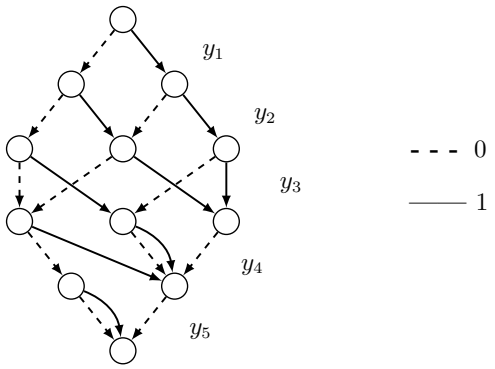

Figure 1 **A decision diagram with six layers, where the zero-one label of each arc is indicated using a dashed or solid line.**

structures. In general, a recursive formulation of a deterministic binary optimization problem can be written as a Bellman equation of the form

$$V_i(\boldsymbol{S}) = \max_{y_i \in Q_i(\boldsymbol{S})} \left\{ f_i(\boldsymbol{S}, y_i) + V_{i+1}(T_i(\boldsymbol{S}, y_i)) \right\}, \tag{6}$$

where $\boldsymbol{S}$ denotes the state of the system, $\mathcal{Q}_i(\boldsymbol{S}) \subseteq \{0,1\}$ denotes the set of feasible actions at stage i for state $\boldsymbol{S}$, $T_i(\boldsymbol{S}, y_i)$ defines the new state after taking action y_i, and the pair $V_i(\boldsymbol{S})$ and $f_i(\boldsymbol{S}, y_i)$ are used to capture the long-term values and immediate rewards of taking specific actions, respectively. However, since we only need to generate unweighted decision diagrams, we only need the state-transition graph corresponding to the states $\boldsymbol{S}$ and feasible actions $Q_i(\boldsymbol{S})$. To define the diagram, we first modify $T_n(\boldsymbol{S}, y_i)$ to be $T_n(\boldsymbol{S}, y_i) = \{\mathtt{t}\}$, and then map each state to a node and each feasible action $y_i \in Q_i(\boldsymbol{S})$ to a unique arc with label y_i that links $\boldsymbol{S}$ and $T_i(\boldsymbol{S}, y_i)$. This creates a diagram with a single root node $\mathtt{r}$ and terminal node $\mathtt{t}$, where each path from $\mathtt{r}$ to $\mathtt{t}$ corresponds to a sequence of actions in the recursive formulation.

Example 1 *Consider a feasible set* $\mathcal{Y} = \left\{ \mathbf{y} \in \{0,1\}^n \,\middle|\, \sum_{i=1}^n g_i y_i \leq h \right\}$ *that is defined by a knapsack constraint. The recursive formulation of* $\mathcal{Y}$ *is defined by feasible actions* $\mathcal{Q}_i(\boldsymbol{S}) = \{y_i \in \{0,1\} \,|\, \boldsymbol{S} + g_i y_i \leq h\}$*, state-transition function* $T_i(\boldsymbol{S}, y_i) = \boldsymbol{S} + g_i y_i$ *with an initial state* $\boldsymbol{S}_1 = 0$*. Each stage* i *in the recursive formulation thus corresponds to a layer* i *in the diagram. Each state in layer* i *represents the total amount of capacity used by the selection of items among* $1, \ldots, i$ *and feasible actions correspond to whether an additional item* $i+1$ *can be placed in the knapsack given the current state. To obtain a single terminal node in the decision diagram, we merge all nodes in layer* $n+1$*, which is equivalent to replacing* $T_n(\boldsymbol{S}, y_n) = \boldsymbol{S} + g_n y_n$ *with* $\mathcal{T}_n(\boldsymbol{S}, y_n) = h$ *in the recursive formulation. Note that this change in the state-transition matrix at the final stage* n *does not change the set of feasible actions in the recursive formulation.*

Finally, we note for any given decision diagram, it may be possible to obtain a *reduced* version that encodes the same set of feasible solutions in a much fewer number of nodes and arcs. Figure 1 is a reduced decision diagram for the knapsack set defined earlier. To obtain reduced decision diagrams, we can use a simple bottom-up merging technique (Bryant 1992, Bergman et al. 2016). Starting with nodes in layer n, we can merge any nodes which have the same set of outgoing arc types and destinations, repeating this procedure from layers $n-1$ to 1.

4.1.2. Recourse network flow formulation. Let $\mathcal{D}$ denote a decision diagram representation of $\mathcal{Y}$. Following the notation of Castro et al. (2022), we use $\text{NF}(\mathcal{D})$ to denote the network flow model of $\mathcal{D}$, which relates arc flows $\mathbf{z}$ to values of $\mathbf{y} \in \mathcal{Y}$ in the recourse problem. Specifically, let $\text{NF}(\mathcal{D})$ be defined as

$$\text{NF}(\mathcal{D}) := \left\{ (\mathbf{y}, \mathbf{z}) \in \mathbb{R}^n \times \mathbb{R}_+^{|\mathcal{A}|} \;\middle|\; \boldsymbol{A}\mathbf{z} = \mathbf{b},\ y_i = \sum_{j \in \mathcal{A}_i^1} z_j,\ \forall i \in \{1, \dots, n\} \right\}, \tag{7}$$

where $\boldsymbol{A} \in \{-1, 0, 1\}^{|\mathcal{V}| \times |\mathcal{A}|}$ denotes the node-arc incidence matrix and $\mathbf{b}$ a vector of zeros with the exception of $b_1 = -1$ and $b_{|\mathcal{V}|} = 1$. The constraint set $\boldsymbol{A}\mathbf{z} = \mathbf{b}$ defines standard flow conservation constraints. The second set of constraints link the value of each variable y_i to the sum of total flow over the one-arcs $\mathcal{A}_i^1$ in layer $i \in \{1, \dots, n\}$.

4.2. A complete network flow reformulation

A key property of $\text{NF}(\mathcal{D})$ is that its projection onto the variables $\mathbf{y}$, denoted by $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}))$, is equal to the convex hull of $\mathcal{Y}$ (Castro et al. 2022). We can use this property to derive an MILP formulation of Problem (1), as shown in the next proposition.

Proposition 3 *Problem* (1) *can be reformulated into the constrained network flow problem*

$$\min_{\mathbf{x},\mathbf{y},\mathbf{z},\boldsymbol{\lambda}} \quad \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \boldsymbol{\lambda} \tag{8a}$$

$$\text{s.t.} \quad \boldsymbol{T}^\top \boldsymbol{\lambda} = \mathbf{y} \tag{8b}$$

$$y_i = \sum_{j \in \mathcal{A}_i^1} z_j \quad \forall i \in \{1, \dots, n\} \tag{8c}$$

$$\boldsymbol{A}\mathbf{z} = \mathbf{b} \tag{8d}$$

$$\mathbf{y} \in \mathcal{S}(\mathbf{x}) \tag{8e}$$

$$\mathbf{x} \in \mathcal{X},\ \mathbf{z} \geq \mathbf{0}. \tag{8f}$$

Model (8) is thus an exact MILP reformulation of any ARBO problem with selective adaptability. As we will show in Section 6, this model can be tractably solved for problems of smaller sizes.

Example 2 *Consider the adaptive robust knapsack problem*

$$\begin{aligned} \min_{\mathbf{x}\in\{0,1\}^5} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}} \quad & \mathbf{c}^\top \mathbf{x} + \boldsymbol{\xi}^\top \mathbf{y} \\ & \mathbf{y} \le \mathbf{x} \\ & \mathbf{y} \in \mathcal{Y} := \left\{ \mathbf{y} \in \{0,1\}^5 \,\middle|\, y_1 + y_2 + 2y_3 + 2y_4 + 3y_5 \le 4 \right\}. \end{aligned} \tag{9}$$

Recall that the decision diagram for $\mathcal{Y}$ is illustrated in Figure 1. Suppose $\Xi = \left\{\boldsymbol{\xi} \,|\, \boldsymbol{\xi} \ge \boldsymbol{\xi}^0, \left\|\boldsymbol{\xi} - \boldsymbol{\xi}^0\right\|_1 \le \delta\right\}$. Then, problem (9) can be reformulated into model (8), where $\boldsymbol{A}$ is the node-arc incidency matrix of the decision diagram given in Figure 1, and

$$\boldsymbol{T} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} -\xi_1^0 \\ -\xi_2^0 \\ -\xi_3^0 \\ -\xi_4^0 \\ -\xi_5^0 \\ (\boldsymbol{\xi}^0)^\top \mathbf{1} + \delta \end{bmatrix}, \quad \begin{aligned} y_1 &= z_2 \\ y_2 &= z_4 + z_6 \\ y_3 &= z_8 + z_{10} + z_{12} \\ y_4 &= z_{14} + z_{16} \\ y_5 &= z_{19}. \end{aligned}$$

The indices $\{2, 4, 6, 8, 10, 12, 14, 16, 19\}$ correspond to the solid lines in Figure 1 when labeled from left to right in each layer, starting with the first layer.

5. Network Flow Approximations

In this section, we use approximate decision diagrams to derive new approximation methods for large-scale ARBO problems. We first define restricted and relaxed decision diagrams, which we then use to propose compact network flow models that generate first-stage solutions, primal bounds and dual bounds, respectively. We then present a multi-network flow model, which serves as a more general framework for generating dual bounds. Finally, we conclude the section by outlining a procedure for evaluating the quality of any computed first-stage solution.

5.1. Approximate decision diagrams

A restricted decision diagram of $\mathcal{Y}$ contains paths that map to a strict subset of the feasible solutions in $\mathcal{Y}$. On the other hand, a relaxed decision diagram contains paths that map to a superset of solutions that include all solutions in $\mathcal{Y}$. Restricted and relaxed decision diagrams can be generated by merging states in the recursive formulation of $\mathcal{Y}$ (Castro et al. 2022). Next, we outline two general "top-down" approaches for building restricted and relaxed decision diagrams.

5.1.1. Merging with width-based thresholds. In Algorithm 1 we present a common approach that merges nodes whenever a threshold on the diagram's "width" has been exceeded. Specifically, each layer in the diagram is constrained to have a width of at most W, that is, there can be at most W nodes per layer. Considering the recursive formulation, this width constraint is equivalent to having at most W number of different states at stage i. Note that if $W = \infty$ in Algorithm 1, then an exact decision diagram will be generated.

Algorithm 1 Width-Based Decision Diagram Construction Procedure

Input: A recursive formulation of $\mathcal{Y}$, a width parameter W

Output: A decision diagram $\mathcal{D}$

1: Initialization: let $\boldsymbol{S}_1(\mathtt{r})$ denote the initial state of the system (at root node $\mathtt{r}$), $\mathcal{V}_1 = \{\mathtt{r}\}$, $\mathcal{V}_2, \ldots, \mathcal{V}_n = \emptyset$, $\mathcal{V}_{n+1} = \{\mathtt{t}\}$, $\mathcal{A} = \emptyset$
2: **for** $i \in \{1, \ldots, n-1\}$ **do**
3: **for** $u \in \mathcal{V}_i$ **do**
4: **for** $y_i \in Q_i(\boldsymbol{S}(u))$ **do**
5: **if** $\exists\ u' \in \mathcal{V}_{i+1}$ where $\boldsymbol{S}(u') = T_i(\boldsymbol{S}, y_i)$ **then** add arc from u to u' with label y_i
6: **else** add node u' with state $T_i(\boldsymbol{S}, y_i)$ to $\mathcal{V}_{i+1}$ and add arc from u to u' with label y_i
7: **if** $|\mathcal{V}_{i+1}| > W$ **then**
8: $\hat{\mathcal{V}}_{i+1} \leftarrow \mathsf{select}(\mathcal{V}_{i+1}, |\mathcal{V}_{i+1}| - W + 1)$
9: $\mathsf{merge}(\hat{\mathcal{V}}_{i+1})$
10: **for** $u \in \mathcal{V}_n$ **do**
11: **for** $y_i \in Q_i(\boldsymbol{S}(u))$ **do** add arc from u to $\mathtt{t}$ with label y_i
12: **Reduce** $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
13: **return** $\mathcal{D}$

In step 8, the algorithm requires a rule for selecting $(|\mathcal{V}_{i+1}| - W + 1)$ nodes from the set $\mathcal{V}_{i+1}$. Random selection is the simplest and most commonly used procedure. Nonetheless, designing node selection procedures have become a more active topic of research in recent years (e.g., van Hoeve 2022), although such procedures focus on specific classes of deterministic problems.

Once this subset of nodes $\hat{\mathcal{V}}_i$ has been selected, they can be merged to create either restricted or relaxed decision diagrams. To create relaxed decision diagrams, merging operators must assign a new state to each merged node such that all subsequent feasible actions in the original recursive formulation remain feasible under this new state (Hooker 2013). Similarly, a restricted decision diagram can be created by assigning a new state to the merged node such that a subset of feasible actions in the original recursive are retained, without introducing infeasible actions. We give an example below.

Example 3 *Consider a feasible space defined by a single knapsack constraint (see Example 1). A restricted decision diagram can be generated by merging the nodes $\hat{\mathcal{V}}_{i+1}$ and assigning the new node a state that is the minimum value of the states in the merged set. A relaxed decision diagram can be generated by assigning the new node the maximum value of the states in the merged set.*

We note that the node merge operation can be performed while building a layer, rather than after the layer is completely built, to reduce the memory requirement, if desired. Furthermore, we note that we can also generate restricted diagrams simply by discarding nodes rather than merging

them in Algorithm 1, since we are effectively removing all feasible actions associated with that state-stage combination in the recursive formulation. Finally, step 12 is to reduce the size of the decision diagram using the bottom-up approach described at the end of Section 4.1.1.

5.1.2. Merging with distance-based thresholds. We now propose a merging approach based on the similarity of states in each layer, that is, we merge two nodes only when their respective state values are within some "distance" of each another. Algorithm 2 outlines this approach, which revolves around a partitioning of nodes in each layer into subgroups, where nodes in each subgroup are then merged. Any suitable set partitioning method can be used, as long two conditions are met: (i) within each subgroup, some pre-defined notion of distance between any pair of states does not exceed a user-specified parameter Q, and (ii) for any two subgroups, there exists a pair of nodes, one node in each subgroup, where the distance exceeds Q. The latter condition ensures the fewest number of partitions given Q. To the best of our knowledge, we are the first to outline and implement this distance-based approach for generating approximate decision diagrams.

Algorithm 2 Distance-Based Decision Diagram Construction Procedure

Input: A recursive formulation of $\mathcal{Y}$, a distance function $d(\cdot,\cdot)$ and distance parameter Q

Output: A decision diagram $\mathcal{D}$

1: Initialization: let $\boldsymbol{S}_1(\mathtt{r})$ denote the initial state of the system (at root node $\mathtt{r}$), $\mathcal{V}_1 = \{\mathtt{r}\}$, $\mathcal{V}_2, \ldots, \mathcal{V}_n = \emptyset$, $\mathcal{V}_{n+1} = \{\mathtt{t}\}$, $\mathcal{A} = \emptyset$
2: **for** $i \in \{1, \ldots, n-1\}$ **do**
3: Steps 3 to 6 from Algorithm 1
4: $\hat{\mathcal{V}}_{i+1,1}, \hat{\mathcal{V}}_{i+1,2}, \ldots \leftarrow \mathsf{partition}(\mathcal{V}_{i+1}, Q)$
5: **for** $\hat{\mathcal{V}}_{i+1} \in \{\hat{\mathcal{V}}_{i+1,1}, \hat{\mathcal{V}}_{i+1,2}, \ldots\}$ **do**
6: $\mathsf{merge}(\hat{\mathcal{V}}_{i+1})$
7: Steps 10 to 13 from Algorithm 1

Example 4 *We give an example of a* partition *function (step 4 of Algorithm 2). Consider the feasible space defined in Example 3, for which we can define the following node partitioning procedure: (i) order nodes according to state values, from smallest to largest, (ii) starting with the smallest state, create a new subgroup for the next node if and only if its state is more than Q units larger than the smallest state in the current subgroup. Here, distance between states is simply defined as the absolutely difference between state values.*

The distance-based merging approach is motivated by our initial attempts at using width-based merging in preliminary experiments. We found that it was difficult to finely tune the size of the diagram and quality of the approximation through the use of the width threshold parameter W.

In particular, when the width threshold is exceeded, it is often exceeded by a large margin, which typically results in the merging of hundreds or thousands of nodes into a single node. Furthermore, because the width constraint must be satisfied at each layer, there are essentially no restrictions on which nodes can or cannot be merged. In contrast, the key advantage of the distance-based approach is that both the size and the quality of the approximation is closely tied to the value of Q. In distance-based merging, Q determines precisely which nodes in each layer can and cannot be merged based on the similarity of their states. Furthermore, at each layer, there may be many subgroups but the number of nodes to be merged within each subgroup may be small. In general, we find that this distance-based merging procedure leads to approximations of higher quality and more precision in tuning the size and tractability of the corresponding models.

Finally, we remark that the distance-based approach can be further customized to generate even more refined approximations, for example, by merging subgroups with some probability $p < 1$. This additional feature could help generate approximations that are in-between those generated solely by incrementally increasing the value of Q (which could be discrete, for example, when all states are integer-valued as in Example 3).

5.2. Primal bounds

For a restricted diagram $\mathcal{D}_{\texttt{inner}}$, it is the case that $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{\texttt{inner}})) \subseteq \text{Conv}(\mathcal{Y})$ since $\mathcal{D}_{\texttt{inner}}$ contains a subset of the feasible solutions of $\mathcal{Y}$. Thus, restricted decision diagrams can be used to design network flow models that generate primal bounds on the ARBO problem.

Proposition 4 *Let $\mathcal{D}_{inner}$ denote a restricted decision diagram of $\mathcal{Y}$. Then, the model*

$$\min_{\mathbf{x},\mathbf{y},\boldsymbol{\lambda}} \quad \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \boldsymbol{\lambda} \tag{10a}$$

$$\text{s.t.} \quad \boldsymbol{T}^\top \boldsymbol{\lambda} = \mathbf{y} \tag{10b}$$

$$\mathbf{y} \in \text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{inner})) \tag{10c}$$

$$\mathbf{y} \in \mathcal{S}(\mathbf{x}) \tag{10d}$$

$$\mathbf{x} \in \mathcal{X} \tag{10e}$$

generates a feasible solution $\mathbf{x} \in \mathcal{X}$ and an upper bound on the optimal objective value of model (1).

There are conceptual connections between model (10) and K-adaptability in that both approximation methods are derived by restricting the set of possible recourse decisions. Nonetheless, there are two important differences. First, the approximation scheme defined by model (10) relies on continuous recourse variables, while K-adaptability relies on discrete recourse variables. Second, the size of model (10) can be more precisely controlled by incrementally changing the width of

the decision diagram. On the other hand, in K-adaptability, each incremental increase in the value of K requires adding a set of recourse decision variables $\mathbf{y}^{K+1}$ and corresponding constraints $\mathcal{Y}$. As we will illustrate in our numerical experiments (Section 6), problems with even a few K can quickly become intractable.

5.3. Dual bounds

A relaxed decision diagram $\mathcal{D}_{\texttt{outer}}$ of $\mathcal{Y}$ represents a superset of $\mathcal{Y}$, which implies that $\text{Conv}(\mathcal{Y}) \subseteq \text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{\texttt{outer}}))$. Relaxed decision diagrams can thus be used to derive a dual bound.

Proposition 5 *Let $\mathcal{D}_{outer}$ denote a relaxed decision diagram of $\mathcal{Y}$. Then, the model*

$$\min_{\mathbf{x},\mathbf{y},\boldsymbol{\lambda}} \quad \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \boldsymbol{\lambda} \tag{11a}$$
$$\text{s.t.} \quad \boldsymbol{T}^\top \boldsymbol{\lambda} = \mathbf{y} \tag{11b}$$
$$\mathbf{y} \in \text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{outer})) \tag{11c}$$
$$\mathbf{y} \in \text{Relax}(\mathcal{Y}) \tag{11d}$$
$$\mathbf{y} \in \mathcal{S}(\mathbf{x}) \tag{11e}$$
$$\mathbf{x} \in \mathcal{X} \tag{11f}$$

generates a feasible solution $\mathbf{x} \in \mathcal{X}$ and a lower bound on the optimal objective value of model (1).

Model (11) provides a dual bound that is at least as strong as that with a simple continuous relaxation of recourse decisions $\mathbf{y}$. However, this bound can be stronger, since in general, $\text{Relax}(\mathcal{Y}) \not\subseteq \text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{\texttt{outer}}))$ and $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{\texttt{outer}})) \not\subseteq \text{Relax}(\mathcal{Y})$. Specifically, $\text{Relax}(\mathcal{Y})$ can have fractional extreme points, whereas $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{\texttt{outer}}))$ is an integral polyhedron but includes solutions $\hat{\mathbf{y}} \in \{0,1\}^n$ that are not in $\mathcal{Y}$. The intersection of $\text{Relax}(\mathcal{Y})$ and $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}_{\texttt{outer}}))$ can thus be a more accurate outer approximation of $\text{Conv}(\mathcal{Y})$ than either of the two sets alone.

While both models (10) and (11) are bounding techniques that rely on approximating $\text{Conv}(\mathcal{Y})$ using a single decision diagram, we now propose a generalization where dual bounds can be derived with an outer approximation of $\text{Conv}(\mathcal{Y})$ using a *collection* of diagrams and feasible sets.

Proposition 6 *Suppose that $\mathcal{Y} = \{\mathbf{y} \in \{0,1\}^n \,|\, (\mathbf{g}^1)^\top \mathbf{y} \geq h_1, \ldots, (\mathbf{g}^J)^\top \mathbf{y} \geq h_J\}$, and let $\mathcal{J} = \{1, \ldots, J\}$ denote the set of constraint indices. Now, suppose we are given a collection of subsets of indices $\mathcal{J}_1^1, \ldots, \mathcal{J}_k^1$ and $\mathcal{J}_1^2, \ldots, \mathcal{J}_q^2$ where $\cup_{i=1}^k \mathcal{J}_i^1 \subseteq \mathcal{J}$ and $\cup_{i=1}^q \mathcal{J}_i^2 \subseteq \mathcal{J}$. For an arbitrary set $\mathcal{J}_i \subseteq \mathcal{J}$,*

let $\mathcal{D}^{\mathcal{J}_i}$ and $\mathcal{D}^{\mathcal{J}_i}_{outer}$ denote an exact and relaxed decision diagram for the feasible set described by the constraints with indices in $\mathcal{J}_i$, respectively. Then, the model

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{y},\boldsymbol{\lambda}} \quad & \mathbf{c}^\top \mathbf{x} + \mathbf{d}^\top \boldsymbol{\lambda} \\
\text{s.t.} \quad & \boldsymbol{T}^\top \boldsymbol{\lambda} = \mathbf{y} \\
& \mathbf{y} \in \text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}^{\mathcal{J}_i})), \qquad \forall \mathcal{J}_i \in \{\mathcal{J}^1_1, \dots, \mathcal{J}^1_k\} \\
& \mathbf{y} \in \text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}^{\mathcal{J}_i}_{outer})), \qquad \forall \mathcal{J}_i \in \{\mathcal{J}^2_1, \dots, \mathcal{J}^2_q\} \\
& \mathbf{y} \in \text{Relax}(\mathcal{Y}) \\
& \mathbf{y} \in \mathcal{S}(\mathbf{x}) \\
& \mathbf{x} \in \mathcal{X}
\end{aligned} \tag{12}$$

generates a feasible solution $\mathbf{x} \in \mathcal{X}$ and lower bound on the optimal objective value of model (1).

As a proof of concept, consider the model from Example 2, but with an additional constraint of $y_1 + y_2 + y_3 \leq 2$. Rather than formulating an exact or relaxed decision diagram of

$$\mathcal{Y} := \Big\{ \mathbf{y} \in \{0,1\}^5 \,|\, y_1 + y_2 + y_3 \leq 2, y_1 + y_2 + 2y_3 + 2y_4 + 3y_5 \leq 4 \Big\},$$

we could generate two exact or relaxed decision diagram, one for each constraint, and combine them into the multi-network flow model (12).

In practice, multi-network approximations can be useful when single-network representations are too large or when the underlying problem structure does not admit an obvious recursive formulation. In these settings, we can generate exact or relaxed decision diagrams for subsets of constraints that do admit a simple recursive formulation. Furthermore, in many decision-making problems, a large subset of constraints defining $\mathcal{Y}$ may have a totally unimodular constraint coefficient matrix and integer right-hand sides, and thus define a feasible space for which its relaxation is an integral polytope. In this setting, we can generate decision diagrams only for the remaining constraints. The corresponding multi-network flow approximations may potentially be smaller in size, easier to implement, or better in quality than single-network approximations. We will further explore these models in the numerical experiments in Section 6.2.

5.4. Evaluating the quality of a solution

All approximation techniques presented in this section generate a feasible first-stage solution $\mathbf{x} \in \mathcal{X}$. We can compute the true objective value of this solution, which we denote using $z(\hat{\mathbf{x}})$, by solving

$$z(\hat{\mathbf{x}}) := \max_{\boldsymbol{\xi} \in \Xi} \min_{\mathbf{y} \in \mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}})} \quad \mathbf{c}^\top \hat{\mathbf{x}} + \boldsymbol{\xi}^\top \mathbf{y}. \tag{13}$$

One way of solving for the value $z(\hat{\mathbf{x}})$ is to use the constraint generation method described in Kämmerling and Kurtz (2020). Specifically, the authors consider an iterative algorithm between

a master problem that computes a specific parameter realization $\hat{\boldsymbol{\xi}} \in \Xi$ and a subproblem which computes recourse solutions $\hat{\mathbf{y}} \in \mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}})$. At iteration k, the master problem is

$$\text{MP}(\hat{\mathbf{y}}^1, \ldots, \hat{\mathbf{y}}^{k-1}) \;=\; \max_{v, \boldsymbol{\xi}} \;\; \Big\{ v \;\Big|\; v \leq \mathbf{c}^\top \hat{\mathbf{x}} + \boldsymbol{\xi}^\top \hat{\mathbf{y}}^j \;\;\; \forall j \in \{1, \ldots, k-1\},\; \boldsymbol{\xi} \in \Xi \Big\}.$$

The optimal solution $\hat{\boldsymbol{\xi}}^k$ to this master problem is then passed to the subproblem

$$\text{SP}(\hat{\boldsymbol{\xi}}^k) \;=\; \min_{\mathbf{y}} \;\; \Big\{ \mathbf{c}^\top \hat{\mathbf{x}} + (\hat{\boldsymbol{\xi}}^k)^\top \mathbf{y} \;\Big|\; \mathbf{y} \in \mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}}) \Big\}.$$

Let $\hat{\mathbf{y}}^k$ denote an optimal solution of the subproblem. If the objective value of the subproblem is less than the master problem objective value, then we add constraint $v \leq \mathbf{c}^\top \hat{\mathbf{x}} + \boldsymbol{\xi}^\top \hat{\mathbf{y}}^k$ into the master problem, and move to iteration $k+1$. Otherwise, $z(\hat{\mathbf{x}}) = \mathbf{c}^\top \hat{\mathbf{x}} + (\hat{\boldsymbol{\xi}}^k)^\top \hat{\mathbf{y}}^k$ is the optimal objective value of Problem (13). Since the master problem is a linear program and the subproblem is one instantiation of the recourse problem, the effort required to compute the value of $z(\hat{\mathbf{x}})$ for a given $\hat{\mathbf{x}} \in \mathcal{X}$ is generally negligible relative to the general ARBO problem of solving for an optimal $\hat{\mathbf{x}} \in \mathcal{X}$.

Finally, recall that the approximation models presented in Section 5.3 generate dual bounds in addition to a solution $\hat{\mathbf{x}} \in \mathcal{X}$. For these models, we can thus compute a *model-based optimality gap*, which we define as

$$\frac{\text{dual bound} - z(\hat{\mathbf{x}})}{z(\hat{\mathbf{x}})} \cdot 100\,, \tag{14}$$

where 'dual bound' denotes the optimal objective value of the approximation model. The model-based optimality gap serves as a proxy for the true optimality gap. In particular, the former is an upper bound on the latter. This distinction is important, because the true optimality gap can be challenging to compute in numerical experiments as it requires solving the ARBO problem exactly. Nonetheless, we will show in Section 6 that for most problems considered, the model-based optimality gap is in fact very low. This observation highlights that these approximation models not only compute high-quality solutions but can also verify the near-optimality of the solutions.

6. Numerical Experiments

In this section, we demonstrate and compare the effectiveness of three types of formulations: (i) exact network flow models, (ii) approximate single-network flow models, and (iii) approximate multi-network flow models. These models are applied to two ARBO problems, namely, a capital budgeting problem and a robust assignment problem. All experiments were coded in Python 3.7 and MILP models were solved using Gurobi 9.1.1 under default settings. Decision diagrams were created and manipulated using the NetworkX package. The experiments were conducted on an Macbook Pro (M1 Chip) with 16GB of RAM. Unless otherwise stated, all benchmark models that we discuss next were also implemented and solved under the same conditions.

6.1. Capital budgeting problems

We first consider a capital budgeting problem, which seeks to compute a robust investment plan for a set of n projects under an investment budget constraint. Several variants of this problem have been studied in the adaptive robust optimization literature (Hanasusanto et al. 2015, Subramanyam et al. 2020, Kämmerling and Kurtz 2020, Arslan and Detienne 2022, Dumouchelle et al. 2023). In this problem, investment decisions are binary and can be made in two stages, i.e., projects can either be invested in at an early stage (denoted using $\mathbf{x} \in \{0,1\}^n$) or in a later stage (denoted using $\mathbf{y} \in \{0,1\}^n$). The profitability of a project is not known with certainty in the first stage and is only revealed in the second stage. Early stage investors are rewarded with a first-mover advantage that entails a higher percentage on final profit generated by the project (otherwise, the optimal action would be to postpone all investment decisions until more information is revealed).

For the rest of this section, we follow the model formulation from Arslan and Detienne (2022) and use their publicly-available problem instances[1]. Specifically, we consider the following adaptive capital budgeting problem

$$\max_{\mathbf{x}\in\{0,1\}^n} \min_{\boldsymbol{\xi}\in\Xi} \max_{\mathbf{y}} \quad (1-f)(\boldsymbol{\xi}^\top \mathbf{x}) + f(\boldsymbol{\xi}^\top \mathbf{y}) \tag{15a}$$

$$\text{s.t.} \quad \mathbf{y} \geq \mathbf{x} \tag{15b}$$

$$\mathbf{g}^\top \mathbf{y} \leq h \tag{15c}$$

$$\mathbf{y} \in \{0,1\}^n. \tag{15d}$$

In this model, g_i denotes the cost of investing in project i, h is the total investment budget, $f \in [0,1)$ captures the first-mover advantage, and ξ_i is the payoff of project i which is unknown in the first stage. The payoff of each project depends on a set of M common risk factors $\alpha_1, \ldots, \alpha_m$. Specifically, it is assumed that $\xi_i = \sum_{j=1}^{M} U_{ij}\alpha_j$ where $U_{ij} \in \mathbb{R}$ describes the impact that each risk factor α_j has on the payoff ξ_i. The risk factors α_j are assumed to reside in $[-1,1]$, i.e.,

$$\Xi = \left\{ \boldsymbol{\xi} \,\middle|\, \boldsymbol{\xi} = \boldsymbol{U}\boldsymbol{\alpha}, \boldsymbol{\alpha} \in [-1,1]^M \right\}.$$

Note that in this problem, constraint (15b) defines a selective adaptability constraint while constraints (15c)-(15d) can be reformulated or approximated in an extended network space.

We consider 300 instances of varying size where $n = \{10, 20, 30, 40, 50\}$ and where each value of n is associated with 60 different instances (Arslan and Detienne 2022). Each instance is characterized by a random sample of project costs and payoffs, as well as an investment budget defined as a fraction of total project costs, i.e., $h = m\sum_{i=1}^{n} g_i$ with $m = \{0.2, 0.4, 0.6, 0.8\}$.

[1] see https://github.com/borisdetienne/RobustDecomposition (accessed February 2024)

Since (15c) is a knapsack constraint, we generate the exact network flow model by following the steps in Example 1 and the reformulation technique outlined in Section 4. We also examine approximate network flow models based on relaxed decision diagrams. These diagrams are generated using the distance-based merging approach discussed in Section 5.1.2. Recall that in this approach, a user-specified value Q bounds the distance between state values of nodes that are to be merged. In the capital budgeting problem, the state value of a node in layer i defines the amount of weight that has been added to the knapsack based on decisions $y_1, \ldots, y_i$, and Q represents the maximum difference in weight values between any two nodes to be merged. We follow the procedure described in Example 4 to determine which nodes should be merged. Once merged, the new node is assigned the minimum state value of the nodes that were merged.

Since the approximation models generate first-stage solutions and dual bound, we can calculate optimality gaps for each solution. A key takeaway of the following numerical experiments is that these models can generate solutions that are verifiably near-optimal in very little time.

6.1.1. Model formulation and solution time. We begin by examining the size, formulation time, and solution time of the exact and approximate network flow models. Table 1 highlights the number of arcs in the decision diagrams used to represent constraint (15c), shown over various values of Q. This number conveys the size of the corresponding network flow model, since each arc in the decision diagram corresponds to a continuous variable in our formulation.

	$Q=0$	$Q=1$	$Q=3$	$Q=5$	$Q=10$
$n=10$	166	164 (99%)	151 (91%)	134 (81%)	109 (65%)
$n=20$	2992	1972 (66%)	1198 (40%)	894 (30%)	554 (19%)
$n=30$	12334	6726 (55%)	3600 (29%)	2463 (20%)	1359 (11%)
$n=40$	28121	14621 (52%)	7374 (26%)	4912 (17%)	2622 (9%)
$n=50$	48285	24589 (51%)	12180 (25%)	7954 (16%)	4160 (9%)

Table 1 The average number of arcs in the reduced decision diagrams under different Q values. The percentage of each value relative to that of the exact decision diagram (i.e., $Q=0$) are given in parentheses.

Similarly, Table 2 highlights the time it takes to generate the decision diagram and solve the network flow model. We make two observations. When n increases, the time it takes to formulate and solve the exact model grows exponentially. This makes the exact models intractable beyond small values n. In contrast, as we increase Q for a fixed value of n, the time it takes for both these processes can be reduced drastically. For example, when $n=50$, the total time it takes to generate and solve the network flow models can be reduced by several orders of magnitude when $Q \geq 3$.

Next, we examine the quality of the solution generated by these approximate models.

	$Q=0$	$Q=1$	$Q=3$	$Q=5$	$Q=10$
Instances	*Average build time for the reduced diagram, in seconds*				
$n=10$	0.1s	0.1s	0.1s	0.1s	0.1s
$n=20$	3s	1s	0.3s	0.2s	0.1s
$n=30$	17s	5s	2s	0.7s	0.2s
$n=40$	55s	16s	4s	2s	0.5s
$n=50$	112s	33s	8s	4s	1s
Instances	*Average solution time of network flow model, in seconds*				
$n=10$	0.1s	0.1s	0.1s	0.1s	0.1s
$n=20$	1s	0.5s	0.3s	0.2s	0.1s
$n=30$	87s	11s	3s	1s	0.3s
$n=40$	534s	45s	8s	3s	1s
$n=50$	>3600s	117s	22s	8s	2s

Table 2 **The average build time and solution time of relaxed decision diagrams under different Q values. Build time includes time to generate and reduce a diagram.**

6.1.2. Quality of model solutions. For each solution, we calculate the model-based optimality gap as well as the true optimality gap where possible (see Section 5.4 for details). Table 3 summarizes the values of these optimality gaps. Note that only the model-based optimality gap is shown for the $n=50$ instances, since the exact models (i.e., where $Q=0$) could not be solved within the time limit. We also note that since the formulation and solution times of the exact models are generally negligible for $n=10$ and $n=20$ (see Table 2), we focus our discussion on the instances with $n=30, 40$ and 50, where tractable approximations become more critical.

Instances	$Q=0$	$Q=1$	$Q=3$	$Q=5$	$Q=10$
$n=10$	0%	1.2% (1.5%)	3.0% (4.3%)	2.6% (4.6%)	4.4% (7.6%)
$n=20$	0%	1.2% (1.5%)	1.5% (2.1%)	1.0% (1.7%)	1.1% (2.2%)
$n=30$	0%	0.3% (0.5%)	0.4% (0.7%)	0.5% (0.8%)	0.9% (1.4%)
$n=40$	0%	0.1% (0.2%)	0.2% (0.3%)	0.2% (0.4%)	0.2% (0.5%)
$n=50$	-	- (0.08%)	- (0.2%)	- (0.3%)	- (0.5%)

Table 3 **The average true optimality gap and model-based optimality gap of the solution, the latter of which is shown in parentheses.**

The main takeaway from Table 3 is that both the true and the model-based optimality gaps remain very small despite the large reduction in solution times (as shown in Table 2). For example, when $n=40$ and $Q=5$, the average optimality gap of solutions is less than 0.5%, despite the models taking two orders of magnitude less time to formulate and solve compared to the exact model (i.e., an average of 5 seconds versus 589 seconds; see Table 2). Similarly, when $n=50$, the

average model-based optimality gap is 0.08% when $Q = 1$ and 0.5% when $Q = 10$, the latter of which requires an average solution time that is three orders of magnitude less than the exact model (i.e., 3 seconds versus 3600+ seconds; see Table 2). These observations highlight that the approximate network flow models are able to independently generate both (i) near-optimal solutions *and* (ii) almost-tight dual bounds, as both of these conditions must be met to observe small model-based optimality gaps. Finally, as a side note, we point out that for any fixed value of Q, the average optimality gap decreases as n increases. This is because in our problem setting, by holding Q constant, the relative degree of the approximation decreases as the problem size increases.

Figure 2 provides a more nuanced illustration of the model-based optimality gap for each instance as a function of solution time. The main observation is that across all problem sizes, we observe an exponential decay in the optimality gap as a function of solution time. Specifically, the results show that when solution times are small, a slight increase in model complexity and solution time (i.e., by decreasing the value of Q slightly) can result in a large decrease in optimality gap. Put differently, a slight approximation of the exact model can drastically reduce solution times while sacrificing very little in terms of the quality of solutions and dual bounds. This is most evident in the $n = 40$ ($n = 50$) instances, where reducing the average solution time from 589 seconds (3600+ seconds) to 1.5 seconds (3 seconds) results in an average suboptimality loss of 0.2% (at most 0.5%). These insights highlight that our approximate network flow models are highly effective and scalable.

6.1.3. Brief comparison with alternative methods. We briefly comment on the performance of the K-adaptability model (Hanasusanto et al. 2015), which is a popular approximation method that has also served as a benchmark for the same capital budgeting instances (i.e., see Arslan and Detienne (2022) and Dumouchelle et al. (2023)). The K-adaptability model generates a feasible solution and a primal bound; for reference, we provide the complete formulation in Section EC.2.2. Since there is strong evidence from previous literature that the model can be challenging to solve even for small problem sizes and small K, we, for practical reasons, select a sub-sample of 100 instances and impose a maximum time limit of 1200 seconds for each instance. Our sub-sample consists of 20 instances for each value of $n = \{10, 20, 30, 40, 50\}$, which we select simply by choosing every third instance in the GitHub repository from Arslan and Detienne (2022).

Table 4 highlights the solution times of the K-adaptability model for $K = 2, 3, 4$. First, note that the solution times for the K-adaptability model increase exponentially as we increase either the value of K or the size of the instances. For example, even when $K = 2$, many instances where $n \geq 30$ cannot be solved within the 1200-second time limit. As another example, when $n = 20$, the average solution time goes from 13 seconds to over 675 seconds when K is increased from 2 to 4. When $n = 40$ and $n = 50$, 15 out of 20 instances could not be solved within 1200 seconds for any K,

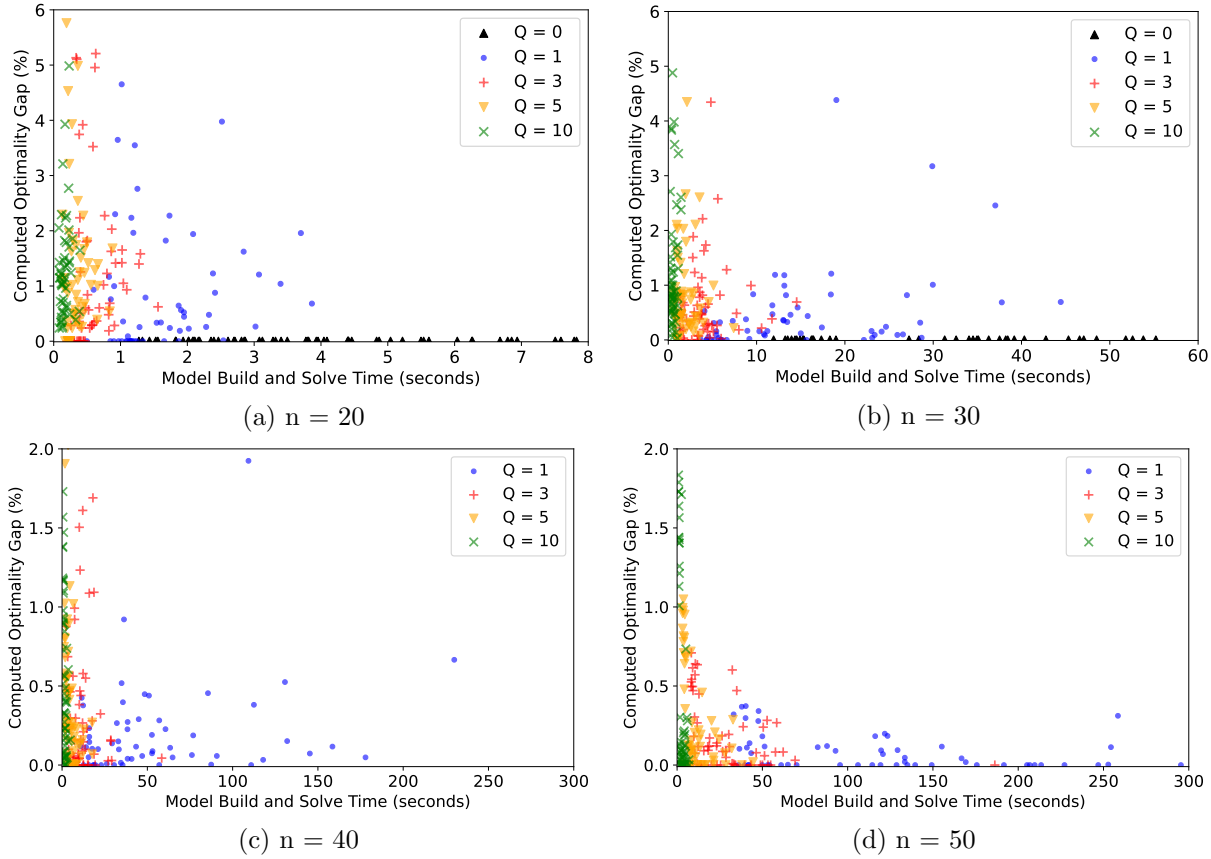

Figure 2 **The model-based optimality gap as a function of the total time to formulate and solve the network flow models. Note that the data points roughly follow an exponential decay curve, that is, an incremental increase in solution time can result in a large decrease in optimality gap, when solution times are small.**

while 5 of 20 took less than one second (resulting in the consistent average of 900 seconds). Finally, many instances of K-adaptability had large optimality gaps (e.g., $> 10\%$) when the 1200-second time limit was reached.

Instances	$Q=0$	$Q=5$	$Q=10$	$K=2$	$K=3$	$K=4$
n = 10	0.1s	0.1s	0.1s	0.1s	0.2s	1.2s
n = 20	4s	0.3s	0.1s	13s	> 171s	> 675s
n = 30	176s	2s	0.5s	> 627s	> 755s	> 900s
n = 40	-	4s	2s	> 900s	> 900s	> 900s
n = 50	-	10s	3s	> 900s	> 900s	> 900s

Table 4 **A comparison of average solution times of approximation models. For each entry, the symbol $>$ is used to denote any average that is taken when there exists at least 1 instance that exceeds the 1200-second threshold. The total time for the network flow models include the model formulation time (i.e., diagram generation time).**

In comparison to the K-adaptability models, our network flow models can be solved much more efficiently. For example, the average solution time for instances where $n = 50$ and $Q = 5$ is 10 seconds, and, as we highlighted in the previous subsection, the generated first-stage solutions are within 0.2% of optimality. As another example, it takes an average of 4 seconds to solve the $n = 20$ instances exactly, while it takes more than 675 seconds to solve the K-adaptability model with $K = 4$ (which still does not generate optimal solutions for all instances). Finally, and perhaps most importantly, our network flow models also simultaneously generate high-quality dual bounds, which allow us to evaluate the quality of *any* feasible solution, including those that are generated independently by heuristics or other approximation models like K-adaptability.

In summary, our models provide a flexible framework for generating both high-quality solutions and dual bounds in significantly less time. Compared to K-adaptability, the complexity of our model can also be tuned much more precisely. For example, increasing the value of Q gradually increases the solution time of the model, whereas solution times increase exponentially with small changes in the value of K. Related discussion was also presented in Sections 5.1.2 and 5.2.

Lastly, we remark that while our discussion focuses on the K-adaptability model for reasons previously mentioned (e.g., popularity and ease of implementation in standard solvers), we also compare our results to the solutions times of the exact branch-and-price algorithm presented in Arslan and Detienne (2022). We find that the solution times of many exact and/or near-exact reformulations do not exceed those reported for the branch-and-price algorithm, but more importantly, our approach can generate approximate reformulations that are significantly faster to solve while sacrificing little in terms of solution quality. We refer the reader to Section EC.2.3 of the Electronic Companion for details.

6.2. Robust assignment problems

In the previous subsection, we examined exact and approximate single-network flow models in the context of the capital budgeting problem, which has a single linking constraint between first-stage decisions and each second-stage decision. In this subsection, we examine a problem setting in which numerous such constraints exist. Specifically, we examine robust assignment problems and focus on the use of multi-network flow models, which are discussed in Section 5.3.

Assignment problems encompass numerous decision-making tasks that span many applications. A standard assignment problem can be modeled as a bipartite graph $(\mathcal{L}, \mathcal{M}, \mathcal{S})$ of agents $\mathcal{L} = \{1, \ldots, L\}$, tasks $\mathcal{M} = \{1, \ldots, M\}$ and directed links $\mathcal{S}$. Let $\mathcal{L}(m) \subseteq \mathcal{L}$ denote the subset of agents for which there exists a directed link to task $m \in \mathcal{M}$, and similarly, let $\mathcal{M}(\ell) \subseteq \mathcal{M}$ denote the subset of tasks which can be assigned to agent $\ell \in \mathcal{L}$. Depending on the context, agents can represent people, products, resources, funding, or jobs, whereas tasks can represent locations, facilities, projects or

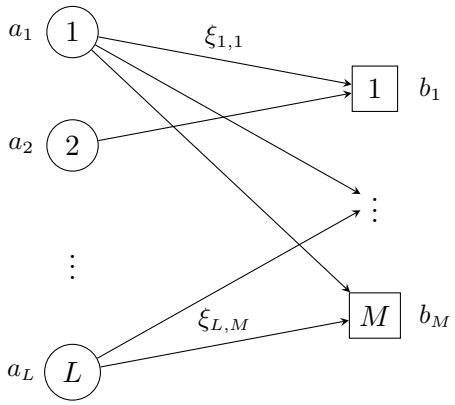

Figure 3 **A robust assignment problem. Agents $1,\ldots,L$ are associated with weight $a_1,\ldots,a_L$, tasks $1,\ldots,M$ are associated with capacity $b_1,\ldots,b_M$, and agent-task pairings (ℓ,m) generate reward $\xi_{\ell m}$.**

machines. Each agent $\ell \in \mathcal{L}$ is associated with weight $a_\ell \geq 0$ and each task m has capacity $b_m \geq 0$, while the reward $\xi_{\ell,m} \geq 0$ (e.g., match quality) of a specific agent-task pairing is unknown (e.g., varies day-to-day or must be estimated from data). Figure 3 illustrates this assignment problem.

Given this setup, a standard robust assignment problem can be formulated as

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} \boldsymbol{\xi}^\top \mathbf{x} \qquad \text{with} \qquad \mathcal{X} = \Big\{ \mathbf{x}\in\{0,1\}^{|\mathcal{S}|} \;:\; \sum_{\ell\in\mathcal{L}(m)} a_\ell x_{\ell,m} \leq b_m, \quad \forall m\in\mathcal{M}, \\ \sum_{m\in\mathcal{M}(\ell)} x_{\ell,m} \leq 1, \qquad \forall \ell\in\mathcal{L} \Big\}.$$

Consider a setting where a planner seeks to introduce a limited degree of flexibility in the assignment planning and decision-making process. Specifically, suppose that the planner wants to pre-identify a set of potential agent-task pairings from which assignments can be made once rewards become known. Practically, this may arise when there is a desire to train people for specific tasks, inform individuals about potential assignments, or inspect match quality before making a final decision. We assume that we have a cardinality constraint limiting the number of pre-identified pairings, and formulate the adaptive robust assignment problem as

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} \max_{\mathbf{y}} \quad \boldsymbol{\xi}^\top \mathbf{y} \tag{16a}$$

$$\text{s.t.} \quad \sum_{\ell\in\mathcal{L}(m)} a_\ell y_{\ell,m} \leq b_m, \quad \forall m\in\mathcal{M} \tag{16b}$$

$$\sum_{m\in\mathcal{M}(\ell)} y_{\ell,m} \leq 1, \qquad \forall \ell\in\mathcal{L} \tag{16c}$$

$$\mathbf{y}\in\{0,1\}^{|\mathcal{S}|} \tag{16d}$$

$$\mathbf{y} \leq \mathbf{x} \tag{16e}$$

where $\mathcal{X} = \left\{\mathbf{x} \in \{0,1\}^{|\mathcal{S}|} \ : \ \|\mathbf{x}\|_1 \leq \beta \cdot |\mathcal{S}|\right\}$. In our experiments, we consider β values of 0.5, 0.6, 0.7, 0.8, and 0.9, which correspond to the ability to pre-identify 50%, 60%, 70%, 80%, and 90% of pairings in the first stage, which can then be used to form a final assignment in the second stage.

6.2.1. Experimental setup. We consider assignment problems of five sizes where (L, M) is equal to (20,2), (20,3), (20,4), (25,5) and (25,8). For each problem size, we randomly generate 10 instances. Each instance is characterized by (i) a randomly generated a bipartite graph with 50% sparsity (such that $|\mathcal{S}| = 20, 30, 40, 63$ and 100, respectively) and (ii) randomly generated vectors of agent weights $\mathbf{a}$ and task capacities $\mathbf{b}$, where each element of $\mathbf{a}$ is a random integer from $[1, 10]$ and each element of $\mathbf{b}$ is an random integer from $[2 \cdot \max(\mathbf{a}), \frac{1}{m}||\mathbf{a}||_1]$.

For each instance, we generate the uncertainty set as follows. Let $\boldsymbol{\xi}^0$ denote a vector representing the nominal reward of each link. With slight abuse of notation, each element ξ_i^0 is a random number generated from $[0.5a, a]$ where a is the weight of the agent associated with this link. Then, the uncertainty set is defined on the percentage of deviation from $\boldsymbol{\xi}^0$, namely,

$$\Xi = \left\{ \boldsymbol{\xi} \ \middle| \ \sum_{i=1}^{|\mathcal{S}|} \left|\xi_i/\xi_i^0 - 1\right| \leq 0.1|\mathcal{S}|, \ \left|\xi_i/\xi_i^0 - 1\right| \leq 0.5 \ \ \forall i \in \{1, \ldots, |\mathcal{S}|\} \right\}.$$

We consider both an exact network flow formulation and a multi-network flow approximation for solving the adaptive assignment problems. These two formulations are denoted as Exact NF and Multi NF in the tables and figures, and we generate them as follows:

- **Exact NF.** Model (16) can be represented as an exact network flow model by reformulating constraints (16b) – (16d) based on a straightforward extension of the procedure outlined in Example 1. Specifically, the state $\boldsymbol{S}$ in the recursive formulation of (16b) – (16d) is a vector of size $L + M$ (rather than scalars) that captures the amount of capacity remaining in (16b) – (16c) based on previous decisions of $y_1, \ldots, y_i$. Generating this recursive formulation (i.e., the decision diagram) is straightforward, as shown in Section EC.3.1 of the Electronic Companion.
- **Multi NF.** To form the multi-network flow approximation, we generate an exact decision diagram for each knapsack constraint in (16b) along with binary domains in (16d) and integrate the corresponding network flow constraints. We leave constraints (16c) intact since they satisfy the integral polyhedron property (as an implication of Proposition 6 in Section 5.3). This model generates a feasible first-stage solution and a dual bound.

The Multi NF model defined above can be considered as one of the most natural or straightforward approximations of the ARBO problem, and we will show that it performs well in the numerical experiments. Nonetheless, we remark that there are many possible variations of this model which we could use to tighten or loosen the approximation. For example, instead of generating one exact decision diagram for each knapsack constraint, we could generate one for each pair of constraints,

which would tighten the approximation quality but potentially lead to larger formulations. On the other hand, we could also generate an approximate decision diagram for each knapsack constraint (like in Section 6.1), which leads to a worse approximation but could significantly reduce the size and solution time of the model. In summary, various Multi NF models could be defined that will trade off between lower solution times and better solution quality.

Finally, for each network flow model, we enforce a one-hour time limit to build and reduce each diagram. Similar to Section 6.1, we also consider the K-adaptability model with $K=3$ and $K=4$ as a point of reference; the complete MILP formulation can be found in Section EC.3.3 of the Electronic Companion. For all models, we enforce a 30-minute time limit for the solver.

Model	BDD attributes	(20,2)	(20,3)	(20,4)	(25,5)	(25, 8)
Exact NF	avg. build time	0.5s	67s	-	-	-
	avg. reduction time	1.4s	243s	-	-	-
	avg. # of arcs (unreduced)	11994	199856	-	-	-
	avg. # of arcs (reduced)	498	5506	-	-	-
Multi NF	avg. build time	0.1s	0.1s	0.1s	0.1s	0.1s
	avg. reduction time	0.1s	0.1s	0.1s	0.2s	0.4s
	avg. # of arcs (unreduced)	1262	2577	3911	8855	20365
	avg. # of arcs (reduced)	287	884	1385	3669	7742

Table 5 The build time and attributes of the decision diagrams. Values for Multi NF are given as a summation over all the diagrams used to generate the multi-network formulation for a particular instance.

6.2.2. Computational results. We first give an overview of the size and formulation time of the network flow models. Then, we examine the solution time, solution quality, and dual bounds generated by the models.

The attributes of the decision diagrams underlying the network flow models are shown in Table 5. Specifically, Table 5 highlights the time it takes to generate and reduce the diagrams as well as the sizes of the diagrams. We note that the size and formulation time of the exact decision diagrams increases rapidly with the size of the assignment problem. Specifically, it takes an average of 2 seconds for the (20,2) instances, 310 seconds for the (20,3) instances, and more than one hour for all the other instances. Since the set of all feasible recourse decisions must be represented within a single decision diagram, the size of this diagram may grow exponentially with the number of constraints used to define the feasible space. In fact, in the robust assignment problem, the main computational bottleneck problem is in generating the exact decision diagram rather than solving the corresponding network flow formulation. This will become even more clear in the next paragraph. In contrast, the average size and formulation time of the multi-network models is far

smaller and generally negligible (<1 second). Since this model is simply a collection of individual decision diagrams that are each exact reformulations of only one constraint in (16b), the size and formulation time scales linearly in the number of constraints in (16b). Note that for Multi NF models, the generation of individual diagrams for each constraint could be parallelized.

Table 6 highlights the average solution time of the various models over different problem instances. Note that despite the exact decision diagram taking significant time to generate for the (20,3) instances, it only takes 1 second to solve the exact network flow formulation. As for the Multi NF models, they can be solved relatively efficiently even as the size of the assignment problem grows larger. We briefly point out that the solution times for the K-adaptability model with $K=3$ and $K=4$ are orders of magnitude larger than the network flow models.

Model	(20,2)		(20,3)		(20,4)		(25,5)		(25,8)	
Exact NF	0.1s	(50)	1.1s	(50)	-		-		-	
Multi NF	0.1s	(50)	0.3s	(50)	1.8s	(50)	81s	(50)	>633s	(33)
3-Adapt	1.2s	(50)	48s	(50)	>324s	(46)	>1634s	(5)	>1800s	(0)
4-Adapt	26s	(50)	>342s	(46)	>1134s	(24)	>1749s	(3)	>1800s	(0)

Table 6 Average solution times across the different models. The value in the parentheses denotes the number of instances, out of 50, that could be solved within the 30-minute time limit given to each instance.

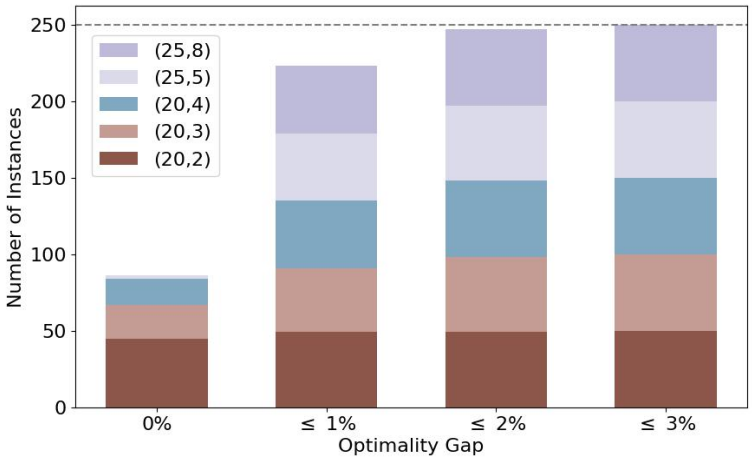

Figure 4 Model-based optimality gap of solutions of the Multi NF model.

Despite having significantly lower solution times, the Multi NF models are still able to generate solutions that have very low model-based optimality gaps, as shown in Figure 4. Note that, as discussed in Section 6.1, these values are upper bounds on the true optimality gap of the solution, meaning that the true quality of the solution could be even better than expected. First, we note that

86 out of the 250 problem instances had a model-based optimality gap of 0%. In these instances, the Multi NF model found the optimal solution *and* verified its optimality. Most of these cases pertained to problem instances that were smaller in size. Second, 223 out of 250 instances had a model-based optimality gap that was less than 1%, 247 out of 250 had a gap less than 2%, and no solution had an optimality gap that was greater than 3%. These statistics include solutions of the Multi NF instances that could not be solved within the time limit (see Table 6).

The main takeaway of these results is that the multi-network flow model can efficiently generate near-optimal first-stage solutions *and* high-quality dual bounds across all problem instances considered. This is consistent with the observations from Section 6.1, which highlighted the same findings using approximate single-network flow models.

6.3. Summary of numerical results

We briefly summarize the main takeaways of the numerical section. First, we show that exact network flow formulations can be tractably formulated and solved for smaller instances of ARBO problems. Second, for larger instances, the approximate network flow formulations are tractable and simultaneously generate (i) high-quality solutions and (ii) high-quality dual bounds. Finally, the size and solution times of the approximate network flow models can often be reduced drastically while sacrificing very little in terms of the quality of solutions and dual bounds.

7. Conclusion

In this paper, we examine adaptive robust binary optimization problems with objective uncertainty. We leverage ideas from the decision diagram community to reformulate and approximate our adaptive problems as single-stage constrained network flow models. We outline methods to generate network models where the size and quality of the models can be easily controlled through a user-specified parameter. Our models are also easy to implement and solve using standard MILP solvers. Through an extensive set of computational experiments, we show that these models can efficiently generate both high-quality solutions and high-quality dual bounds.

By forming this connection between adaptive robust optimization and decision diagrams, our framework can also take advantage of independent research contributions that emerge from the latter research community. Specifically, these developments may create opportunities to extend our ideas to more general problem settings. For example, recent literature has considered the use of decision diagrams to solve deterministic integer and/or nonlinear optimization problems (Castro et al. 2022), and similar ideas could potentially be integrated into our framework to reformulate adaptive robust problems with general integer recourse.

References

Eduardo Álvarez-Miranda, Elena Fernández, and Ivana Ljubić. The recoverable robust facility location problem. *Transportation Research Part B: Methodological*, 79:93–120, 2015.

Ayşe N Arslan and Boris Detienne. Decomposition-based approaches for a class of two-stage robust binary optimization problems. *INFORMS Journal on Computing*, 34(2):857–871, 2022.

Vedat Bayram, Gohram Baloch, Fatma Gzara, and Samir Elhedhli. Optimal order batching in warehouse management: A data-driven robust approach. *INFORMS Journal on Optimization*, 2022.

David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016.

David Bergman, Merve Bodur, Carlos Cardonha, and Andre A Cire. Network models for multiobjective discrete optimization. *INFORMS Journal on Computing*, 34(2):990–1005, 2022.

Dimitris Bertsimas and Iain Dunning. Multistage robust mixed-integer optimization with adaptive partitions. *Operations Research*, 64(4):980–998, 2016.

Dimitris Bertsimas and Angelos Georghiou. Design of near optimal decision rules in multistage adaptive mixed-integer optimization. *Operations Research*, 63(3):610–627, 2015.

Dimitris Bertsimas, Ebrahim Nasrabadi, and Sebastian Stiller. Robust and adaptive network flows. *Operations Research*, 61(5):1218–1242, 2013.

Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

Christoph Buchheim and Jannis Kurtz. Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238, 2018.

Margarita P Castro, Andre A Cire, and J Christopher Beck. Decision diagrams for discrete optimization: A survey of recent advances. *INFORMS Journal on Computing*, 34(4):2271–2295, 2022.

Andre A Cire, Adam Diamant, Tallys Yunes, and Alejandro Carrasco. A network-based formulation for scheduling clinical rotations. *Production and Operations Management*, 28(5):1186–1205, 2019.

Gülesin Sena Daş, Fatma Gzara, and Thomas Stützle. A review on airport gate assignment problems: Single versus multi objective approaches. *Omega*, 92:102146, 2020.

Vinícius L de Lima, Cláudio Alves, François Clautiaux, Manuel Iori, and José M Valério de Carvalho. Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *European Journal of Operational Research*, 296(1):3–21, 2022.

Justin Dumouchelle, Esther Julien, Jannis Kurtz, and Elias B Khalil. Neur2ro: Neural two-stage robust optimization. *arXiv preprint arXiv:2310.04345*, 2023.

Lars Eufinger, Jannis Kurtz, Christoph Buchheim, and Uwe Clausen. A robust approach to the capacitated vehicle routing problem with uncertain costs. *INFORMS Journal on Optimization*, 2(2):79–95, 2020.

Bram L Gorissen, İhsan Yanıkoğlu, and Dick den Hertog. A practical guide to robust optimization. *Omega*, 53:124–137, 2015.

Cheng Guo, Merve Bodur, Dionne M Aleman, and David R Urbach. Logic-based Benders decomposition and binary decision diagram based approaches for stochastic distributed operating room scheduling. *INFORMS Journal on Computing*, 33(4):1551–1569, 2021.

Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

John N Hooker. Decision diagrams and dynamic programming. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 94–110. Springer, 2013.

Nicolas Kämmerling and Jannis Kurtz. Oracle-based algorithms for binary two-stage robust optimization. *Computational Optimization and Applications*, 77(2):539–569, 2020.

Adam Kasperski and Paweł Zieliński. Robust recoverable and two-stage selection problems. *Discrete Applied Mathematics*, 233:52–64, 2017.

Leonardo Lozano and J Cole Smith. A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. *Mathematical Programming*, pages 1–24, 2018.

Leonardo Lozano, David Bergman, and Andre A Cire. Constrained shortest-path reformulations for discrete bilevel and robust optimization. *arXiv preprint arXiv:2206.12962*, 2022.

Moira MacNeil and Merve Bodur. Leveraging decision diagrams to solve two-stage stochastic programs with binary recourse and logical linking constraints. *European Journal of Operational Research*, 2023.

Duncan C McElfresh, Hoda Bidkhori, and John P Dickerson. Scalable robust kidney exchange. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1077–1084, 2019.

J v Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

Krzysztof Postek and Dick den Hertog. Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. *INFORMS Journal on Computing*, 28(3):553–574, 2016.

Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

Thiago Serra, Arvind U Raghunathan, David Bergman, John Hooker, and Shingo Kobori. Last-mile scheduling under uncertainty. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 519–528. Springer, 2019.

Anirudh Subramanyam, Chrysanthos E Gounaris, and Wolfram Wiesemann. K-adaptability in two-stage mixed-integer robust optimization. *Mathematical Programming Computation*, 12(2):193–224, 2020.

Willem-Jan van Hoeve. Graph coloring with decision diagrams. *Mathematical Programming*, 192(1):631–674, 2022.

Phebe Vayanos, Daniel Kuhn, and Berç Rustem. Decision rules for information discovery in multi-stage stochastic programming. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 7368–7373. IEEE, 2011.

Chiwei Yan and Jerry Kung. Robust aircraft routing. *Transportation Science*, 52(1):118–133, 2018.

İhsan Yanıkoğlu, Bram L Gorissen, and Dick den Hertog. A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3):799–813, 2019.

Bo Zeng and Long Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

Electronic Companion

EC.1. Proofs

Proof of Lemma 1. We first show that $\text{Conv}(\mathcal{Y} \cap \mathcal{S}(\mathbf{x})) \subseteq \text{Conv}(\mathcal{Y}) \cap \mathcal{S}(\mathbf{x}),\ \forall \mathbf{x} \in \{0,1\}^m$. For any $\hat{\mathbf{x}} \in \{0,1\}^m$, let $\hat{\mathbf{y}}$ be an arbitrary point in $\text{Conv}(\mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}}))$. By definition, $\hat{\mathbf{y}}$ can be written as a convex combination of points $\mathbf{y}^1, \ldots, \mathbf{y}^R$ that are feasible in both $\mathcal{Y}$ and $\mathcal{S}(\hat{\mathbf{x}})$. Since $\mathbf{y}^1, \ldots, \mathbf{y}^R \in \mathcal{Y}$, it must be true that $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{Y})$. Similarly, since $\mathbf{y}^1, \ldots, \mathbf{y}^R \in \mathcal{S}(\hat{\mathbf{x}})$, $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{S}(\hat{\mathbf{x}})) = \mathcal{S}(\hat{\mathbf{x}})$.

Next, we show that $\text{Conv}(\mathcal{Y}) \cap \mathcal{S}(\hat{\mathbf{x}}) \subseteq \text{Conv}(\mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}}))$. We prove this by contradiction. Suppose there exists a vector $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{Y}) \cap \mathcal{S}(\hat{\mathbf{x}})$ such that $\hat{\mathbf{y}} \notin \text{Conv}(\mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}}))$. Then, $\hat{\mathbf{y}}$ must satisfy one of the two conditions:

- The vector $\hat{\mathbf{y}}$ is a binary vector, i.e., $\hat{\mathbf{y}} \in \{0,1\}^n$. However, if $\hat{\mathbf{y}} \in \{0,1\}^n$ and $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{Y})$, then $\hat{\mathbf{y}} \in \mathcal{Y}$. This implies that $\hat{\mathbf{y}} \in \mathcal{Y}$ and $\hat{\mathbf{y}} \in \mathcal{S}(\hat{\mathbf{x}})$, which further implies that $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}}))$. This is a contradiction.
- The vector $\hat{\mathbf{y}}$ must have at least one index k where $\hat{y}_k \in (0,1)$. This implies that $\hat{\mathbf{y}}$ must be a strict convex combination of a set of binary vectors $\mathbf{y}^1, \ldots, \mathbf{y}^r \in \mathcal{Y}$, that is, $\hat{\mathbf{y}} = \sum_{r=1}^{R} \lambda_r \mathbf{y}^r$, $\sum_{r=1}^{R} \lambda_r = 1$, and $\lambda_r > 0\ \forall r \in \{1, \ldots, R\}$. If all binary vectors $\mathbf{y}^1, \ldots, \mathbf{y}^r \in \mathcal{S}(\hat{\mathbf{x}})$, then $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{S}(\hat{\mathbf{x}})) = \mathcal{S}(\hat{\mathbf{x}})$ which implies that $\hat{\mathbf{y}} \in \text{Conv}(\mathcal{Y} \cap \mathcal{S}(\hat{\mathbf{x}}))$. This is a contradiction. Thus, it must be the case that there exists a binary vector $\mathbf{y}^\ell \in \{\mathbf{y}^1, \ldots, \mathbf{y}^r\}$ such that $\mathbf{y}^\ell \notin \mathcal{S}(\hat{\mathbf{x}})$. Based on our definition of $\mathcal{S}(\hat{\mathbf{x}})$, this implies that there is a violated constraint of the form $y_i^\ell \leq \hat{x}_j$, $y_i^\ell = \hat{x}_j$ or $y_i^\ell \geq \hat{x}_j$ for some $j \in \{1, \ldots, m\}$. Recall that $\mathbf{y}^\ell$ and $\hat{\mathbf{x}}$ are binary vectors. Without loss of generality, suppose $y_i^\ell = 1$ which violates the constraint $y_i^\ell \leq \hat{x}_j$ when $\hat{x}_j = 0$. However, since $\hat{\mathbf{y}}$ is a strict combination of binary vectors which includes $\mathbf{y}^\ell$, we have $\hat{y}_i > 0$, thus it must be true that $\hat{y}_i$ also violates this constraint. This implies that $\hat{\mathbf{y}} \notin \mathcal{S}(\hat{\mathbf{x}})$, which contradicts our initial assumption. This argument can be made for every value of $\hat{y}_i^\ell$ and $\hat{x}_j$ which violates one of the selectively adaptive constraints. This concludes our proof. □

Proof of Proposition 1. The main idea behind this proof comes from Proposition 1 of Arslan and Detienne (2022). We first note that Problem (1) is equivalent to

$$\min_{\mathbf{x} \in \mathcal{X}} \max_{\boldsymbol{\xi} \in \Xi} \min_{\mathbf{y} \in \text{Conv}(\mathcal{Y} \cap \mathcal{S}(\mathbf{x}))} \mathbf{c}^\top \mathbf{x} + \boldsymbol{\xi}^\top \mathbf{y}.$$

Since both the maximization problem and the inner minimization problem are now over convex sets, we can apply the minimax theorem (Neumann 1928) to swap the order of these two operations and derive the equivalent reformulation of

$$\min_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{y} \in \text{Conv}(\mathcal{Y} \cap \mathcal{S}(\mathbf{x}))} \max_{\boldsymbol{\xi} \in \Xi} \mathbf{c}^\top \mathbf{x} + \boldsymbol{\xi}^\top \mathbf{y}.$$

Using Lemma 1, we have $\text{Conv}(\mathcal{Y} \cap \mathcal{S}(\mathbf{x})) = \text{Conv}(\mathcal{Y}) \cap \mathcal{S}(\mathbf{x})$, thus we can combine the two minimization operations into a single one that is solved with $\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \text{Conv}(\mathcal{Y})$, and $\mathbf{y} \in \mathcal{S}(\mathbf{x})$. Finally, we can introduce the auxiliary variable v to switch to the epigraph formulation of the maximization problem, which concludes the proof. □

Proof of Proposition 6 This proposition comes directly from the result that for any two sets $\mathcal{A}, \mathcal{B} \subseteq \mathbb{R}^n$, it must be true that $\text{Conv}(\mathcal{A} \cap \mathcal{B}) \subseteq \text{Conv}(\mathcal{A}) \cap \text{Conv}(\mathcal{B})$. As a quick proof, let $\mathbf{x}$ be an arbitrary point in $\text{Conv}(\mathcal{A} \cap \mathcal{B})$. By definition, $\mathbf{x}$ can be written as a convex combination of feasible points in $\mathcal{A} \cap \mathcal{B}$. Since these feasible points are in both $\mathcal{A}$ and $\mathcal{B}$, they must also be in $\text{Conv}(\mathcal{A})$ and $\text{Conv}(\mathcal{B})$. Finally, since $\mathbf{x}$ is a convex combination of these points, $\mathbf{x}$ must also be in $\text{Conv}(\mathcal{A})$ and $\text{Conv}(\mathcal{B})$. This implies that $\text{Conv}(\mathcal{A} \cap \mathcal{B}) \subseteq \text{Conv}(\mathcal{A}) \cap \text{Conv}(\mathcal{B})$. Furthermore, we also note that for any set $\mathcal{B}'$ such that $\mathcal{B} \subseteq \mathcal{B}'$, it must be true that $\text{Conv}(\mathcal{B}) \subseteq \text{Conv}(\mathcal{B}')$, which implies that $\text{Conv}(\mathcal{A} \cap \mathcal{B}) \subseteq \text{Conv}(\mathcal{A}) \cap \text{Conv}(\mathcal{B}')$.

By applying these two arguments, $\text{Conv}(\mathcal{Y})$ must be a subset of the intersection of sets (i) $\text{Relax}(\mathcal{Y})$, (ii) $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}^{\mathcal{J}_i}))$ $\forall \mathcal{J}_i \in \{\mathcal{J}_1^1, \ldots, \mathcal{J}_1^k\}$, and (iii) $\text{Proj}_{\mathbf{y}}(\text{NF}(\mathcal{D}^{\mathcal{J}_i}_{\texttt{outer}}))$ $\forall \mathcal{J}_i \in \{\mathcal{J}_1^2, \ldots, \mathcal{J}_q^2\}$. In other words, the intersection of these three sets is a valid outer approximation of $\text{Conv}(\mathcal{Y})$. □

EC.2. Details of the Capital Budgeting Problem

The complete capital budgeting problem from Arslan and Detienne (2022) is

$$\max_{(\mathbf{x},x_0)\in\mathcal{X}} \min_{\xi\in\Xi} \max_{(\mathbf{y},y_0)\in\mathcal{Y}\cap\mathcal{S}(\mathbf{x})} \sum_{i\in\mathcal{N}}\left(\sum_{j=1}^{M}\frac{Q_{ij}\xi_j}{2}\right)\bar{p}_i((1-f)x_i+fy_i)+\sum_{i\in\mathcal{N}}\bar{p}_i((1-f)x_i+fy_i)-\gamma x_0-\gamma\mu y_0$$

where $\Xi := [-1,1]^M$ and where

$$\mathcal{Y}\cap\mathcal{S}(\mathbf{x}) = \left\{(\mathbf{y},y_0,w_0)\in\{0,1\}^{N+2} \;\middle|\; \mathbf{c}^\top\mathbf{y}\le B+C_1w_0+C_2y_0,\ w_0=x_0,\ y_i\ge x_i\ \forall i\in\mathcal{N}\right\}.$$

Note that w_0 is an auxiliary variable that is used to maintain the selective adaptability condition.

EC.2.1. Deriving the network flow formulation

We show a step-by-step process to obtain our network flow formulation. First, recall that if we had a description for $\text{Conv}(\mathcal{Y}\cap\mathcal{S}(\mathbf{x}))$, then the problem can be rewritten as

$$\max_{\substack{(\mathbf{x},x_0)\in\mathcal{X}\\(\mathbf{y},y_0)\in\text{Conv}(\mathcal{Y}\cap\mathcal{S}(\mathbf{x}))}} -\gamma x_0-\gamma\mu y_0+\sum_{i\in\mathcal{N}}\bar{p}_i((1-f)x_i+fy_i)+\min_{\xi\in\Xi}\sum_{i\in\mathcal{N}}\left(\sum_{j=1}^{M}\frac{Q_{ij}\xi_j}{2}\right)\bar{p}_i((1-f)x_i+fy_i).$$

Given the uncertainty set $\Xi := [-1,1]^M$, the adversarial problem can be written as

$$\begin{aligned}\min_{\boldsymbol{\xi}}\quad & \sum_{i\in\mathcal{N}}\left(\sum_{j=1}^{M}\frac{Q_{ij}\xi_j}{2}\right)\bar{p}_i((1-f)x_i+fy_i)\\ \text{s.t.}\quad & \xi_i\ge -1, \qquad \forall i\in\mathcal{M}\\ & \xi_i\le 1, \qquad \forall i\in\mathcal{M},\end{aligned}$$

where $\mathcal{M}=\{1,\dots,M\}$. The dual of this problem is

$$\begin{aligned}\max_{\boldsymbol{\lambda}^1,\boldsymbol{\lambda}^2}\quad & -\sum_{i\in\mathcal{M}}\lambda^1_m+\lambda^2_m\\ \text{s.t.}\quad & \lambda^1_m-\lambda^2_m\le\sum_{i\in\mathcal{N}}\left(\frac{Q_{i,m}}{2}\right)\bar{p}_i((1-f)x_i+fy_i)\quad \forall m\in\mathcal{M}\\ & \boldsymbol{\lambda}^1,\boldsymbol{\lambda}^2\ge\mathbf{0}.\end{aligned}$$

Finally, combining this inner maximization problem with the outer maximization problem, we obtain our final network flow formulation of

$$
\begin{aligned}
\max_{(\mathbf{x},x_0),(\mathbf{y},y_0),\mathbf{z},\boldsymbol{\lambda}} \quad & -\gamma x_0 + \sum_{i\in\mathcal{N}} \bar{p}_i((1-f)x_i + f y_i) - (\sum_{m\in\mathcal{M}} \lambda_m^1 + \lambda_m^2) - \gamma\mu y_0 \\
\text{s.t.} \quad & \lambda_m^1 - \lambda_m^2 = \sum_{i\in\mathcal{N}} \left(\frac{Q_{i,m}}{2}\right) \bar{p}_i((1-f)x_i + f y_i) \quad \forall m \in \mathcal{M}, \\
& \boldsymbol{A}\mathbf{z} = \mathbf{b}, \\
& y_i = \sum_{j\in l^+(i)} z_j, \quad \forall i \in \mathcal{N} \\
& y_0 = \sum_{j\in l^+(N-1)} z_j, \\
& w_0 = \sum_{j\in l^+(N)} z_j, \\
& \mathbf{y} \geq \mathbf{x}, \\
& w_0 = x_0, \\
& \mathbf{c}^\top \mathbf{x} \leq B + C_1 x_0, \\
& (\mathbf{x}, x_0) \in \{0,1\}^{N+1}, \\
& \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \mathbf{z} \geq \mathbf{0}.
\end{aligned}
\tag{EC.1}
$$

EC.2.2. Deriving the K-adaptability formulation

We derive the K-adaptability formulation for the capital budgeting problem, which is based on Theorem 2 of Hanasusanto et al. (2015). For a given solution $(\mathbf{x}, \mathbf{y}^1, \ldots, \mathbf{y}^K)$, the adversarial problem can be written as

$$
\begin{aligned}
\min_{\boldsymbol{\xi},\tau} \quad & \tau \\
\text{s.t.} \quad & \tau \geq -\gamma x_0 - \gamma\mu y_0^k + \sum_{i\in\mathcal{N}} \left(\sum_{j=1}^{M} \frac{Q_{ij}\xi_j}{2}\right) \bar{p}_i((1-f)x_i + f y_i^k) + \sum_{i\in\mathcal{N}} \bar{p}_i((1-f)x_i + f y_i^k), \quad \forall k \in \mathcal{K} \\
& \xi_i \geq -1, \qquad \forall i \in \mathcal{M} \\
& \xi_i \leq 1, \qquad \forall i \in \mathcal{M}.
\end{aligned}
$$

The dual of this problem is

$$
\begin{aligned}
\min_{\boldsymbol{\lambda}^1,\boldsymbol{\lambda}^2,\boldsymbol{\pi}} \quad & -(\sum_{m\in\mathcal{M}} \lambda_m^1 + \lambda_m^2) + \sum_{k=1}^{K} \pi_k \left(-\gamma x_0 - \gamma\mu y_0^k + \sum_{i\in\mathcal{N}} \bar{p}_i((1-f)x_i + f y_i^k) \right) \\
\text{s.t.} \quad & \lambda_m^1 - \lambda_m^2 = \sum_{k=1}^{K} \sum_{i\in\mathcal{N}} \left(\frac{Q_{i,m}}{2}\right) \bar{p}_i((1-f)x_i\pi_k + f y_i^k \pi_k) \quad \forall m \in \mathcal{M} \\
& \sum_{k=1}^{K} \pi_k = 1 \\
& \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2 \in \mathbb{R}^m, \boldsymbol{\pi} \in \mathbb{R}^k.
\end{aligned}
$$

Note that since $\sum_{k=1}^{K} \pi_k = 1$, the model is equivalent to

$$\begin{aligned}
\min_{\boldsymbol{\lambda}^1,\boldsymbol{\lambda}^2,\boldsymbol{\pi}} \quad & -(\sum_{m\in\mathcal{M}} \lambda_m^1 + \lambda_m^2) - \gamma x_0 + (1-f)\sum_{i\in\mathcal{N}} \bar{p}_i x_i + \sum_{k=1}^{K}\sum_{i\in\mathcal{N}} f\bar{p}_i y_i^k \pi_k - \gamma\mu y_0^k \pi_k \\
\text{s.t.} \quad & \lambda_m^1 - \lambda_m^2 = \sum_{i\in\mathcal{N}} \left(\frac{Q_{i,m}}{2}\right)\bar{p}_i(1-f)x_i + \sum_{k=1}^{K}\sum_{i\in\mathcal{N}} \left(\frac{Q_{i,m}}{2}\right)\bar{p}_i f y_i^k \pi_k \quad \forall m\in\mathcal{M} \\
& \sum_{k=1}^{K} \pi_k = 1 \\
& \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2 \in \mathbb{R}_+^m, \boldsymbol{\pi} \in \mathbb{R}_+^k.
\end{aligned}$$

Finally, because of the bilinearity, we can replace every instance of $\pi_k y_i^k$ with q_i^k. We thus arrive at the final K-adaptability formulation of

$$\begin{aligned}
\min \quad & -(\sum_{m\in\mathcal{M}} \lambda_m^1 + \lambda_m^2) - \gamma x_0 + (1-f)\sum_{i\in\mathcal{N}} \bar{p}_i x_i + \sum_{k=1}^{K}\sum_{i\in\mathcal{N}} f\bar{p}_i q_i^k - \gamma\mu q_0^k \\
\text{s.t.} \quad & \lambda_m^1 - \lambda_m^2 = \sum_{i\in\mathcal{N}} \left(\frac{Q_{i,m}}{2}\right)\bar{p}_i(1-f)x_i + \sum_{k=1}^{K}\sum_{i\in\mathcal{N}} \left(\frac{Q_{i,m}}{2}\right)\bar{p}_i f q_i^k, && \forall m\in\mathcal{M} \\
& q_i^k \le y_i^k, q_i^k \le \pi_k, q_i^k \ge \pi_k + y_i^k - 1, && \forall i\in\{1,\dots,|\mathcal{S}|\}, k\in\{1,\dots,K\} \\
& \mathbf{c}^\top \mathbf{y}^k \le B + C_1 x_0 + C_2 y_0^k, && \forall k\in\{1,\dots,K\} \\
& \mathbf{y}^k \ge \mathbf{x}, && \forall k\in\{1,\dots,K\} \\
& \mathbf{c}^\top \mathbf{x} \le B + C_1 x_0 \\
& \sum_{k=1}^{K} \pi_k = 1 \\
& \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2 \in \mathbb{R}_+^m,\ \boldsymbol{\pi} \in \mathbb{R}_+^k,\ \mathbf{q}^1,\dots,\mathbf{q}^k \in \mathbb{R}_+^n \\
& \mathbf{x}, \mathbf{y}^1, \dots, \mathbf{y}^k \in \{0,1\}^n.
\end{aligned}$$

EC.2.3. A brief comparison with branch-and-price results

We briefly compare the results of our network flow models against the solution times presented in Arslan and Detienne (2022) for the capital budgeting instances. As discussed in the main body of the paper, branch-and-price algorithms represent a family of methods that can be effective if implemented properly, but may require significant implementation effort (e.g., fine-tuning of many intermediate steps, see Cire et al. (2019), de Lima et al. (2022)). The main purpose of our comparison is simply to show that our approach, which is model-based and can be solved directly using any standard solver, is a viable and promising alternative for solving these problems.

Table EC.1 highlights the total time to compile and solve the network flow models, as well as the solution time of the branch-and-price (B&P) algorithm reported in Arslan and Detienne (2022).

Instances	$Q=0$	$Q=1$	$Q=3$	$Q=5$	B&P
n = 10	0.1s	-	-	-	0.1s
n = 20	4s	-	-	-	1.8s
n = 30	104s	16s (0.3%)	-	-	> 268s
n = 40	589s	78s (0.1%)	12s (0.4%)	-	> 451s
n = 50	-	150s (0.08%*)	30s (0.2%*)	12s (0.3%*)	116s

Table EC.1 Solution time of the network flow models compared to the branch-and-price algorithm. Notes: Solution times of B&P algorithm are taken from Arslan and Detienne (2022). Two of the $n=30$ instances and five of the $n=40$ instances could not be solved within the one-hour time limit using the B&P algorithm. For the $n=30$ and $n=40$ instances, we give the true optimality gap of the solutions in the brackets. For the $n=50$ instances, we present the model-based optimality gap (denoted using $*$), which is an upper bound on the true optimality gap.

For the network flow models, the times are taken directly from Table 2 in the main body of the paper, and we censor some values here for ease of discussion. The values in the brackets next to the solution times denote the average optimality gap of the solutions, which are taken directly from Table 3. We remind the reader that there are 60 instances for each value of $n=10,20,30,40,50$.

First, we observe that for the $n=10$ and $n=20$ instances, both methods can solve the instances exactly within a few seconds. For the $n=30$ instances, our exact network flow model takes an average of 104 seconds to solve while the B&P algorithm takes an average of 268 seconds with 2 instances that cannot be solved within the one-hour time limit imposed in the paper. For the instances where $n=40$ and $n=50$, our exact formulations become more difficult to compile and solve, and their solution times often exceed those of the B&P algorithm. However, the approximation techniques in our paper provide a method for drastically reducing solution times while sacrificing little in solution quality. For example, for the $n=40$ instances, we can generate significantly smaller network flow models with $Q=1$ ($Q=3$) that solves in an average of 78 (12) seconds while generating solutions that have an average optimality gap of 0.1% (0.4%). In comparison, the B&P algorithm requires an average of 451 seconds with 5 instances that cannot be solved within the one-hour time limit. Similar observations can be made for the $n=50$ instances.

In summary, we remark that both exact approaches will become more difficult as the size of instances increase (this will naturally be the case for any exact approach for solving ARBO problems). We observe that the solution times of our approach are comparable to those of branch-and-price, but more importantly, we have proposed a rigorous framework for generating approximate formulations that are significantly faster to solve while sacrificing little in terms of solution quality.

EC.3. Details of the Robust Assignment Problem

EC.3.1. Recursive formulation

In order to generate the exact decision diagram for the set of constraints

$$\begin{aligned}
&\sum_{\ell\in\mathcal{L}(m)} a_\ell y_{\ell,m} \le b_m, \quad \forall m\in\{1,\dots,M\}\\
&\sum_{m\in\mathcal{M}(\ell)} y_{\ell,m} \le 1, \qquad \forall \ell\in\{1,\dots,L\}\\
&\mathbf{y}\in\{0,1\}^{|\mathcal{S}|},
\end{aligned}$$

we can extend Example 1 to consider state vectors. For simplicity, we abuse notation slightly and consider decision vector $\mathbf{y}$ in the form of $\mathbf{y}=[y_1,\dots,y_{|\mathcal{S}|}]$, where for some element $y_{k=(\ell,m)}$, $agent(y_k)$ is a zero vector with a single entry $a_\ell y_k$ at the m-th element, where $task(y_k)$ is a zero vector with a single entry y_k at the ℓ-th element. The recursive formulation can then be given with initial state $\boldsymbol{S}_1=\mathbf{0}$, state-transition function $T_i(\boldsymbol{S},y_k)=\boldsymbol{S}+[agent(y_k);task(y_k)]$ and feasible action space $\mathcal{Q}_i(\boldsymbol{S})=\{y_k\in\{0,1\}|T_i(\boldsymbol{S},y_k)\le[\mathbf{b};\mathbf{1}]\}$.

EC.3.2. Complete network flow formulation

Recall that the robust assignment problem considered is a max-min-max problem. Through a simple transformation, we can consider an equivalent min-max-min formulation (i.e., by applying a negative in the objective function and taking the negative of the optimal objective value). We use this min-max-min model to formulate our network flow problem. We show the step-by-step process below.

Recall that the uncertainty set to the robust assignment problem is defined as

$$\Xi=\left\{\boldsymbol{\xi}\;\middle|\;\sum_{i=1}^{|\mathcal{S}|}\left|\xi_i/\xi_i^0-1\right|\le 0.1|\mathcal{S}|,\ \left|\xi_i/\xi_i^0-1\right|\le 0.5\ \ \forall i\in\{1,\dots,|\mathcal{S}|\}\right\}.$$

Given this uncertainty set, we can then write the adversarial problem as

$$\begin{aligned}
\max_{\boldsymbol{\xi},\mathbf{w}}\quad & -\boldsymbol{\xi}^\top\mathbf{y}\\
\text{s.t.}\quad & w_i\ge \xi_i/\xi_i^0-1, && \forall i\in\{1,\dots,|\mathcal{S}|\}\\
& w_i\ge 1-\xi_i/\xi_i^0, && \forall i\in\{1,\dots,|\mathcal{S}|\}\\
& w_i\le 0.5, && \forall i\in\{1,\dots,|\mathcal{S}|\}\\
& \sum_{i=1}^{|\mathcal{S}|} w_i\le 0.1|\mathcal{S}|.
\end{aligned}$$

The dual of this problem is

$$\min\quad 0.1|\mathcal{S}|\alpha+\sum_{i=1}^{|\mathcal{S}|}(\lambda_i^1-\lambda_i^2+0.5\pi_i)$$

$$\begin{aligned}
\text{s.t.} \quad & \lambda_i^1 - \lambda_i^2 = -\xi_i^0 y_i, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \pi_i - \lambda_i^1 - \lambda_i^2 + \alpha = 0, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \boldsymbol{\pi} \in \mathbb{R}_+^{|\mathcal{S}|}, \ \alpha \geq 0.
\end{aligned}$$

Finally, the complete network flow formulation of the adaptive robust optimization problem is

$$\begin{aligned}
\min \quad & 0.1|\mathcal{S}|\alpha + \sum_{i=1}^{|\mathcal{S}|}(\lambda_i^1 - \lambda_i^2 + 0.5\pi_i) \\
\text{s.t.} \quad & \lambda_i^1 - \lambda_i^2 = -\xi_i^0 y_i, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \pi_i - \lambda_i^1 - \lambda_i^2 + \alpha = 0, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \mathbf{y} \in \text{Proj}(\text{NF}(\mathcal{D})) \\
& \mathbf{y} \leq \mathbf{x} \\
& ||\mathbf{x}||_1 \leq \beta \\
& \mathbf{x} \in \{0,1\}^{|\mathcal{S}|}, \ \mathbf{y}, \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \boldsymbol{\pi} \in \mathbb{R}_+^{|\mathcal{S}|}, \ \alpha \geq 0.
\end{aligned}$$

EC.3.3. K-adaptability formulation

We similarly outline the step-by-step process for formulating the K-adaptability model, which is based on Hanasusanto et al. (2015). First, for a given solution $(\mathbf{x}, \mathbf{y}^1, \dots, \mathbf{y}^K)$, the epigraph formulation of the adversarial maximization problem is

$$\begin{aligned}
\max_{\boldsymbol{\xi}, \mathbf{w}, \tau} \quad & \tau \\
\text{s.t.} \quad & \tau \leq -\boldsymbol{\xi}^\top \mathbf{y}^k && \forall k \in \{1,\dots,K\} \\
& w_i \geq \xi_i/\xi_i^0 - 1, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& w_i \geq 1 - \xi_i/\xi_i^0, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& w_i \leq 0.5, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \sum_{i=1}^{|\mathcal{S}|} w_i \leq 0.1|\mathcal{S}|.
\end{aligned}$$

This problem has a dual of

$$\begin{aligned}
\min \quad & 0.1|\mathcal{S}|\alpha + \sum_{i=1}^{|\mathcal{S}|}(\lambda_i^1 - \lambda_i^2 + 0.5\pi_i) \\
\text{s.t.} \quad & \lambda_i^1 - \lambda_i^2 = -\sum_{k=1}^{K} \xi_i^0 \mu_k y_i^k, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \pi_i - \lambda_i^1 - \lambda_i^2 + \alpha = 0, && \forall i \in \{1,\dots,|\mathcal{S}|\} \\
& \sum_{k=1}^{K} \mu_k = 1 \\
& \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \boldsymbol{\pi} \in \mathbb{R}_+^{|\mathcal{S}|}, \ \boldsymbol{\mu} \in \mathbb{R}^K, \ \alpha \geq 0.
\end{aligned}$$

Note that there are bilinear terms of the form $\mu_k y_i^k$. However, since $\mathbf{y}^k \in \{0,1\}^n$, we can replace each bilinear term with auxiliary variable q_i^k and add constraints

$$q_i^k \leq y_i^k, q_i^k \leq \mu_k, q_i^k \geq \mu^k + y_i^k - 1, \quad \forall i \in \{1,\ldots,|\mathcal{S}|\}, k \in \{1,\ldots,K\}.$$

Since the outer problem of choosing $\mathbf{x}, \mathbf{y}^1, \ldots, \mathbf{y}^K$ is also a minimization problem, we obtain the following K-adaptability formulation

$$\begin{aligned}
\min \quad & 0.1|\mathcal{S}|\alpha + \sum_{i=1}^{|\mathcal{S}|}(\lambda_i^1 - \lambda_i^2 + 0.5\pi_i) \\
\text{s.t.} \quad & \sum_{k=1}^{K} \mu_k = 1 \\
& \lambda_i^1 - \lambda_i^2 = -\sum_{k=1}^{K} \xi_i^0 q_i^k, && \forall i \in \{1,\ldots,|\mathcal{S}|\} \\
& \pi_i - \lambda_i^1 - \lambda_i^2 + \alpha = 0, && \forall i \in \{1,\ldots,|\mathcal{S}|\} \\
& q_i^k \leq y_i^k, q_i^k \leq \mu_k, q_i^k \geq \mu_k + y_i^k - 1, && \forall i \in \{1,\ldots,|\mathcal{S}|\},\ k \in \{1,\ldots,K\} \\
& \mathbf{y}^k \leq \mathbf{x}, && \forall k \in \{1,\ldots,K\} \\
& ||\mathbf{x}||_1 \leq \beta \\
& \mathbf{y}^1, \ldots, \mathbf{y}^K \text{ are feasible assignments} \\
& \mathbf{x}, \mathbf{y}^1, \ldots, \mathbf{y}^K, \in \{0,1\}^{|\mathcal{S}|} \\
& \mathbf{q}^1, \ldots, \mathbf{q}^K, \boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2, \boldsymbol{\pi} \in \mathbb{R}_+^{|\mathcal{S}|},\ \boldsymbol{\mu} \in \mathbb{R}^K,\ \alpha \geq 0.
\end{aligned}$$