Noname manuscript No.
(will be inserted by the editor)

SIPLIB 2.0

Stochastic Integer Programming Test Problem Library 2.0

Yongkyu Cho · Kibaek Kim · Cong Han Lim · James Luedtke · Jeffrey Linderoth

Received: date / Accepted: date

Abstract SIPLIB was first constructed in 2002 by Shabbir Ahmed and his colleagues. It has been providing a collection of test instances to facilitate computational and algorithmic research in SIP. State-of-the-art in SIP combined with the speedup in computing machinery, however, has made many SIPLIB instances trivial. By SIPLIB 2.0 we provide richer collection of test instances with benchmarking computational results. We not only provide SMPS files but also a software package for generating/analyzing instances. The package for SIPLIB 2.0 is implemented in `Julia` programming language with structured algebraic modeling package StructJuMP (block-structured optimization framework for `Julia` mathematical programming).

Keywords SIPLIB · Stochastic Integer Programming · Problem Instances · Julia programming language

Yongkyu Cho
Department of Industrial and Management Engineering, Pohang University of Science and Technology, Gyeongbuk, Korea
E-mail: jyg1124@postech.ac.kr

Kibaek Kim
Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA
E-mail: kimk@anl.gov

Cong Han Lim · James Luedtke · Jeffrey Linderoth
Department of Industrial and Systems Engineering, University of Wisconsin-Madison Madison, WI 53706, USA
E-mail: clim9@wisc.edu; jim.luedtke@wisc.edu; linderoth@wisc.edu

Contents

List of Tables

List of Figures

1 Introduction

The SIPLIB [1] is an abbreviated term of the Stochastic Integer Programming (SIP) Library firstly contructed in 2002 by Shabbir Ahmed and his colleagues. The library has been providing a collection of test instances to facilitate computational and algorithmic research in SIP. Some new test problems with instances have been added to SIPLIB gradually and now it contains nine different problems in total.

At the time SIPLIB appeared, it provided enoughly large-sized instances that is reasonable to argue that the performance of algorithm is remarkable if it handles the instances well. State-of-the-art in SIP combined with the speedup in computing machinery, however, makes many instances in SIPLIB trivial so that we have not enough basis to use them for showing the excellence of newly suggested solution methods. At this point, we are motivated to develop the second version of SIPLIB say SIPLIB 2.0 that provides larger-sized test instances with higher degree of tailorability, e.g., users can easily generate instances of test problems as largely as they want.

Stochastic programming (SP) is a framework for modeling optimization problems that involve uncertainty. Whereas optimization problems are typically formulated with known parameters, the problems in real world contain some unknown parameters in many cases. For details on SP, see, e.g., [2,3]. SIP is a branch of SP that indicates any type of SP including at least one integer decision variable. We restrict our focus on two-stage SIP with linear objective throughout this paper and SIPLIB 2.0. The main reason is that the class of SIP is most widely used to model real world problems. Moreover, two-stage SIP itself has enough difficulties that have not been conquered yet even without any other details like chance-constraints and multi-stages. The main difficulty in solving two-stage SIP is that the second-stage value function is not necessarily convex. Thus, the standard decomposition approaches that work nicely for stochastic *linear* programs, break down when the second stage integer variables are present [4]. Hereinafter, we use the term SIP to indicate the two-stage SIP.

We provide SIPLIB 2.0 in two ways: SMPS files (`*.cor`, `*.tim`, `*.stoch`) and open-source `Julia` package (`Siplib.jl`). SMPS is a file format widely used to describe stochastic program instances. Once SMPS files of a problem instance are given, we can directly solve it using SIP solvers like DSP [5] and SMI [7]. A drawback of SMPS is low readability by human, which we decided to provide `Julia` scripts to let users be able to easily catch up the problems and tailor the instances.

`Julia` is an open source high-level, high-performance dynamic programming language for numerical computing. It is also known as nice performance, approaching that of statically-compiled languages like `C` [8]. The syntax of `Julia` is simple and should feel familiar to anyone who has experienced in another high-level languages like `MATLAB` or `Python`. A `Julia` package called JuMP (`Julia` for Mathematical Programming [9]) provides a domain-specific modeling language for mathematical optimization embedded in `Julia`. JuMP enables us to easily translate a paper-written mathematical models to a `JuMP.Model`-type object. Some structured mathematical models like SIP can also be translated to the `JuMP.Model`-type object by loading a structured modeling package StructJuMP [10]. Once we have a `Julia` script for constructing `JuMP.Model`-type object, it is easy to modify the original model. We implement a `Julia` package `Siplib.jl` to provide various functions for handling SIPLIB 2.0 instances for users' convenience. Those who feel the given instances are not enough can simply generate new instances using `Siplib.jl`.

The contributions of this work can be summarized as follows.

- We provide SIP-dedicated test instances that are more richer than the former SIPLIB.
- We collect, implement, summarize, and open all the problem-specific details: mathematical formulation, stochastic data generation, `Julia` scripts.
- We provide pre-analysis on the instances (component type, size, sparsity).
- We implement an open source `Julia` package `Siplib.jl` for handling instances: generation, analysis, and test.
- We provide well-summarized benchmark computational experiments results.

This paper is organized as follows. In Section 2, we briefly review SIP. This includes the mathematical formulation with notation, solution methods, and available software packages. In Section 3, we provide summary of the test problems. This consists of origin of the problems with brief description, instance naming rule, problem type based on its components (variables and constraints), the number of the components, and sparsity information. Together with the problem-specific description given in SectionA, we believe that users can quickly catch up what they need without investigating every detail. We present functions of a `Julia` package `Siplib.jl` in Section 4. Tutorials for utilizing the package are also accompanied. In Section 5, we report the computational results of the accompanied SMPS instances. We conclude this study with provision of future directions to improve this ongoing project in Section 6.

1.1 SMPS format for stochastic programming instances

SMPS format [23] is a data conventions for the automatic input of multiperiod stochastic linear programs. The input format is based on an old column-oriented format MPSX standard and is designed to promote the efficient conversion of originally deterministic problems by introducing stochastic variants in separate files.

Three input files are required to specify an SP in SMPS format:

- `.cor`: Core file written in MPS format. This describes the fundamental problem structure and contains the first-stage data and one second-stage scenario data.

– `.tim`: Time file which specifies the location where the 2nd stage begins.
– `.sto`: Stoch file which contains stochastic data of all scenarios except the one included in `.cor` file.

1.2 Literature review

The optimizers, not only SIP researchers, have always needed test instances to figure out the performances of newly developed methods. MIPLIB 2010 [21] for mixed integer programming (MIP) is a good example of such collection hence the main motivation of this study. MIPLIB 2010 comprises 361 instances. The authors collected the instances from various sources and categorized them into 8 groups after the computational experiments. MIPLIB 2010 provides instances in MPS format with a test engine developed to run different solvers in a defined way to check the answers for consistency.

For stochastic programming, to the best of our knowledge, the first such approach is Holmes and Birge's *portable stochastic programming test set* (POSTS) [24] since 1994. POSTS is still available now and provides a small test set of stochastic programming recourse problems in SMPS format. POSTS consists of 15 stochastic linear programming problems but not dedicated to SIP. Birge provides some computational results of the POSTS instances [25], but most of them needs to be updated. Moreover, analysis on the instances does not seem to be enough.

Ariyawansa and Felt [26] have constructed a test problem collection for stochastic linear programming since 2001 and published a paper on it. Unlike Holmes and Birge's work, the authors provide an accompanied document explaining short description, mathematical problem statement, and notational reconciliation to a standard problem format for each of the 9 problems. Despite its name, Ariyawansa and Felt's collection also includes 3 problems that contain mixed integer variables. However, still the library is not dedicated to SIP and size of the instances are not large enough to perform intensive computational experiments.

The first SIP-oriented instance collection is the SIPLIB [1] constructed in 2002 by Shabbir Ahmed and his colleagues. The instances are basically given in SMPS format accompanied with simple information on the problem and computational experiment. Although SIPLIB provides basic ingredient to be exploited for SIP research, it has rooms for improvement. First, SIPLIB needs to be polished systematically in terms of both pre-analysis on the instances and post-analysis on the benchmark results. Currently, there is no predefined contribution rule for SIPLIB so different problem provides different information. Therefore, interpretation on the new results can be inconsistent highly depending on researchers. Second, SIPLIB only provides static instance files and sometimes does not provide ready-made instance files, which limits usability of the library. Moreover, the precise information for implementing the original problem is sometimes not allowed. Third, we need more problems of various types. Considering three types of variables (continuous, binary, and integer) and two stages, the possible number of combination is $\left[\sum_{k=1}^{3}\binom{3}{k}\right]^2 = 49$ in total while SIPLIB provides only 5 such combinations regarding the problems that can be fully implementable based on the open information.

2 Stochastic integer programming

In this section, we explain general description of SIP. This includes formal mathematical formulation, existing general solution methods to solve the SIPs, and currently available software libraries.

2.1 Formulation

In this subsection, we introduce the form of SIP of interest. The notations and dimensional information are summarized in Table 1. We are interested in finding solution for two-stage SIP of the form:

$$z := \min_{x \in X} \left\{ c^\top x + \mathcal{Q}(x) : \; Ax \geq b \right\}, \tag{1}$$

where $\mathcal{Q}(x) := \mathbb{E}_{\boldsymbol{\xi}} \left[\phi \left(h(\boldsymbol{\xi}) - T(\boldsymbol{\xi}) x \right) \right]$ is the recourse function associated with the random variable (r.v.) $\boldsymbol{\xi}$. We assume that $\boldsymbol{\xi}$ follows a known discrete probability distribution with the finite realizations, called *scenarios*, $\xi_1, \cdots, \xi_r$ and respective nonnegative probabilities $\mathbb{P}(1), \cdots, \mathbb{P}(r)$, i.e., $\mathbb{P}(s) \equiv \mathbb{P}[\boldsymbol{\xi} = \xi_s]$ for $s \in \mathcal{S} := \{1, \ldots, r\}$. When the distribution is continuous, we assume that we can reasonably approximate it by a suitably discretized distribution. The real-valued map $\phi_{\xi_s} : \mathbb{R}^{m_2} \rightarrow \mathbb{R}$ is the optimal value of the second-stage problem defined by

$$\phi_{\xi_s}(t) := \min_{y_s \in Y} \left\{ q(\xi_s)^\top y_s : \; W(\xi_s) y_s \geq t \right\}, \; t \in \mathbb{R}^{m_2}, \tag{2}$$

where ξ_s is an arbitrarily realized scenario. The sets $X \subseteq \mathbb{R}^{n_1}$ and $Y \subseteq \mathbb{R}^{n_2}$ represent integer or binary restrictions on a subset of the decision variables x and y_s, respectively. The first-stage problem data comprise A, b, and c. The second-stage data are given by $T(\xi_s)$, $W(\xi_s)$, $h(\xi_s)$, and $q(\xi_s)$ (for dimensional information refer to Table 1). Hereinafter, we use the simplified notations (T_s, W_s, h_s, q_s). The SIP (1) can be rewritten in the extensive form

$$\begin{aligned}
z = \min_{x, y_s} \; & c^\top x + \sum_{s=1}^{r} \mathbb{P}(s)(q_j^\top y_s), && \text{(3a)} \\
\text{s.t. } \; & Ax \geq b, && \text{(3b)} \\
& T_s x + W_s y_s \geq h_s, \quad \forall s \in \{1, \ldots, r\}, && \text{(3c)} \\
& x \in X, && \text{(3d)} \\
& y_s \in Y, \quad \forall s \in \{1, \ldots, r\}. && \text{(3e)}
\end{aligned}$$

Table 1: Summary of notations in SIP formulation

Sets:	
$X \subseteq \mathbb{R}^{n_1}$	first-stage polyhedral set (continuous, integer, binary)
$Y \subseteq \mathbb{R}^{n_2}$	second-stage polyhedral set (continuous, integer, binary)
$\mathcal{S} = \{1, \ldots, r\}$	index set of realizable scenarios
Scalas:	
$\boldsymbol{\xi}$	r.v. denoting scenario that realizes by one of the set $\{\xi_1, \cdots, \xi_r\}$
$z \in \mathbb{R}$	optimal objective value of the SIP
$r \in \mathbb{N}$	number of scenarios
$s \in \mathcal{S}$	index denoting scenario
$\mathbb{P}(s) \in [0, 1]$	probability that scenario s happens, i.e., $\mathbb{P}(s) \equiv \mathbb{P}[\boldsymbol{\xi} = \xi_s]$
Vectors:	
$x \in \mathbb{R}^{n_1}$	first-stage decision vector
$c \in \mathbb{R}^{n_1}$	first-stage cost vector
$b \in \mathbb{R}^{m_1}$	first-stage RHS vector
$y_s \in \mathbb{R}^{n_2}$	second-stage decision vector under scenario ξ_s
$q_s \equiv q(\xi_s) \in \mathbb{R}^{n_2}$	second-stage cost vector
$h_s \equiv h(\xi_s) \in \mathbb{R}^{m_2}$	second-stage RHS vector
Matrices:	
$A \in \mathbb{R}^{m_1 \times n_1}$	first-stage constraint matrix corresponds to decision vector x
$W_s \equiv W(\xi_s) \in \mathbb{R}^{m_2 \times n_2}$	second-stage constraint matrix corresponds to decision vector y_s
$T_s \equiv T(\xi_s) \in \mathbb{R}^{m_2 \times n_1}$	second-stage constraint matrix corresponds to decision vector x
Functions:	
$\phi_{\xi_s} : \mathbb{R}^{m_2} \to \mathbb{R}$	second stage program optimal value under the realization of scenario ξ_s
$\mathcal{Q} : \mathbb{R}^{n_1} \to \mathbb{R}$	recourse function (the expectation of $\phi\,(h(\boldsymbol{\xi}) - T(\boldsymbol{\xi})x)$ over the r.v. $\boldsymbol{\xi}$)

2.2 Solution methods

Assuming finite and discrete distribution, much of the solution methods in SIP is studied to resolve the difficulty of optimizing the form in Equation 1, that is, the sum of the first stage costs and the expected costs in the second stage. Most methods also assume that a single scenario evaluation of SIP is somehow tractable. In this section, we introduce representative approaches to resolve the difficulties in SIP considering many scenarios.

2.2.1 Stage-wise decomposition

This type of algorithms exploit the natural viewpoint of optimizing the objective function in Equation (1) over the set of feasible first-stage decisions. The method used most frequently are based on building an outer linearization of the recourse cost function and a solution of the first-stage problem plus this linearization. This cutting plane technique is called the *L-shaped method* or *Benders decomposition* [31], from which the variants of stage-wise decomposition methods are derived.

For SIP with pure binary first-stage variables and mixed integer second-stage, the integer L-shaped method [32] approximates the second-stage recourse function by linear cuts that are exact at the binary solution where the cut is generated and are under-estimates at other binary solutions.

For SIP where the first-stage variables are not necessarily binary, dual functions from the second-stage integer program can be used to construct cuts to build the approximation of $\mathcal{Q}(x)$ [33].

2.2.2 Scenario-wise decomposition

In this type algorithms, we first introduce the copies of the first stage variables into the second stage. Then, the extensive form in (3) is changed as below.

$$z = \min_{x_s, y_s} \sum_{s=1}^{r} p_s \left(c^\top x_s + q_s^\top y_s \right) \tag{4a}$$

$$\text{s.t.} \sum_{s=1}^{r} H_s x_s = 0 \tag{4b}$$

$$(x_s, y_s) \in G_s, \quad \forall s \in \{1, \ldots, r\}, \tag{4c}$$

where the scenario feasibility set G_s is defined as

$$G_s := \{(x_s, y_s) : \; Ax_s \geq b, \; T_s x_s + W_s y_s \geq h_s, \; (x_s, y_s) \in X \times Y\}. \tag{5}$$

The newly added constraints (4b) (called *nonanticipativity*) guarantee $x_1 = x_r$ and $x_s = x_{s-1}$ for $s = 2, \ldots, r$ with a suitable $rn_1 \times n_1$ coefficient matrix H_s. This can be presented in another expression other than the one above that Carøe and Schultz [28] proposed.

Two representative scenario-wise decomposition algorithms are Dual Decomposition (DD, [28]) and Progressive Hedging (PH, [29]). The main idea of DD algorithm is to relax the nonanticipativity constraints using Lagrange multipliers and then to restore the nonanticipativity. Given set of multipliers, the problem is separable by scenarios hence the associated dual function can be optimized scenario-wise independently. Due to the nonconvexities, there can exist a duality gap between relaxed problem and the original problem so the optimality needs to be attained branch-and-bound scheme.

PH algorithm is developed by Rockafellar and Wets motivated by augmented Lagrangian theory. To briefly introduce the idea of PH, we define the following terms for solution systems.

- admissible: a solution that satisfies the constraints for all scenarios
- implementable (or nonanticipative): a solution in which the first stage decisions are indifferent over all scenarios
- feasible: a solution that is both admissible and implementable

The basic idea of PH can be summarized by the following procedure [30].

1. For each scenario s, solutions are obtained for the single-scenario problem of minimizing, subject to the problem constraints, the formulation (4a)-(4c), and (5).
2. The variable values for an implementable—but likely not admissible—solution are obtained by averaging over all scenarios at a scenario tree node.

3. For each scenario s, solutions are obtained for the problem of minimizing, subject to the problem constraints, the deterministic solution as in Step 1. plus terms that penalize the lack of implementability using a subgradient estimator for the nonanticipativity constraints and a squared penalty term.
4. If the solutions have not converged sufficiently and the allocated compute time is not exceeded, goto Step 2.
5. Post-process, if needed, to produce a fully admissible and implementable solution.

One advantage of scenario-wise decomposition over the stage-wise methods is their ability to relieve the computational burden associated with large number of scenarios by decomposing the problem by scenario and solving the descendant subproblems in parallel manners.

2.3 Software libraries

Assuming discrete distribution, all SIP can be represented by any algebraic modeling languages and then solved by any available MIP solvers. However, such manual implementation without exploiting structural characteristics in SIP can often result in inefficient computation and unnecessary memory allocation. In this subsection, we introduce some open-source software libraries dedicated to SIP.

2.3.1 Modeling language

We introduce two relatively new algebraic SIP modeling languages which are compatible with general purpose high-level programming languages `Julia` and `Python`. Both modeling libraries allows non-specialists to easily write mathematical model on the computing environment.

StructJuMP is a `Julia` package provides a parallel algebraic modeling framework for block-structured optimization models. StructJuMP is an extension of the JuMP modeling package (`Julia` for Mathematical Optimization), which is faster than any other modeling tools. StructJuMP enables JuMP to express the structure of problems and efficiently interface with structure-exploiting solvers. It also works in parallel (distributed memory), and thus allows the specification of much larger (structured) problems than JuMP can handle.

PySP is an extension of a `Python`-based algebraic modeling package Pyomo. To formulate a stochastic program in PySP, the user specifies both the deterministic base model and the scenario tree with associated uncertain parameters in Pyomo. Given these two models, PySP can directly solve the stochastic program using two ways: extensive form and PH algorithm.

2.3.2 Solver

We introduce two popular open source solver packages: DSP and PySP.

DSP is an open-source SIP solver implemented in `C++`. DSP reads SMPS files as input and provides three ways to solve a SIP instance.

- Invoking standard MIP solver to solve the extensive form SIP
- L-shape method (Benders decomposition)
- Dual Decomposition algorithm

DSP provides parallel implementations for the two decomposition-based algorithms, which allows users fully exploit computing clusters and multi-core processors. DSP also provides `Julia` interface to improve its usability.

PySP is a stochastic programming extension to Pyomo, a `Python`-based open source software package that supports variants of mathematical optimization. PySP enables the expression of stochastic programming problems as extensive form. PySP provides two ways to solve the instance.

- Invoking standard MIP solver to solve the extensive form SIP
- Progressive Hedging algorithm

PySP can also solve the extensive form SIP invoking standard MIP solver. To solve more complex and large-scale SIP, PySP implements PH algorithm, which provides an effective heuristic for approximating general multi-stage SIP.

3 The test problems

In this section, we explain information about the test problems in summarized manner. This includes problem origin, type, components (variables and constraints), and sparsity for each problem. This section reflects our philosophy in developing SIPLIB 2.0. Detailed problem-specific information is available in Section A for those who are interested in.

3.1 Origin of the problems

Table 2 summarizes the description of each test problem in SIPLIB 2.0. The five of them are adopted from SIPLIB [1]. We tried to implement the SIPLIB instances as the same as the original references possible. Not all of them, however, are exactly the same due to insufficient information on some parameters. We guess the missing links and develop our own way to implement the problems as long as it does not harm the endemic characteristic.

Table 2: Problems in SIPLIB 2.0

Problem	Description	Main reference
DCAP	Dynamic capacity planning with stochastic demand (A.1)	Ahmed and Garcia [4]
MPTSPs	Multi-path traveling salesman problem with stochastic travel costs (A.2)	Tadei et al. [13]
SIZES	Optimal product substitution with stochastic demand (A.3)	Jorjani et al. [15]
SMKP	Stochastic multiple knapsack problem (A.4)	Angulo et al. [17]
SSLP	Stochastic server location problem (A.5)	Ntaimo and Sen [20]
SUC	Stochastic unit commitment problem (A.6)	Papavasiliou and Oren [18]

3.2 Instance naming rule

Table 3 shows how we name the instances. We change the original naming convention for consistency and future extension. Some legacy naming rules do not consider the case when the set cardinality becomes larger than 1 digit number. Moreover, since some SIPLIB 2.0 instances can be generated using more sets other than used in SIPLIB, we needed to define a new naming convention. For example, we change the instance names of DCAP and SMKP as below.

$$\text{dcap}RNT\_\mathcal{S} \longrightarrow \text{DCAP}\_R\_N\_T\_\mathcal{S}$$
$$\text{smkp}\_\mathcal{S} \longrightarrow \text{SMKP}\_I\_\mathcal{S}$$

For DCAP, we just add underbars "_" to delimit set cardinalities. Without delimiter, the instance name causes confusion when set cardinality is greater than or equal to 10. For SMKP, we add new set cardinality I since the fixed number $|I| = 120$ can be changed by user if desired.

The capital Roman letters mean the sets defining the problems. In particular, the calligraphic letter $\mathcal{S}$ always denotes the scenario set. For notational convenience, we sometimes skip the cardinality sign $|\cdot|$ for sets, i.e., for set S, S itself denotes the number of elements $|S|$ in Table 3 and 5. Note that not all sets are used to define an instance. The sets that do not appear in the instance name are fixed by some pre-determined value by the original references so we follow them. For example, in SMKP there are four sets in total, $I, J, K, \mathcal{S}$, but the numbers of knapsacks $|J|$ and $|K|$ are fixed by 50 and 5 so do not appear in the instance name.

Table 3: Instance naming rules

Problem	Instance name	Remark
DCAP	DCAP_R_N_T_$\mathcal{S}$	R: number of resources, N: number of tasks, T: number of time periods, $\mathcal{S}$: number of scenarios
MPTSPs	MPTSPs_D_N_$\mathcal{S}$	D: node distribution strategy, N: number of nodes, $\mathcal{S}$: number of scenarios
SIZES	SIZES_$\mathcal{S}$	$\mathcal{S}$: number of scenarios
SMKP	SMKP_I_$\mathcal{S}$	I:number of types for item, $\mathcal{S}$: number of scenarios
SSLP	SSLP_I_J_$\mathcal{S}$	I: number of clients, J: number of server locations, $\mathcal{S}$: number of scenarios
SUC	SUC_D_$\mathcal{S}$	D: day type, $\mathcal{S}$: number of scenarios

3.3 Type of the problems

In SIPLIB 2.0 we mainly classify each problem by its stage-wise variable types. We consider three types of variable: continuous, binary, and integer. Considering two stages, the possible number of combination is $\left[\sum_{k=1}^{3}\binom{3}{k}\right]^2 = 49$ in total. We try to include problems with non-overlapping such combination. Table 4 shows the stage-wise components (variable and constraints) of each problem. For the abbreviated notation on the constraints, we refer MIPLIB 2010 [21]. Although the constraint type is possibly one of the important factors that define the problem characteristic, we decided not to consider it for classification since it can cause too much variety, which we cannot easily capture the insight from the problem type classification.

Table 4: Components of the problems

	1st stage		2nd stage	
Problem	Variable	Constraint	Variable	Constraint
DCAP (6)	$\mathbb{C}$, $\mathbb{B}$	`VBB`	$\mathbb{B}$	`PAR`, `M01`
MPTSPs (7)	$\mathbb{C}$, $\mathbb{B}$	`PAR`, `GEN`	$\mathbb{B}$	`GEN`
SIZES (8)	$\mathbb{I}$	`VBD`, `GEN`	$\mathbb{B}$, $\mathbb{I}$	`IKN`
SMKP (9)	$\mathbb{B}$	`KNA`	$\mathbb{B}$	`KNA`
SSLP (10)	$\mathbb{B}$	`IVK`, `GEN`	$\mathbb{C}$, $\mathbb{B}$	`GEN`
SUC (11)	$\mathbb{C}$, $\mathbb{B}$	`VBB`, `GEN`	$\mathbb{C}$, $\mathbb{B}$	`VBB`, `GEN`

*$\mathbb{C}$: continuous, $\mathbb{B}$: binary, $\mathbb{I}$: integer
**Constraint type notation is adopted from `MIPLIB 2010`. Refer to the tables in Section C.

3.4 Number of components

Table 5 summarizes the number of components (variables and constraints) in each problem from SIPLIB 2.0. The numbers can be calculated based on the cardinality of the sets that define the problems. Although there is no universally effective way to measure the difficulty of MIP yet, the number of components in instance is one of the closely related factor. For example, instances tend to be more difficult as the number of discrete variables increases. For those who want to generate instances with some desired number of components can utilize Table 5.

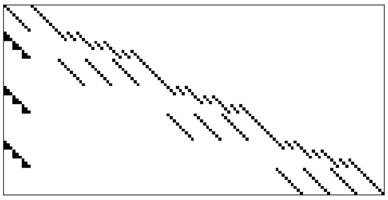
Table 5: Number of components in each problem

		Components			
		#Continuous	#Binary	#Integer	#Constraint
DCAP (8)	1st stage	RT	RT	-	RT
	2nd stage	-	$(1+R)\,NT$	-	$(R+N)T$
	Total	RT	$RT+(1+R)\,NT\mathcal{S}$	-	$RT+(R+N)T\mathcal{S}$
MPTSPs(9)	1st stage	$(N-1)N$	$(N-1)N$	-	N^2+2N-1
	2nd stage	-	$3(N-1)N$	-	$(N-1)N$
	Total	$(N-1)N$	$(N-1)(1+3\mathcal{S})N$	-	$(1+\mathcal{S})N^2+(2-\mathcal{S})N-1$
SIZES (10)	1st stage	-	$2N$	$2N$	$2(1+N)$
	2nd stage	-	-	$N(N+1)$	$4N$
	Total	-	$2N$	$2N+N(N+1)\mathcal{S}$	$2(1+N+2N\mathcal{S})$
SMKP (12)	1st stage	-	$2I$	-	J
	2nd stage	-	I	-	K
	Total	-	$(2+\mathcal{S})I$	-	$J+K\mathcal{S}$
SSLP (13)	1st stage	-	J	-	1
	2nd stage	J	IJ	-	$I+J$
	Total	$J\mathcal{S}$	$(1+I\mathcal{S})J$	-	$1+(I+J)\mathcal{S}$
SUC (14)	1st stage	960	1000	-	2208
	2nd stage	21274	2250	-	24780
	Total	$960+21274\mathcal{S}$	$1000+2250\mathcal{S}$	-	$2208+24780\mathcal{S}$

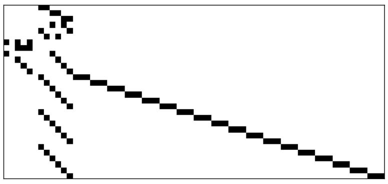
*For convenience, we skip the cardinality sign $|\cdot|$ for sets.
**We insert numerical value for the predetermined set, e.g., for SIZES, we use $|T|=2$ and for MPTSPs, we use $|K_{ij}|=3$. In SUC, all the sets are predetermined based on the given data except for the scenario set $\mathcal{S}$.

3.5 Sparsity

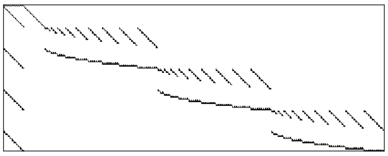
In section B, we append tables providing block-wise sparsity that is derived solely based on the set cardinality. Every SIP has a block-diagonal structure in its coefficient matrix of the extensive form (Fig 1). This characteristic differentiates SIP from the general MIP where the sparsity pattern varies instance by instance. In particular, the block-diagonal structure results always in very high sparsity (i.e., low density of non-zero values) as scenario increases. Unlike general MIP, it does not seem to be meaningful to just report the sparsity information of the extensive form coefficient matrix since decomposition-based algorithms in SIP can efficiently handle the sparsity.



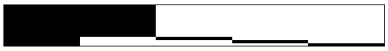
(a) DCAP_3_3_3_3



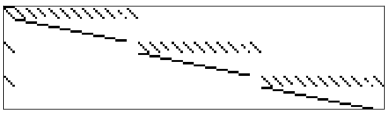
(b) MPTSPs_D0_3_3



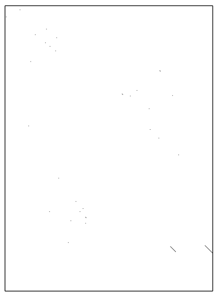
(c) SIZES_3



(d) SMKP_120_3



(e) SSLP_5_10_3



(f) SUC_FallWD_1

Fig. 1: Block-diagonal structure for each problem in extensive form
*SUC instance is too huge and extremely sparse to plot more than 1 scenario

As can be seen in Fig 2, there are three different blocks in terms of the structure. Every other block is the duplication of block T or W hence the same sparsity pattern repeats as many as the number of scenarios considered. Block A and W are only related with their own stage while block T is related with both. We call the block T *complicating* block.

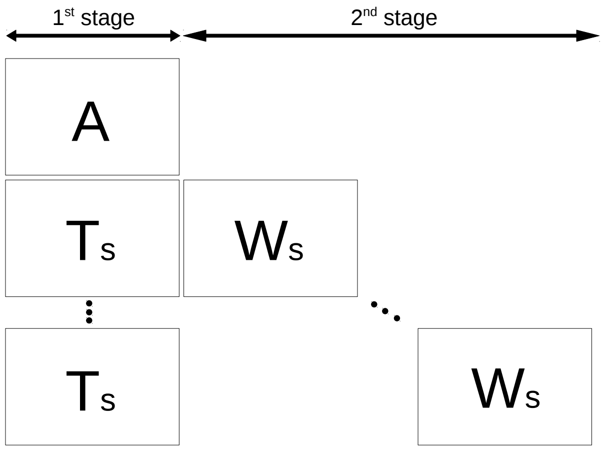


Fig. 2: Three (structurally) independent blocks in SIP

Dense coefficient matrix usually causes slowdown in decomposition algorithms. For example, the Dual Decomposition based solver DSP shows much slower convergence speed than the centralized solver CPLEX in problems like SMKP which always have low sparsity (nonzero ratio: 50%-100%, refer to Table 18).

4 `Siplib.jl`: A `Julia` package for SIPLIB 2.0

SIPLIB 2.0 is implemented in `Julia` programming language with algebraic modeling packages JuMP and StructJuMP. We implement and provide `Julia` package `Siplib.jl` for users to utilize ingredients of SIPLIB 2.0. In this section, we introduce how to use `Siplib.jl`.

To use `Siplib.jl`, we need to perform the following steps.

– install `Julia` $\geq$ version 0.6.2
– install `Julia` packages: `Distributions.jl`, `JuMP.jl`, `StructJuMP.jl`, `PyPlot.jl`
– change working directory to "`~/Siplib/src/`"
– run `Julia`
– excute `include("Siplib.jl")`
– excute `using Siplib`

4.1 Generating instances: `JuMP.Model`-object and `SMPS` files

`JuMP.Model`-type object is an object that contains every information of an instance. Hence, almost every function in `Siplib.jl` requires `JuMP.Model`-type object as one of its input arguments. `Siplib.jl` provides two functions to construct the `JuMP.Model`-type object of an instance.

```
function getJuMPModel(problem::Symbol, param_arr::Any)::JuMP.Model
function generateSMPS(problem::Symbol, param_arr::Any, DIR_NAME::String="$(
    dirname(@__FILE__))/../instance")::JuMP.Model
```

The first function `getJuMPModel` takes `Symbol`-typed argument `problem` and associated parameter array `param_arr`. Then, it constructs `JuMP.Model`-type object and return it. For example, the follwing command returns the `JuMP.Model` object of instance DCAP_3_4_2_100.

```
getJuMPModel(:DCAP, [3,4,2,100])
```

Keep in mind that the number of elements in `param_arr` should match with the problem as in Table 6, otherwise it prints a warning message.

Table 6: `problem` arguments and corresponding parameter array

`problem`	`param_arr`	Remark
`:DCAP`	`[R, T, N,` $\mathcal{S}$`]`	All parameters are integer.
`:MPTSPs`	`[D, N,` $\mathcal{S}$`]`	String `D` $\in$ {"D0", "D1", "D2", "D3"}
`:SIZES`	`[`$\mathcal{S}$`]`	Integer $\mathcal{S} \geq 20$.
`:SMKP`	`[I,` $\mathcal{S}$`]`	All parameters are integer.
`:SSLP`	`[I, J,` $\mathcal{S}$`]`	All parameters are integer.
`:SUC`	`[D,` $\mathcal{S}$`]`	String `D` $\in$ {"FallWD", "FallWE", "WinterWD", "WinterWE", "SpringWD", "SpringWE", "SummerWD", "SummerWE"}

The second function `generateSMPS` generates SMPS files as well as returns `JuMP.Model` object by taking one more argument `DIR_NAME` to indicate a directory where the files are stored. The SMPS files are stored in the default folder "~/Siplib/instance/" unless the argument `DIR_NAME` is specified. The file name is automatically generated using the arguments, e.g., `generateSMPS(:DCAP, [3, 4, 2, 100])` generates three files.

– DCAP_3_4_2_100.cor
– DCAP_3_4_2_100.tim
– DCAP_3_4_2_100.sto

Sometimes one might want to generate SMPS files using pre-declared `JuMP.Model` object. The function `writeSMPS` is defined to do such task.

```
function writeSMPS(model::JuMP.Model, INSTANCE::String="instance", DIR_NAME::
    String="$(dirname(@__FILE__))/../instance")
```

The function above takes `JuMP.Model` object as input argument and stores SMPS files into `DIR_NAME` folder with file name `INSTANCE`. The `String`-type arguments `INSTANCE` and `DIR_NAME` can be omitted since they have default values "`instance`" and "~/Siplib/instance/."

We also define a conventional function to return the instance name in `String`-type.

```
function getInstanceName(problem::Symbol, param_arr::Any)::String
```

4.2 Pre-analyzing instances: size, sparsity, plot

`Siplib.jl` provides pre-analysis functions for instances. By "size", we mean the number of components (continuous, binary, integer, constraint) in an instance. As we discussed in Section B, sparsity is analyzed in block-wisely. The size and sparsity information is stored in the object of the following types: `Size` and `Sparsity`.

`Siplib.jl` also provides functions to plot sparsity pattern in the coefficient matrix. The plots can be drawn in four ways.

– Coefficient matrix of extensive form
– First stage-only block (block A)
– Second stage-only block (block W)
– Complicating block (block T)

4.2.1 Get size information

To get the size information of an instance, excute the following function.

```
function getSize(model::JuMP.Model, InstanceName::String="")::Size
```

The function `getSize` takes `JuMP.Model` as an input argument and returns `Size`-type object defined as follows.

```
type Size
   InstanceName::String # instance name
   nCont1::Int # number of continuous variables in 1st stage
   nBin1::Int # number of binary variables in 1st stage
   nInt1::Int # number of integer variables in 1st stage
   nCont2::Int # number of continuous variables in 2nd stage
   nBin2::Int # number of binary variables in 2nd stage
   nInt2::Int # number of integer variables in 2nd stage
   nCont::Int # number of continuous variables in total
   nBin::Int # number of binary variables in total
   nInt::Int # number of integer variables in total
   nRow::Int # number of rows in coefficient matrix in extensive form
   nCol::Int # number of columns in coefficient matrix in extensive form
   nNz::Int # number of nonzero values in coefficient matrix in extensive form
   Size() = new()
end
```

4.2.2 Get sparsity information

To get the sparsity information of an instance, excute the following function.

```
function getSparsity(model::JuMP.Model, InstanceName::String="")::Sparsity
```

The function `getSparsity` takes `JuMP.Model` as an input argument and returns `Sparsity`-type object.

```
type Sparsity
   InstanceName::String # instance name
   nRow1::Int # number of rows in 1st stage-only block (block A)
   nCol1::Int # number of columns in 1st stage-only block (block A)
   nNz1::Int # number of nonzero values in 1st stage-only block (block A)
   sparsity1::Float64 # sparsity ([0,1] scale) of 1st stage-only block (block A)
   nRow2::Int # number of rows in 2nd stage-only block (block W)
   nCol2::Int # number of columns in 2nd stage-only block (block W)
   nNz2::Int # number of nonzero values in 2nd stage-only block (block W)
   sparsity2::Float64 # sparsity ([0,1] scale) of 2nd stage-only block (block W)
   nRowC::Int # number of rows in complicating block (block T)
   nColC::Int # number of columns in complicating block (block T)
   nNzC::Int # number of nonzero values in complicating block (block T)
   sparsityC::Float64 # sparsity ([0,1] scale) of complicating block (block T)
   nRow::Int # number of rows in total
   nCol::Int # number of columns in total
   nNz::Int # number of nonzero values in total
   sparsity::Float64 # sparsity ([0,1] scale) in total
   Sparsity() = new()
end
```

4.2.3 Plot sparsity patterns

To plot the sparsity patterns of coefficient matrices, we provide the following functions.

```
function plotConstrMatrix(model::JuMP.Model, INSTANCE::String="instance",
     DIR_NAME::String="$(dirname(@__FILE__))/../plot")

function plotFirstStageBlock(model::JuMP.Model, INSTANCE::String="
     instance_block_A", DIR_NAME::String="$(dirname(@__FILE__))/../plot")

function plotSecondStageBlock(model::JuMP.Model, INSTANCE::String="
     instance_block_W", DIR_NAME::String="$(dirname(@__FILE__))/../plot")

function plotComplicatingBlock(model::JuMP.Model, INSTANCE::String="
     instance_block_T", DIR_NAME::String="$(dirname(@__FILE__))/../plot")

function plotAllBlocks(model::JuMP.Model, INSTANCE::String="instance", DIR_NAME
     ::String="$(dirname(@__FILE__))/../plot")

function plotAll(model::JuMP.Model, INSTANCE::String="instance", DIR_NAME::
     String="$(dirname(@__FILE__))/../plot")
```

The function `plotConstrMatrix` takes `JuMP.Model`-type object and plots the constraint matrix of extensive form. For example, the following command lines plot Fig. 3b.

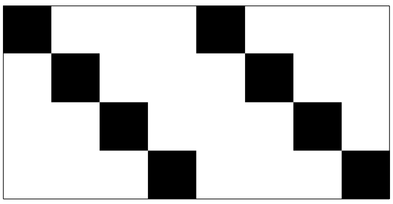
```
param_arr = [2,2,2,2] # declare parameters
problem = :DCAP # declare problem
INSTANCE = getInstanceName(problem, param_arr) # save instance name
model = getJuMPModel(problem, param_arr) # construct JuMP.Model object
plotConstrMatrix(model, INSTANCE) # plot extensive form constraint matrix
```

The functions `plotFirstStageBlock`, `plotSecondStageBlock`, and `plotComplicatingBlock` take `JuMP.Model`-type object and plots each block. For example, the following command lines plot Fig. 3a, 3c, and 3d.

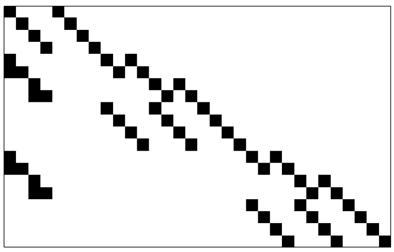
```
param_arr = [2,2,2,2] # declare parameters
problem = :DCAP # declare problem
INSTANCE = getInstanceName(problem, param_arr) # save instance name
model = getJuMPModel(problem, param_arr) # construct JuMP.Model object
plotFirstStageBlock(model, INSTANCE) # plot 1st stage block
plotSecondStageBlock(model, INSTANCE) # plot 2nd stage block
plotComplicatingBlock(model, INSTANCE) # plot complicating block
```

One might want to draw all the plots at once. The following two functions are defined to do that.

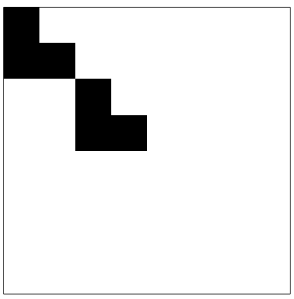
```
plotAllBlocks(model, INSTANCE) # plot all blocks A,W,T
plotAll(model, INSTANCE) # plot all the plots above
```



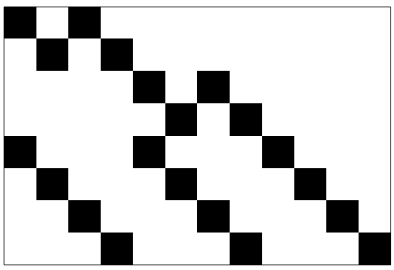
(a) DCAP_2_2_2_2_block_A.pdf



(b) DCAP_2_2_2_2.pdf



(c) DCAP_2_2_2_2_block_T.pdf



(d) DCAP_2_2_2_2_block_W.pdf

Fig. 3: Plots drawn by executing function `plotAll`

By executing `plotAll`, one can obtain all the plots in Fig. 3.

5 The instance catalog: Computational benchmarks for accompanied **SMPS** files

In this subsection, we give an overview of all ready-made SIPLIB 2.0 instances. For each instance, we

Table 7: Size report on the instances

		1st stage variables			2nd stage variables			Total							File size		
Problem	Instance	#cont1	#bin1	#int1	#cont2	#bin2	#int2	#cont	#bin	#int	#rows	#cols	#nonzeros	%density	.cor	.tim	.sto
DCAP	DCAP_2_3_3_500	6	6	0	9	18	0	4506	9006	0	7506	13512	28512	0.0281			
	DCAP_2_3_3_1000	6	6	0	9	18	0	9006	18006	0	15006	27012	57012	0.0141			
	DCAP_2_3_3_5000	6	6	0	9	18	0	45006	90006	0	75006	135012	285012	0.0028			
	DCAP_2_3_3_10000	6	6	0	9	18	0	90006	180006	0	150006	270012	570012	0.0014			
	DCAP_2_4_3_500	6	6	0	12	24	0	6006	12006	0	9006	18012	36012	0.0222			
	DCAP_2_4_3_1000	6	6	0	12	24	0	12006	24006	0	18006	36012	72012	0.0111			
	DCAP_2_4_3_5000	6	6	0	12	24	0	60006	120006	0	90006	180012	360012	0.0022			
	DCAP_2_4_3_10000	6	6	0	12	24	0	120006	240006	0	180006	360012	720012	0.0011			
	DCAP_3_3_2_500	6	6	0	6	18	0	3006	9006	0	6006	12012	25512	0.0354			
	DCAP_3_3_2_1000	6	6	0	6	18	0	6006	18006	0	12006	24012	51012	0.0177			
	DCAP_3_3_2_5000	6	6	0	6	18	0	30006	90006	0	60006	120012	255012	0.0035			
	DCAP_3_3_2_10000	6	6	0	6	18	0	60006	180006	0	120006	240012	510012	0.0018			
	DCAP_3_4_2_500	6	6	0	8	24	0	4006	12006	0	7006	16012	32512	0.0290			
	DCAP_3_4_2_1000	6	6	0	8	24	0	8006	24006	0	14006	32012	65012	0.0145			
	DCAP_3_4_2_5000	6	6	0	8	24	0	40006	120006	0	70006	160012	325012	0.0029			
	DCAP_3_4_2_10000	6	6	0	8	24	0	80006	240006	0	140006	320012	650012	0.0015			
MPTSPs	MPTSPs_D0_50_100	2450	2450	0	0	7350	0	2450	737450	0	247550	739900	994504	0.0005			
	MPTSPs_D0_50_500	2450	2450	0	0	7350	0	2450	3677450	0	1227550	3679900	4914504	0.0001			
	MPTSPs_D0_50_1000	2450	2450	0	0	7350	0	2450	7352450	0	2452550	7354900	9814504	0.0001			
	MPTSPs_D1_50_100	2450	2450	0	0	7350	0	2450	737450	0	247550	739900	994504	0.0005			
	MPTSPs_D1_50_500	2450	2450	0	0	7350	0	2450	3677450	0	1227550	3679900	4914504	0.0001			
	MPTSPs_D1_50_1000	2450	2450	0	0	7350	0	2450	7352450	0	2452550	7354900	9814504	0.0001			
	MPTSPs_D2_50_100	2450	2450	0	0	7350	0	2450	737450	0	247550	739900	994504	0.0005			
	MPTSPs_D2_50_500	2450	2450	0	0	7350	0	2450	3677450	0	1227550	3679900	4914504	0.0001			
	MPTSPs_D2_50_1000	2450	2450	0	0	7350	0	2450	7352450	0	2452550	7354900	9814504	0.0001			
	MPTSPs_D3_50_100	2450	2450	0	0	7350	0	2450	737450	0	247550	739900	994504	0.0005			
	MPTSPs_D3_50_500	2450	2450	0	0	7350	0	2450	3677450	0	1227550	3679900	4914504	0.0001			
	MPTSPs_D3_50_1000	2450	2450	0	0	7350	0	2450	7352450	0	2452550	7354900	9814504	0.0001			
	MPTSPs_D0_100_100	9900	9900	0	0	29700	0	9900	2979900	0	1000100	2989800	4019004	0.0001			
	MPTSPs_D0_100_500	9900	9900	0	0	29700	0	9900	14859900	0	4960100	14869800	19859004	0			
	MPTSPs_D0_100_1000	9900	9900	0	0	29700	0	9900	29709900	0	9910100	29719800	39659004	0			
	MPTSPs_D1_100_100	9900	9900	0	0	29700	0	9900	2979900	0	1000100	2989800	4019004	0.0001			
	MPTSPs_D1_100_500	9900	9900	0	0	29700	0	9900	14859900	0	4960100	14869800	19859004	0			
	MPTSPs_D1_100_1000	9900	9900	0	0	29700	0	9900	29709900	0	9910100	29719800	39659004	0			
	MPTSPs_D2_100_100	9900	9900	0	0	29700	0	9900	2979900	0	1000100	2989800	4019004	0.0001			
	MPTSPs_D2_100_500	9900	9900	0	0	29700	0	9900	14859900	0	4960100	14869800	19859004	0			
	MPTSPs_D2_100_1000	9900	9900	0	0	29700	0	9900	29709900	0	9910100	29719800	39659004	0			
	MPTSPs_D3_100_100	9900	9900	0	0	29700	0	9900	2979900	0	1000100	2989800	4019004	0.0001			
	MPTSPs_D3_100_500	9900	9900	0	0	29700	0	9900	14859900	0	4960100	14869800	19859004	0			
	MPTSPs_D3_100_1000	9900	9900	0	0	29700	0	9900	29709900	0	9910100	29719800	39659004	0			
SIZES	SIZES_100	0	20	20	0	0	110	0	20	11020	4022	11040	36060	0.0812			
	SIZES_500	0	20	20	0	0	110	0	20	55020	20022	55040	180060	0.0163			
	SIZES_1000	0	20	20	0	0	110	0	20	110020	40022	110040	360060	0.0082			
	SIZES_2000	0	20	20	0	0	110	0	20	220020	80022	220040	720060	0.0041			
	SIZES_4000	0	20	20	0	0	110	0	20	440020	160022	440040	1440060	0.0020			
SMKP	SMKP_120_20	0	240	0	0	120	0	0	2640	0	150	2640	36000	9.0909			
	SMKP_120_100	0	240	0	0	120	0	0	12240	0	550	12240	132000	1.9608			
	SMKP_120_200	0	240	0	0	120	0	0	24240	0	1050	24240	252000	0.9901			
	SMKP_120_400	0	240	0	0	120	0	0	48240	0	2050	48240	492000	0.4975			
	SMKP_120_800	0	240	0	0	120	0	0	96240	0	4050	96240	972000	0.2494			
SSLP	SSLP_5_25_100	0	5	0	5	125	0	500	12505	0	3001	13005	25305	0.0648			
	SSLP_5_25_500	0	5	0	5	125	0	2500	62505	0	15001	65005	126505	0.0130			
	SSLP_5_25_1000	0	5	0	5	125	0	5000	125005	0	30001	130005	253005	0.0065			
	SSLP_5_25_2000	0	5	0	5	125	0	10000	250005	0	60001	260005	506005	0.0032			
	SSLP_5_25_4000	0	5	0	5	125	0	20000	500005	0	120001	520005	1012005	0.0016			
	SSLP_5_25_8000	0	5	0	5	125	0	40000	1000005	0	240001	1040005	2024005	0.0008			
	SSLP_5_50_100	0	5	0	5	250	0	500	25005	0	5501	25505	50005	0.0356			
	SSLP_5_50_500	0	5	0	5	250	0	2500	125005	0	27501	127505	250005	0.0071			
	SSLP_5_50_1000	0	5	0	5	250	0	5000	250005	0	55001	255005	500005	0.0036			
	SSLP_5_50_2000	0	5	0	5	250	0	10000	500005	0	110001	510005	1000005	0.0018			
	SSLP_5_50_4000	0	5	0	5	250	0	20000	1000005	0	220001	1020005	2000005	0.0009			
	SSLP_5_50_8000	0	5	0	5	250	0	40000	2000005	0	440001	2040005	4000005	0.0004			
	SSLP_10_50_100	0	10	0	10	500	0	1000	50010	0	6001	51010	100110	0.0327			
	SSLP_10_50_500	0	10	0	10	500	0	5000	250010	0	30001	255010	500510	0.0065			
	SSLP_10_50_1000	0	10	0	10	500	0	10000	500010	0	60001	510010	1001010	0.0033			
	SSLP_10_50_2000	0	10	0	10	500	0	20000	1000010	0	120001	1020010	2002010	0.0016			
	SSLP_10_50_4000	0	10	0	10	500	0	40000	2000010	0	240001	2040010	4004010	0.0008			
	SSLP_10_50_8000	0	10	0	10	500	0	80000	4000010	0	480001	4080010	8008010	0.0004			
	SSLP_15_45_100	0	15	0	15	675	0	1500	67515	0	6001	69015	135915	0.0328			
	SSLP_15_45_500	0	15	0	15	675	0	7500	337515	0	30001	345015	679515	0.0066			
	SSLP_15_45_1000	0	15	0	15	675	0	15000	675015	0	60001	690015	1359015	0.0033			
	SSLP_15_45_2000	0	15	0	15	675	0	30000	1350015	0	120001	1380015	2718015	0.0016			
	SSLP_15_45_4000	0	15	0	15	675	0	60000	2700015	0	240001	2760015	5436015	0.0008			
	SSLP_15_45_8000	0	15	0	15	675	0	120000	5400015	0	480001	5520015	10872015	0.0004			
SUCW	SUCW_FallWD_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_FallWD_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_FallWD_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_FallWE_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_FallWE_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_FallWE_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_SpringWD_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_SpringWD_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_SpringWD_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_SpringWE_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_SpringWE_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_SpringWE_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_SummerWD_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_SummerWD_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_SummerWD_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_SummerWE_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_SummerWE_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_SummerWE_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_WinterWD_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_WinterWD_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_WinterWD_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			
	SUCW_WinterWE_10	960	1000	0	21274	2250	0	213700	23500	0	330408	237200	1030146	0.0013			
	SUCW_WinterWE_50	960	1000	0	21274	2250	0	1064660	113500	0	1643208	1178160	5091706	0.0003			
	SUCW_WinterWE_100	960	1000	0	21274	2250	0	2128360	226000	0	3284208	2354360	10168656	0.0001			

6 Concluding remarks

Any further contribution or suggestions for SIPLIB 2.0 are always welcomed. Better solutions than discovered so far, more functions, more problems with `Julia` scripts for instance generation, more effective classification rules, etc.

Appendix

A Problem descriptions

In this section, we explain details for each problem in `SIPLIB 2.0`. We also explain data generation procedures. Due to limited access to the original data in reference papers, we selectively choose the methods from several available references and modify some of them without harming validity. Also, we guess some parameters about scenario generation to connect the missing links.

A.1 DCAP: Dynamic capacity planning with stochastic demand

DCAP is the problem of determining a capacity expansion schedule for a set of resources, and the assignment of resource capacity to task with stochastic requirement over a multi-period planning horizon. We refer to the main reference [4].

A.1.1 DCAP: Mathematical formulation

We consider the problem of deciding the capacity expansion schedule for $|R|$ resources over $|T|$ time periods to satisfy the processing requirements of $|N|$ tasks where R, T, and N denote set of resources, set of time periods, and set of tasks, respectively. We define decision variables: the first-stage continuous variable x_{it} for the capacity acquisition of resource i in period t and the second-stage binary variable y^s_{ijt} to indicate whether resource i is assigned to task j in period t under scenario s. Additional first-stage binary variable u_{it} is for logical constraint whether or not we decided to acquire more capacity of resource i in period t. Hence, for all resource $i \in R$ and time $t \in T$, $u_{it} = 1$ if $x_{it} > 0$, $u_{it} = 0$ otherwise.

Under the definition of the decision variables, the extensive form of DCAP is written below and the summarized notation is available in Table 8.

$$
\begin{aligned}
(\mathsf{DCAP})\ \min\ & \sum_{t\in T}\sum_{i\in R}(\alpha_{it}x_{it}+\beta_{it}u_{it}) + \sum_{s\in\mathcal{S}}\mathbb{P}(s)\sum_{t\in T}\sum_{i\in R\cup\{0\}}\sum_{j\in N} c^s_{ijt}y^s_{ijt} && \text{(6a)}\\
\text{s.t. } & x_{it} \le \mathrm{M}u_{it}, \quad \forall i\in R,\ \forall t\in T, && \text{(6b)}\\
& \sum_{j\in N} d^s_{jt}y^s_{ijt} \le \sum_{\tau=1}^{t} x_{i\tau}, \quad \forall i\in R,\ \forall t\in T,\ \forall s\in\mathcal{S}, && \text{(6c)}\\
& \sum_{i\in R\cup\{0\}} y^s_{ijt} = 1, \quad \forall j\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, && \text{(6d)}\\
& x_{it} \ge 0, \quad \forall i\in R,\ \forall t\in T, && \text{(6e)}\\
& u_{it}\in\{0,1\}, \quad \forall i\in R,\ \forall t\in T, && \text{(6f)}\\
& y^s_{ijt}\in\{0,1\}, \quad \forall i\in R\cup\{0\},\ \forall j\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, && \text{(6g)}
\end{aligned}
$$

The objective function (6a) is to minimize total expected cost for the capacity expansion schedule. The first double summation denotes the expansion cost for resource i in period t where α_{it} and β_{it} are the variable and fixed cost, respectively. The second term in the objective function represents the expected assignment cost in period t over all scenario $s \in \mathcal{S}$. Note that a dummy resource $i = 0$ is included with infinite capacity. The cost c^s_{0jt} denotes the penalty of failing to assign a resource to task j. The dummy resource enforces the *complete recourse property*, which ensures that there is a feasible second-stage assignment in all periods and all scenarios for any capacity acquisition schedule [4]. Constraint (6b) is the logical constraint containing a suitably large value M (we set M=1 in SIPLIB 2.0 to follow the original implementation in SIPLIB although it does not seem to be large enough) to define the cost for capacity expansion. Constraint (6c) reflects that the processing requirement of all tasks assigned to a resource in any period cannot exceed the installed capacity in that period under all scenarios. Constraint (6d) guarantees that each task needs to be assigned to exactly one resource in each period under all scenarios. Finally, constraints (6e)-(6g) restrict the space from which the variables take values.

Table 8: Notations for DCAP

Index sets:	
R	index set of resources ($i \in R \cup \{0\}$ where 0 is a dummy resource with infinite capacity)
N	index set of tasks ($j \in N$)
T	index set of time periods ($t \in T$)
$\mathcal{S}$	index set of scenarios ($s \in \mathcal{S}$)
Parameters:	
α_{it}	variable cost for expanding capacity of resource i
β_{it}	fixed cost for expanding capacity of resource i
c^s_{ijt}	cost of processing task j using resource i in period t under scenario s
d^s_{jt}	processing requirement for task j in period t under scenario s
$\mathbb{P}(s)$	the probability of occurence of scenario s
Decision variables:	
x_{it} (1^{st} stage)	capacity acquisition amount of resource i in period t
u_{it} (1^{st} stage)	1 if capacity of resource i is expanded in period t, 0 otherwise
y^s_{ijt} (2^{nd} stage)	1 if resource i is assigned to task j in period t under scenario s, 0 otherwise

A.1.2 DCAP: Data generation

There are four factors that define the instance of DCAP $|R|$, $|N|$, $|T|$, and $|\mathcal{S}|$. Once we decide the factors, the instance is named by DCAP_$|R|$_$|N|$_$|T|$_$|\mathcal{S}|$. Let U be a continuous uniform random variable: $U \sim Unif(0,1)$. Then, the parameters are generated as follows:

$$\begin{aligned}
\alpha_{it} &= 5U + 5, \quad \forall i \in R,\ \forall t \in T,\\
\beta_{it} &= 40U + 10, \quad \forall i \in R,\ \forall t \in T,\\
c^s_{ijt} &= 5U + 5, \quad \forall i \in R,\ \forall j \in N,\ \forall t \in T,\ \forall s \in \mathcal{S},\\
c^s_{0jt} &= 500U + 500, \quad \forall j \in N,\ \forall t \in T,\ \forall s \in \mathcal{S},\\
d^s_{jt} &= U + 0.5, \quad \forall j \in N,\ \forall t \in T,\ \forall s \in \mathcal{S}.
\end{aligned}$$

A.2 MPTSPs: Mutli-path traveling salesman problem with stochastic travel times

MPTSPs is a variant of the travelling salesman problem (TSP) where a set of paths exists between any two nodes and each path is characterized by a random travel time.

In SIPLIB, only limited data (e.g., number of nodes, coordinates of nodes, generated travel times) are provided and no SMPS file is available. We mainly refer to [12] for deriving the mathematical formulation. Due to the malfunction of subtour breaking constraints in the reference model, we refer to another paper [14] to contain working subtour-breaking constraint.

A.2.1 MPTSPs: Mathematical formulation

We consider a two-stage SIP with recourse. The travel time oscillation e^k_{ij} by using path k between nodes i and j. We present each realization (scenario) of random travel time oscillation by e^s_{ijk} where s indicates the scenario. In MPTSPs at the first stage, the decision-maker does not have any information about the travel time oscillation. The tour paths among the nodes, however, should be determined before the complete information is available. The first stage decision variable y_{ij} is represented by the selection of nodes i and j to be visited in a tour. In the second stage where the random travel time c^s_{ijk} are available, the paths k between each couple of nodes i and j under scenario s, x^s_{ijk} can be calculated.

Let N and K_{ij}, respectively, be the finite set of nodes of the graph and the set of paths between the pair of nodes $i, j \in N$. We denote with $\mathcal{S}$ the set of scenarios with associated equally distributed probability of each scenario $\mathbb{P}(s)$, i.e., $\mathbb{P}(s) \equiv 1/|\mathcal{S}|$. Each path $k \in K_{ij}$ between nodes $i, j \in N$ is characterized by a non-negative estimation of the mean unit travel time $\bar{c}_{ij}$ and a non-negative unit random travel time c^s_{ijk} under the scenario $s \in S$. Let $e^s_{ijk} \equiv c^s_{ijk} - \bar{c}_{ij}$ be the error on the travel time estimated for the path $k \in K_{ij}$ under time scenario $s \in S$.

The first stage binary variables $y_{ij} = 1$ if node $j \in N$ is visited right after node $i \in N$, 0 otherwise. The second stage binary variables $x^s_{ijk} = 1$ if path $k \in K_{ij}$ between nodes $i, j \in N$ is selected at the second stage, 0 otherwise. We have one more set of first stage variables ϕ_{ij} which is introduced to break the subtours [14]. The non-negative continuous variables ϕ_{ij} describe the flow of a single commodity to node 1 from every other nodes (without loss of generality, 1 is the starting node).

The extensive form of MPTSPs is as follows and the notations used are summarized in Table 9.

$$(\mathsf{MPTSPs}) \min \sum_{i\in N}\sum_{j\in N\setminus\{i\}} \bar{c}_{ij} y_{ij} + \sum_{s\in\mathcal{S}} \mathbb{P}(s) \sum_{i\in N}\sum_{j\in N\setminus\{i\}}\sum_{k\in K_{ij}} e^s_{ijk} x^s_{ijk} \tag{7a}$$

$$\text{s.t.} \sum_{j\in N\setminus\{i\}} y_{ij} = 1, \quad \forall i \in N, \tag{7b}$$

$$\sum_{i\in N\setminus\{j\}} y_{ij} = 1, \quad \forall j \in N, \tag{7c}$$

$$\sum_{j\in N} \phi_{lj} - \sum_{i\in N\setminus\{1\}} \phi_{il} = 1, \quad \forall l \in N\setminus\{1\}, \tag{7d}$$

$$\phi_{ij} \le (|N| - 1)\, y_{ij}, \quad \forall i \in N\setminus\{1\},\ \forall j \in N\setminus\{i\}, \tag{7e}$$

$$\sum_{k\in K_{ij}} x^s_{ijk} = y_{ij}, \quad \forall i \in N,\ \forall j \in N\setminus\{i\},\ \forall s \in \mathcal{S}, \tag{7f}$$

$$y_{ij} \in \{0,1\}, \quad \forall i \in N,\ \forall j \in N\setminus\{i\}, \tag{7g}$$

$$\phi_{ij} \ge 0, \quad \forall i \in N\setminus\{1\},\ \forall j \in N. \tag{7h}$$

$$x^s_{ijk} \in \{0,1\}, \quad \forall i \in N,\ \forall j \in N\setminus\{i\},\ \forall k \in K_{ij},\ \forall s \in \mathcal{S}, \tag{7i}$$

The first sum in the objective function (7a) represents the first stage travel cost, while the second sum represents the recourse action, consisting in choosing the best path $k \in K_{ij}$ under scenario $s \in \mathcal{S}$. The constraints (7b) and (7c) form the assignment constraints and ensure that each node is visited only once. Given the fixed values of y_{ij}, constraint (7d) and (7e) form a network flow problem, and therefore the ϕ_{ij} values will be integer. In case the solutions of the above formulation contain at least one subtour, the constraints (7d) and (7e) are violated. Moreover, no tour can exist that does not contain node 1 by the two constraints. For more explanation on the subtour breaking mechanism accompanied with rigorous proof, refer to [16]. The constraint (7f) guarantees that path k between nodes i and j can be chosen at stage 2 only if nodes i and j were part of the tour fixed at stage 1. Finally, the constraints (7g)-(7i) restrict the space from which the variables take values.

Table 9: Notations for MPTSPs

Index sets:	
N	index set of nodes ($i, j, l \in N$)
K_{ij}	index set of paths between nodes i and j ($k \in K_{ij}$)
$\mathcal{S}$	index set of scenarios ($s \in \mathcal{S}$)
Parameters:	
c^s_{ijk}	unit random travel time of path k between nodes i, j under scenario s
$\bar{c}_{ij}$	estimation of the mean unit travel time (expectation of c^s_{ijk} over all s and k)
e^s_{ijk}	the error on the travel time on estimated for arc (i, j) and path k under scenario s
$\mathbb{P}(s)$	the probability of occurence of scenario s
Decision variables:	
ϕ_{ij} (1^{st} stage)	the nonnegative real-valued flow on arc (i, j)
y_{ij} (1^{st} stage)	1 if path k between nodes $i, j \in N$ is selected at the second stage, 0 otherwise
x^s_{ijk} (2^{nd} stage)	1 if node j is visited just after node i, 0 otherwise

A.2.2 MPTSPs: Data generation

We follow the scenario generation methods described through the references [11, 12, 13]. For MPTSPs there are three mainly distinguished characteristics for each instance: the nodes partition strategy ($D \in \{D0, D1, D2, D3\}$, explanation on each strategy is forthcoming), the number of nodes ($|N| \in \{2, 3, \ldots\}$), and the number of scenarios ($|\mathcal{S}| \in \{1, 2, \ldots\}$). Another important charicteristic $|K_{ij}| \in \{1, 2, 3, \ldots\}$ is the number of paths for each edge which is fixed by 3 as a default following [13]. Once we decide D, $|N|$, and $|S|$ by, each instance is named by MPTSPs_D_$|N|$_$|\mathcal{S}|$.

The nodes are distributed in a circle with radius equal to r km. We use Cartesian coordinate system where the geometric center of the circle is (r, r). The nodes are distinguished by two subsets: *central* and *suburban*. If the Euclidean distance between a node and the geometric center is less than or equal to the half of the radius ($r/2$), then the node is of *central* type. Otherwise, if the Euclidean distance is greater than the half of the radius, the node is of *suburban* type. Each arc between any two nodes i and j is either *homogeneous* or *heterogeneous*. If the two nodes are of the same type of node, i.e., both are *central* or both are *suburban*, the type of the arc is *homogeneous*. Otherwise, the type of the arc is *heterogeneous*. Later, the travel time of each path between two nodes are affected by the type of arc.

The nodes are generated by one of the following distribution strategies:

– $D0$: All the nodes are *central*.
– $D1$: All the nodes are *suburban*.
– $D2$: 3/4 of the nodes are *central* and the remaining 1/4 are *suburban*.
– $D3$: 1/2 of the nodes are *central* and the remaining 1/2 are *suburban*.

Given $D, |N|$ and $|\mathcal{S}|$, the next procedure can be summarized as follows:

1. Generate $|N|$ nodes based on the predetermined strategy D. Then, the nodes are generated by acceptance-rejection procedure with uniform random number generation. Again following [13], we fix $r = 7km$.
2. Calculate Euclidean distances between the nodes (EC_{ij}).
3. We guess and fix the deterministic velocity profile by 40*km/h* for the *central* nodes and 80*km/h* for the *suburban* nodes: $v_{cntr} = 40$ and $v_{sbrb} = 80$.
4. Generate random travel times (c^s_{ijk}) for each scenario s.
   – The velocity for traveling arc (i, j) is affected by its arc type.
   – If the arc is *homogeneous*, the random travel time of all the paths are generated only based on the corresponding velocity profile.
   – If the arc is *heterogeneous*, $\left\lceil \frac{|K_{ij}|}{3} \right\rceil$ paths are generated based on $v_{cntr} = 40$ and the remaining paths are generated based on $v_{sbrb} = 80$.
   – The velocities are distributed by $Unif(\frac{v}{2}, 2v)$ for $v = v_{cntr}, v_{sbrb}$.

– In summary, if the arc (i,j) is *homogeneous*,

$$c^s_{ijk} \sim \begin{cases} \frac{EC_{ij}}{Unif(\frac{v_{cntr}}{2}, 2v_{cntr})} & \text{if } i,j \text{ are both } \textit{central}, \\ \frac{EC_{ij}}{Unif(\frac{v_{sbrb}}{2}, 2v_{sbrb})} & \text{if } i,j \text{ are both } \textit{suburban}, \end{cases} \quad \forall k \in K_{ij}.$$

– Otherwise, if (i,j) is *heterogeneous*,

$$c^s_{ijk} \sim \begin{cases} \frac{EC_{ij}}{Unif(\frac{v_{cntr}}{2}, 2v_{cntr})} & \text{for } k \in \left\{1,\ldots,\left\lceil \frac{|K_{ij}|}{3} \right\rceil\right\}, \\ \frac{EC_{ij}}{Unif(\frac{v_{sbrb}}{2}, 2v_{sbrb})} & \text{for } k \in \left\{\left\lceil \frac{|K_{ij}|}{3} \right\rceil + 1,\ldots,|K_{ij}|\right\}. \end{cases}$$

5. Finally, we multiply 3600 for each component of c^s_{ijk} to convert the unit from *hours* to *seconds*.

A.3 SIZES: Selection of an optimal subset of sizes

SIZES is a simplified version of the cutting-stock problem with multi-period stochastic demand. We only consider the two-periods model to follow [15]. The first period demand is deterministic and the demand for the second period is stochastic. We refer to the mathematical formulation in [15] to construct `JuMP.Model`. Due to some unclear explanations (or typo), we slightly modify the formulation and use it for SIPLIB 2.0.

A.3.1 SIZES: Mathematical formulation

Suppose a product is available in a finite number $|N|$ of sizes where 1 is the index of the smallest size and $|N|$ is the index of the largest size. Further, suppose size i is substitutable for size j if $i > j$, i.e., larger-sized items may fulfill demand for smaller sizes. Unlike typical cutting-stock problem, an item cannot be substituted into several pieces. Let p_i be the unit production cost for size i. Generally $p_i > p_j$ for $i > j$. Let f be the fixed setup cost for producing units of any size and r be the unit penalty cost of meeting demand for size j with a larger size i. Let d^s_{jt} be the stochastic demand for size j at time t under scenario l. Let c^l_t be the stochastic production capacity at time t under scenario s. $\mathbb{P}(s)$ is the equiprobable probability of occurence for scenario s. We introduce three decision variables. The first-stage integer variable y_{it} is the number of units of sizes i produced at time t. Another first-stage variable z_it is a binary variable that denotes whether or not we produce size i item at time t under scenario l. The second-stage integer variable x^s_{ijt} denotes the number of units of size i cut to meet demand for smaller size j at time t under scenario s. For x^s_{ijt} with $i = j$, we use it to indicate that items of length index i are to be used without cutting at time t under scenario s. Based on the above definitions, SIZES can be formulated by the following extensive form.

$$(\mathsf{SIZES})\ \min \sum_{t\in T}\sum_{i\in N}(fz_{it}+p_iy_{it})+\sum_{s\in\mathcal{S}}\mathbb{P}(s)\sum_{t\in T}\sum_{i\in N\setminus\{1\}}\sum_{j=1}^{i-1}rx_{ijt}^s \tag{8a}$$

$$\text{s.t.} \sum_{i\in N} y_{it}\le c_t, \quad \forall t\in T, \tag{8b}$$

$$y_{it}\le c_t z_{it}, \quad \forall i\in N,\ \forall t\in T, \tag{8c}$$

$$\sum_{t'=1}^{t}\sum_{i=j}^{|N|} x_{ijt'}^s \ge d_{jt}^s, \quad \forall j\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, \tag{8d}$$

$$\sum_{t'=1}^{t}\sum_{j=1}^{i} x_{ijt'}^s \le \sum_{t'=1}^{t} y_{it'}, \quad \forall i\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, \tag{8e}$$

$$y_{it}\in\mathbb{Z}_+, \quad \forall j\in N,\ \forall t\in T, \tag{8f}$$

$$z_{it}\in\{0,1\}, \quad \forall i\in N,\ \forall t\in T, \tag{8g}$$

$$x_{ijt}^s\in\mathbb{Z}_+, \quad \forall(i,j:i\ge j)\in N\times N,\ \forall t\in T,\ \forall s\in\mathcal{S}. \tag{8h}$$

The first sum of the objective function (8a) is the costs for producing items for all time periods (fixed + variable costs). The second term corresponds to the expectation of the penalty costs for substituting items. Constraint (8b) ensures the production for each period cannot exceed the capacity under all scenarios. Constraint (8c) is the logical constraint for the cost expression. (8d) guarantees the demand for each item can be met for all time periods and for all scenarios. Notice that constraint (8d) means the demand can be met by the items that are produced in the previous periods as well. Constraint (8e) enforces the supply limit. Constraints (8f)-(8h) are binary or integer restrictions of the decision variables.

Table 10: Notations for SIZES

Index sets	
N	index set of items ($i,j\in N$)
T	index set of time periods ($t\in T$)
$\mathcal{S}$	index set of scenarios ($s\in\mathcal{S}$)
Parameters	
p_i	unit production cost for item i
f	fixed setup cost for producing any item
r	unit cutting cost
c_t	production capacity at time t
d_{it}^s	demand for item i at time t under scenario s
$\mathbb{P}(s)$	the probability of occurence of scenario s
Decision variables	
y_{it} (1st stage)	number of units of size i produced at time t
z_{it} (1st stage)	1 if we produce size i at time t, 0 otherwise
x_{ijt}^s (2nd stage)	number of units of size i cut to meet demand for smaller size j at time t under scenario s

A.3.2 SIZES: Data generation

Instances of SIZES are generated based on the one-period data given in Table 11. Note that although the table includes sleeve length data, we do not use this information for SIZES since this is not a typical cutting stock problem. Following [15], we set the stochastic parameter $c_t^s = 200,000$ to be deterministic for all $t\in T$ and $s\in\mathcal{S}$, hence only the demand parameter (d_i^s) is stochastic throughout the scenarios. The stochastic demand data is generated based on Table 11. First, we decide the number of scenarios to be generated by $|\mathcal{S}|$. Then, the demand data is specified by a vector of multipliers: one multiplier for each scenario that is

Table 11: Base data for SIZES scenarios [15]

i	sleeve length	unit production cost (p_i)	demand (d_i)
1	25	0.748	2500
2	30	0.7584	7500
3	35	0.7688	12500
4	40	0.7792	10000
5	45	0.7896	35000
6	50	0.8	25000
7	55	0.8014	15000
8	60	0.8208	12500
9	65	0.8312	12500
10	70	0.8416	5000
unit cutting cost (u): \$0.008			
setup cost (f): \$453			
production capacity (c_t): 200,000			

multiplied times the demand vector from Table 11. For example, if $|\mathcal{S}| = 3$, the instance is defined by $(0.7, 1, 1.3)$. Or if $|\mathcal{S}| = 5$, the instance is defined by $(0.6, 0.8, 1, 1.2, 1.4)$. Since SIPLIB provides instances with $|\mathcal{S}| \leq 20$, we recommend the users to use SIPLIB 2.0 to only generate instances with $|\mathcal{S}| \geq 20$. In SIPLIB 2.0 the multiplier vector is defined by the equally split set of subintervals between $[0.5, 1.5]$, e.g., when $|\mathcal{S}| = 20$, the multiplier vector is $(0.5, 0.55, 0.6, \ldots, 1.4, 1.45, 1.5)$. With larger value of $|\mathcal{S}|$, we will have vector with finer granularity. To generate more random instances, the demand vector is multiplied by a continuous random number U that is uniformly distributed in $(0.5, 1.5)$. After that, the instance is named by SIZES_$|\mathcal{S}|$.

A.4 SMKP: Stochastic multiple knapsack problem

SMKP is a class of stochastic multiple binary knapsack problems. Unlike typical knapsack problems where the objective is to maximize total profits under the restriction of the weight capacity of each knapsack, SMKP is to minimize total weights while satisfying a certain required profit for each knapsack.

SIPLIB provides 30 instances of SMKP in total. The first-stage problems contain 240 binary variables and 50 knapsack constraints. The second-stage problems have 120 binary variables and 5 knapsack constraints. Each instance has 20 scenarios. We mainly refer to [17].

A.4.1 SMKP: Mathematical formulation

We have three types of items x, z, and y where the first two types are of the first-stage and the last one is of the second-stage with stochastic scenarios. For each type, we have $|I|$ number of items where I is the index set of the items. Hence, we define the binary variables x_i, z_i, and y_i^s which are equal to 1 if the i^{th} item is decided to be included (s denotes scenario so only appears in y-type variables). We consider two types of knapsacks: one associated with x-type and z-type items (say xz-knapsack) and the other one with x-type and y-type items (say xy-knapsack). xz-knapsacks are indexed by $j \in J$ and xy-knapsacks are indexed by $k \in K$. Each knapsack has its own minimum level of profit that should be satisfied by the items of the associated types, e.g., the profit of the j^{th} xz-type knapsack is calculated based on the inclusion or exclusion of x-type and z-type items and should satisfy a certain requirement b_j. Bear in mind that the inclusion or exclusion of a certain item i affects all the associated knapsacks.

Each parameter c_i, d_i, and q_i^s denotes the gain of weight when including items of type x, z, and y, respectively. Here, c_i and d_i are deterministic and q_i^s is stochastic. Parameters

a_{ji}, e_{ji}, t_{ki}, and w_{ki} are all deterministic and denote the profits for including items in the knapsacks. The RHS parameters b_j and h_k are the minimum levels of profit requirements for xz-knapsacks and xy-knapsacks, respectively.

The extensive form of SMKP is as follows and the notations used are summarized in Table 12.

$$(\mathsf{SMKP})\ \min \sum_{i \in I} (c_i x_i + d_i z_i) + \sum_{s \in \mathcal{S}} \mathbb{P}(s) \sum_{i \in I} q_i^s y_i^s \tag{9a}$$

$$\text{s.t.} \sum_{i \in I} a_{ji} x_i + \sum_{i \in I} e_{ji} z_i \geq b_j, \quad \forall j \in J, \tag{9b}$$

$$\sum_{i \in I} t_{ki} x_i + \sum_{i \in I} w_{ki} y_i^s \geq h_k, \quad \forall k \in K,\ \forall s \in \mathcal{S}, \tag{9c}$$

$$x_i \in \{0,1\}, \quad \forall i \in I, \tag{9d}$$

$$z_i \in \{0,1\}, \quad \forall i \in I, \tag{9e}$$

$$y_i^s \in \{0,1\}, \quad \forall i \in I,\ \forall s \in \mathcal{S}. \tag{9f}$$

The objective (9a) is to minimize the expected value of the total weights. Constraint (9b) ensures the minimum levels of profit requirements for all xz-knapsacks are satisfied. Constraint (9c) guarantees the minimum levels of profit requirements are satisfied for all xy-knapsacks under every scenario. Constraints (9d)-(9f) are binary restriction of the decision variables.

Table 12: Notations for SMKP

Index sets:	
I	index set of items for each type ($i \in I$)
J	index set of xz-knapsacks ($j \in J$)
K	index set of xy-knapsacks ($k \in K$)
$\mathcal{S}$	index set of scenarios ($s \in \mathcal{S}$)
Parameters:	
c_i	weight of the i^{th} x-type item
d_i	weight of the i^{th} z-type item
q_i^s	weight of the i^{th} y-type item under scenario s
a_{ji}	profit of the j^{th} xz-knapsack for including i^{th} x-type item
e_{ji}	profit of the j^{th} xz-knapsack for including i^{th} z-type item
t_{ki}	profit of the k^{th} xy-knapsack for including i^{th} x-type item
w_{ki}	profit of the k^{th} xy-knapsack for including i^{th} y-type item
b_j	minimum required profit for the j^{th} xz-knapsack
h_k	minimum required profit for the k^{th} xy-knapsack
$\mathbb{P}(s)$	the probability of occurence of scenario s
Decision variables:	
x_i (1^{st} stage)	1 if the i^{th} x-type item is decided to be included, 0 otherwise
z_i (1^{st} stage)	1 if the i^{th} z-type item is decided to be included, 0 otherwise
y_i^s (2^{nd} stage)	1 if the i^{th} y-type item is decided to be included under scenario s, 0 otherwise

A.4.2 SMKP: Data generation

There are two factors that define the instance of SMKP: $|I|$ and $|\mathcal{S}|$. The sizes for another sets are fixed by $|J| = 50$ and $|K| = 5$ following [17]. Once we decide the factors $|I|$, $|\mathcal{S}|$, $|T|$, and $|\mathcal{S}|$, each instance is named by SMKP_$|I|$_$|\mathcal{S}|$. Again directly following [17], we randomly generate the parameters. Let U be a discrete uniform random variable: $U \sim Unif[1, 100]$.

Then, the parameters are generated as follows:

$$
\begin{aligned}
c_i &= U, \quad \forall i \in I, \\
d_i &= U, \quad \forall i \in I, \\
q_i^s &= U, \quad \forall i \in I,\ \forall s \in \mathcal{S}, \\
a_{ji} &= U, \quad \forall j \in J,\ \forall i \in I, \\
e_{ji} &= U, \quad \forall j \in J,\ \forall i \in I, \\
t_{ki} &= U, \quad \forall k \in K,\ \forall i \in I, \\
w_{ki} &= U, \quad \forall k \in K,\ \forall i \in I, \\
b_j &= \frac{3}{4} \sum_{i \in I} \left(a_{ji} + e_{ji}\right), \quad \forall j \in J, \\
h_k &= \frac{3}{4} \sum_{i \in I} \left(t_{ji} + w_{ji}\right), \quad \forall k \in K.
\end{aligned}
$$

A.5 SSLP: Stochastic server location problem

SSLP is a class of problem that finds the optimal location of servers and the optimal allocation of clients to servers which maximizes the expected net income under uncertain presents of clients. SSLP finds applications in a variety of domains such as network design for electric power, internet services, telecommunications, and water distribution. SIPLIB provides 12 instances with varying number of clients, server locations, and scenarios in SMPS format. The largest instance includes 10 server locations, 50 clients, and 2,000 scenarios which corresponds to 120,001 constraints, 1,000,010 binary variables, and 20,000 continuous variables.

We refer to [20] for mathematical formulation and data generation forthcoming through the following subsections.

A.5.1 SSLP: Mathematical formulation

Let I, J, Z, and $\mathcal{S}$ be index sets for the clients, servers, zones, and scenarios. For $i \in I$, $j \in J$, $z \in Z$, and $s \in \mathcal{S}$, we define the notations in Table 13.

Suppose that we place a server at location j. Then, the allocation costs c_j and the server will provide capacity to serve up to u amount of resource to clients. The revenue earned by serving client i from location j is denoted by q_{ij}. We have also a shortage cost (penalty) q_{0j} for each unit of demand that remains unserved among the clients assigned to server j. If client i is served by a server at location j, it uses d_{ij} units of resource from the server. We allow only one server to be installed at each location and each client can only be served by one server. There is a requirement that a minimum number of servers to be located in a zone z, and is denoted by w_z.

The first-stage binary variables x_j decide whether or not a server is located at location j. The second-stage binary variables y_{ij}^s are referred to as recourse decision under scenario s and associated with the decision on serving client i by server j. The variables y_{ij}^s will be implemented in the future, when scenario s is finally observed.

Based on the above, the extensive form of SSLP can be stated as follows:

$$(\mathsf{SSLP})\ \min \sum_{j\in J} c_j x_j - \sum_{s\in\mathcal{S}} \mathbb{P}(s) \left(\sum_{i\in I}\sum_{j\in J} q^s_{ij} y^s_{ij} - \sum_{j\in J} q^s_{0j} y^s_{0j} \right) \tag{10a}$$

$$\text{s.t.} \sum_{j\in J} x_j \le v, \tag{10b}$$

$$\sum_{j\in J_z} x_j \ge w_z, \quad \forall z \in Z, \tag{10c}$$

$$\sum_{i\in I} d_{ij} y^s_{ij} - y^s_{0j} \le u x_j, \quad \forall j \in J,\ \forall s \in \mathcal{S}, \tag{10d}$$

$$\sum_{j\in J} y^s_{ij} = h^s_i, \quad \forall i \in I,\ \forall s \in \mathcal{S}, \tag{10e}$$

$$x_j \in \{0,1\}, \quad \forall j \in J, \tag{10f}$$

$$y^s_{ij} \in \{0,1\}, \quad \forall i \in I,\ j \in J,\ s \in \mathcal{S}, \tag{10g}$$

$$y^s_{0j} \ge 0, \quad \forall j \in J,\ \forall s \in \mathcal{S}. \tag{10h}$$

The objective function (10a) is to maximize total expected revenue of locating servers and serving customers by the servers. Constraint (10b) satisfies the requirement that only up to a total of v available servers can be installed. The zonal requirements that specify how many servers are needed in each zone are given by constraint (10c). Constraint (10d) ensures that a server located at site j can serve only up to its capacity u. The variable y^s_{0j} is introduced in the constraint (10d) to accomodate any overflows that are not served due to limitations in server capacity. These overflows result in a loss of revenue at a rate of q^s_{0j}. The inclusion of an artificial variable may allow a client to be assigned to servers that are not located. However, penalty costs associated with such an assignment may result in such high costs as to preclude it in an optimal solution, unless server capacity is so limited that some clients have to be turned away [20]. Constraint (10e) guarantees that each client is served by only one server. Constraint (10f) and (10g) are binary restrictions on the decision variables. Finally, constraint (10h) is the non-negativity requirement on the overflow variables.

Table 13: Notations for SSLP

Index sets:	
J	index set of server locations ($j \in J$)
I	index set of clients ($i \in I$)
Z	index set of zones ($z \in Z$)
$\mathcal{S}$	index set of scenarios ($s \in \mathcal{S}$)
Parameters:	
c_j	cost of locating a server at location j
q^s_{ij}	revenue from client i being served by server at location j under scenario s
q^s_{0j}	rate of revenue loss for overflows that are not served due to limited server capacity under scenario s
d_{ij}	resource demand of client i from server at location j
u	server capacity
v	upper bound on the total number of servers that can be located
w_z	minimum number of servers to be located in zone z
J_z	subset of server locations that belong to zone z
h^s_i	1 if client i is present under scenario s, 0 otherwise
$\mathbb{P}(s)$	probability of occurence for scenario s
Decision variables:	
x_j (1^{st} stage)	1 if a server is located at site j, 0 otherwise
y^s_{ij} (2^{nd} stage)	1 if client i is served by a server at location j under scenario s, 0 otherwise
y^s_{0j} (2^{nd} stage)	non-negative amount of overflows that are not served due to limitations in server j's capacity

A.5.2 SSLP: Data generation

For each instance of SSLP we determine the number of potential server locations $|J|$, the number of clients $|I|$, and the number of scenarios $|\mathcal{S}|$. Then, the instance is named by SSLP_$|J|$_$|I|$_$|\mathcal{S}|$. The client-server revenue are set to be 1 per unit of client demand. Some of deterministic parameters are randomly generated from the discrete uniform distribution while scenario data are generated from the Bernoulli distribution. In summary, the parameters are generated as follows:

$$\begin{aligned}
c_j &= Unif[40, 80], \quad \forall j \in J, \\
d_{ij} &= Unif[0, 25], \quad \forall i \in I, \ \forall j \in J, \\
h_i^s &= Bernoulli(0.5), \quad \forall i \in I, \ \forall s \in \mathcal{S}, \\
q_{ij}^s &= d_{ij}, \quad \forall i \in I, \ \forall j \in J, \ \forall s \in \mathcal{S}, \\
q_{0j}^s &= 1000, \quad \forall j \in J, \ \forall s \in \mathcal{S}, \\
v &= |J| \\
u &= \frac{3}{2} \times \frac{\sum_{i \in I} \sum_{j \in J} d_{ij}}{|J|}
\end{aligned}$$

Note that the zonal data is omitted due to the lack of available information. Hence, constraint (10c) does not appear in SIPLIB 2.0 instances.

A.6 SUC: Stochastic unit commitment problem

The unit commitment (UC) problem is a production cost model (PCM) that plans power system operations over an extended time horizon. SUC is a stochastic version of UC for studying the impact of incorporating highly uncertain power generation of large-scale wind turbines with transmission constraints and system component failures. We refer to [18] for mathematical models. For model parameters, we use Western Electricity Coordinating Council data set (WECC [27]) interconnected with California ISO Open Access Same-Time Information System (CAISO) as in the reference.

A.6.1 SUC: Mathematical formulation

In SUC, we make commitment decision on slow generators in the first stage. In the second stage, the commitment decisions of fast generators, wind generators, and non-wind renewable generators, shedding decisions of loads, and import decisions from other points are determined. In addition, phase angle for each transmission line is decided in the second stage. In SUC, we also consider ramping constraints, transmission line capacity constraints, phase angle constraints, and minimum up/down time constraints. We assume the piecewise linear convex cost function for the power generation.

Using the notation in Table 14, the extensive form of SUC can be stated as in the following model 11.

$$
\begin{aligned}
(\mathsf{SUC})\ \min\ & \sum_{g\in G_s}\sum_{t\in T}(K_g w_{gt}+S_g z_{gt}) \\
& +\sum_{s\in\mathcal{S}}\mathbb{P}(s)\sum_{t\in T}\Bigg[\sum_{g\in G_f}(K_g u_{gt}+S_g v_{gt})+\sum_{g\in G}C_g p_{gt}+\sum_{j\in J}C^J\lambda_{dt} \\
& \qquad +\sum_{i\in I}C^I\mu_{it}+\sum_{r\in R}C^R\lambda_{rt}+\sum_{k\in W}C^W\omega_{kt}\Bigg] \qquad (11a)\\
\text{s.t. }& \text{(First stage constraints)}\\
& \sum_{q=t-UT_g+1}^{t} z_{gq}\le w_{qt},\quad \forall g\in G_s,\ \forall t\ge UT_g, \qquad (11b)\\
& \sum_{q=t+1}^{t+DT_g} z_{gq}\le 1-w_{gt},\quad \forall g\in G_s,\ \forall t\le |T|-DT_g, \qquad (11c)\\
& z_{gt}\ge w_{gt}-w_{g,t-1},\quad \forall g\in G_s,\ \forall t\in T, \qquad (11d)\\
& \text{(Second stage constraints)}\\
& \sum_{q=t-UT_g+1}^{t} v_{gqs}\le u_{qts},\quad \forall g\in G_f,\ \forall t\ge UT_g,\ \forall s\in\mathcal{S}, \qquad (11e)\\
& \sum_{q=t+1}^{t+DT_g} v_{gqs}\le 1-u_{gts},\quad \forall g\in G_f,\ \forall t\le |T|-DT_g,\ \forall s\in\mathcal{S}, \qquad (11f)\\
& v_{gts}\ge u_{gts}-u_{g,t-1,s},\quad \forall g\in G_f,\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11g)\\
& \sum_{l\in L_n^-} e_{lts}+\sum_{g\in G_n} p_{gts}+\sum_{\lambda\in IG_n^{\mathcal{L}}} x^{\mathcal{L}}_{\lambda ts}+\sum_{\omega\in IG_n^{\mathcal{W}}} IC^{\mathcal{W}}_{\omega ts}\\
& = D_{nt}+\sum_{l\in L_n^+} e_{lts}+\sum_{\iota\in IG_n^{\mathcal{I}}} x^{\mathcal{I}}_{\iota ts}+\sum_{\rho\in IG_n^{\mathcal{R}}} x^{\mathcal{R}}_{\rho ts}+\sum_{\omega\in IG_n^{\mathcal{W}}} x^{\mathcal{W}}_{\omega ts},\\
& \qquad\qquad \forall n\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11h)\\
& e_{lts}=B_l\left(\theta_{n_1ts}-\theta_{n_2ts}\right),\quad \forall l\equiv(n_1,n_2)\in L,\ t\in T, s\in\mathcal{S}, \qquad (11i)\\
& P_g^- w_{gt}\le p_{gts}\le P_g^+ w_{gt},\quad \forall g\in G_s,\ \forall t\in T\cup\{0\},\ \forall s\in\mathcal{S}, \qquad (11j)\\
& P_g^- u_{gts}\le p_{gts}\le P_g^+ u_{gts},\quad \forall g\in G_f,\ \forall t\in T\cup\{0\},\ \forall s\in\mathcal{S}, \qquad (11k)\\
& -R_g^-\le p_{gts}-p_{g,t-1,s}\le R_g^+,\quad \forall g\in G,\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11l)\\
& \text{(First stage variable bounds)}\\
& w_{gt}\in\{0,1\},\quad \forall g\in G_s,\ \forall t\in T\cup\{0\}, \qquad (11m)\\
& 0\le z_{gt}\le 1,\quad \forall g\in G_s,\ \forall t\in T, \qquad (11n)\\
& \text{(Second stage variable bounds)}\\
& u_{gts}\in\{0,1\},\quad \forall g\in G_f,\ \forall t\in T\cup\{0\},\ \forall s\in\mathcal{S}, \qquad (11o)\\
& 0\le v_{gts}\le 1,\quad \forall g\in G_f,\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11p)\\
& -360\le \theta_{nts}\le 360,\quad \forall n\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11q)\\
& -TC_l\le e_{lts}\le TC_l,\quad \forall l\in L,\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11r)\\
& p_{gts}\ge 0,\quad \forall g\in G,\ \forall t\in T\cup\{0\},\ \forall s\in\mathcal{S}, \qquad (11s)\\
& 0\le x^{\mathcal{L}}_{\lambda ts}\le IC^{\mathcal{L}}_{\lambda t},\quad \forall\lambda\in\mathcal{L},\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11t)\\
& 0\le x^{\mathcal{I}}_{\iota ts}\le IC^{\mathcal{I}}_{\iota t},\quad \forall\iota\in\mathcal{I},\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11u)\\
& 0\le x^{\mathcal{R}}_{\rho ts}\le IC^{\mathcal{R}}_{\rho t},\quad \forall\rho\in\mathcal{R},\ \forall t\in T,\ \forall s\in\mathcal{S}, \qquad (11v)\\
& 0\le x^{\mathcal{W}}_{\omega ts}\le IC^{W}_{\omega t},\quad \forall\omega\in\mathcal{W},\ \forall t\in T,\ \forall s\in\mathcal{S}. \qquad (11w)
\end{aligned}
$$

The objective function (11a) is to minimize expected operating costs. Constraints (11b)-(11d) are for the first-stage so constraints for the slow generators. Constraint (11b) and (11c) represent the minimum up/down time of the slow generators. The transition rule for slow generator start-up variables is imposed by constraint (11d). Constraints (11e)-(11g) are stochastic version of the constraints (11b)-(11d) so represent the minimum up/down time and the transition rule of the fast generators. Constraint (11h) requires balancing the amount of power that flows in and out of each bus. Constraint (11i) represents a linearized, lossless model of the power flow equations (Kirchhoff's law) according to which the power flow on a line l is proportional to the phase angle difference between the two end buses of the line. Constraints (11j) and (11k) restrict minimum/maximum capacity limits for both slow and fast generators. Constraint (11l) represents ramping restriction on the rate of change of generator output. Constraints (11m)-(11w) define the types and bounds for decision variables.

Table 14: Notations for the SUC

Index sets:	
$\mathcal{S}$	index set of scenarios ($s \in \mathcal{S}$)
N	index set of all buses ($n \in N$)
T	index set of all time periods ($t \in T$)
L	index set of all transmission lines ($l \in L$)
$\mathcal{L}$	index set of all loads for shedding ($\lambda \in \mathcal{L}$)
$\mathcal{I}$	index set of all import points ($\iota \in \mathcal{I}$)
$\mathcal{R}$	index set of all non-wind renewable generators ($\rho \in \mathcal{R}$)
$\mathcal{W}$	index set of all wind generators ($\omega \in \mathcal{W}$)
G	index set of all generators ($g \in G$)
G_s	index set of slow generators ($g \in G_s$)
G_f	index set of fast generators ($g \in G_f$)
(mapping sets)	
G_n	index set of generators that are located in bus n ($g \in G_n$)
$IG_n^{\mathcal{L}}$	index set of loads that bus n can shed ($\lambda \in IG_n^{\mathcal{L}}$)
$IG_n^{\mathcal{I}}$	index set of import points that can supply bus n ($\iota \in IG_n^{\mathcal{I}}$)
$IG_n^{\mathcal{R}}$	index set of non-wind renewable generators that can supply bus n ($\rho \in IG_n^{\mathcal{R}}$)
$IG_n^{\mathcal{W}}$	index set of wind generators that can supply bus n ($\omega \in IG_n^{\mathcal{W}}$)
L_n^+	index set of outgoing transmission lines from bus n ($l \in L_n^+$)
L_n^-	index set of incoming transmission lines to bus n ($l \in L_n^-$)
Parameters:	
$\mathbb{P}(s)$	probability of occurence for scenario s
(cost)	
K_g	fixed commitment cost of generator g
S_g	fixed startup cost of generator g
C_g	marginal generation cost of generator g
$C^{\mathcal{L}}$	marginal load shedding cost of loads in $\mathcal{L}$
$C^{\mathcal{I}}$	marginal spillage cost of import points in $\mathcal{I}$
$C^{\mathcal{R}}$	marginal spillage cost of non-wind renewable generators in $\mathcal{R}$
$C^{\mathcal{W}}$	marginal spillage cost of wind generators in $\mathcal{W}$
(capacity)	
B_l	susceptance of line l under scenario s
TC_l	capacity of transmission line l
P_g^+	maximum generation capacity of generator g
P_g^-	minimum generation capacity of generator g
R_g^+	maximum ramping capacity of generator g
R_g^-	minimum ramping capacity of generator g
UT_g	minimum up time of generator g
DT_g	minimum down time of generator g
(supply/demand)	
D_{nt}	net demand in bus n at time t
$IC_{\lambda t}^{\mathcal{L}}$	shedding from load λ
$IC_{\iota t}^{\mathcal{I}}$	generation from import point ι
$IC_{\rho t}^{\mathcal{R}}$	generation from renewable generator ρ
$IC_{\omega ts}^{\mathcal{W}}$	generation from wind generator ω under scenario s
Decision variables:	
w_{gt} (1st stage)	(binary) commitment of slow generator g at time t
z_{gt} (1st stage)	(continuous) start-up of slow generator g at time t
u_{gts} (2nd stage)	(binary) commitment of fast generator g at time t under scenario s
v_{gts} (2nd stage)	(continuous) startup of fast generator g at time t under scenario s
θ_{gts} (2nd stage)	(continuous) phase angle of generator g at time t under scenario s
e_{lts} (2nd stage)	(continuous) power flow on line l at time t under scenario s
p_{gts} (2nd stage)	(continuous) production of generator g at time t under scenario s
$x_{\lambda ts}^{\mathcal{L}}$ (2nd stage)	(continuous) load shedding for load λ at time t under scenario s
$x_{\iota ts}^{\mathcal{I}}$ (2nd stage)	(continuous) spillage for import point ι at time t under scenario s
$x_{\rho ts}^{\mathcal{R}}$ (2nd stage)	(continuous) spillage for non-wind renewable generator ρ at time t under scenario s
$x_{\omega ts}^{\mathcal{W}}$ (2nd stage)	(continuous) spillage for wind generator ω at time t under scenario s

A.6.2 SUC: Data generation

For model parameters, we use a reduced model of the CAISO to follow the reference [26]. The model includes a sparse representation of the entire WECC western interconnect outside of California. Based on the WECC, set cardinalities in SUC are as follows.

$$
\begin{aligned}
|N| &= 225 \\
|T| &= 24 \\
|L| &= 375 \\
|\mathcal{L}| &= 40 \\
|\mathcal{I}| &= 5 \\
|\mathcal{R}| &= 11 \\
|\mathcal{W}| &= 5 \\
|G| &= 130 \\
|G_s| &= 40 \\
|G_f| &= 90
\end{aligned}
$$

The marginal cost parameters for additional sources (load, import, non-wind renewable, wind) are fixed by

$$
\begin{aligned}
&C^{\mathcal{L}} = 5000, \\
&C^{\mathcal{I}} = C^{\mathcal{R}} = C^{\mathcal{W}} = 0,
\end{aligned}
$$

which means the cost of shedding any load λ is 5,000 \$/MWh and no penalty cost for spilling the import, non-wind renewable, and wind sources.

The demand data D_{nt} is assumed to be constant over all scenarios and generated based on the load, import, and non-wind renewable source data as follows.

$$
D_{nt} = \sum_{n \in N} \sum_{t \in T} \left(\sum_{\lambda \in IG_n^{\mathcal{L}}} IC_{\lambda t}^{\mathcal{L}} - \sum_{\iota \in IG_n^{\mathcal{I}}} IC_{\iota t}^{\mathcal{I}} - \sum_{\rho \in IG_n^{\mathcal{R}}} IC_{\rho t}^{\mathcal{R}} \right), \quad \forall n \in N,\ \forall t \in T.
$$

Unlike other problems where stochastic data is generated while `Julia` script is running, we pre-generate and provide up to 1,000 wind power production profiles for each day type. This data is included in "`~/Siplib/src/problems/SUC/data/WIND`" folder and used to generate the stochastic parameter $IC_{\omega ts}^{\mathcal{W}}$. Hence, the number of scenarios that can be considered in an instance is limited to 1,000, which we think is large enough regarding the intrinsic large-scale size of the problem SUC.

B Block-wise sparsity based on set cardinality

In this section, we provide block-wise size information as well as the sparsity. Block A is only related with the 1st stage, Block W is only related with the 2nd stage, and Block T is related with both stages. For graphical representation, please refer to Figure 2. Based on this information, one can easily derive the sparsity of the coefficient matrix in extensive form as well.

Table 15: DCAP: Block-wise sparsity information

Block	#row	#col	#nonzero	Sparsity
A (1st stage)	RT	$2RT$	$2RT$	$\frac{1}{RT}$
W (2nd stage)	$(R+N)T$	$(1+R)T$	$NRT+(R+1)NT$	$\frac{1+2R}{T(R+N)(1+R)}$
T (complicating)	$(R+N)T$	$2RT$	$\frac{1}{2}T(T+1)$	$\frac{1+T}{4T(R+N)}$

Table 16: MPTSPs: Block-wise sparsity information

Block	#row	#col	#nonzero	Sparsity
A (1st stage)	$N(N+1)$	$2N(N-1)$	$2(3N-2)(N-1)$	$\frac{3N-2}{(N+1)N^2}$
W (2nd stage)	$N(N-1)$	$KN(N-1)$	$N(N-1)$	$\frac{1}{KN(N-1)}$
T (complicating)	$N(N-1)$	$2N(N-1)$	$3N(N-1)$	$\frac{3}{2N(N-1)}$

Table 17: SIZES: Block-wise sparsity information

Block	#row	#col	#nonzero	Sparsity
A (1st stage)	$T(N+1)$	$2NT$	$3NT$	$\frac{3}{2T(N+1)}$
W (2nd stage)	$2NT$	$N(N+1)$	$\frac{1}{2}NT(T+1)(N+1)$	$\frac{T+1}{4N}$
T (complicating)	$2NT$	$2NT$	$\frac{1}{2}NT(T+1)$	$\frac{T+1}{8NT}$

Table 18: SMKP: Block-wise sparsity information

Block	#row	#col	#nonzero	Sparsity
A (1st stage)	J	$2I$	$2IJ$	1
W (2nd stage)	K	I	IK	1
T (complicating)	K	$2I$	IK	0.5

Table 19: SSLP: Block-wise sparsity information

Block	#row	#col	#nonzero	Sparsity
A (1st stage)	1	J	J	1
W (2nd stage)	$I+J$	$(1+I)J$	$(1+2I)J$	$\frac{1+2I}{(I+J)(1+I)}$
T (complicating)	$I+J$	J	J	$\frac{1}{1+J}$

Table 20: SUC: Block-wise sparsity information

Block	#row	#col	#nonzero
A (1st stage)	$\sum_{g\in G_s}(3T-UT_g-DT_g+1)$	$G_s(2T+1)$	$3G_sT+\sum_{g\in G_s}\left(\sum_{t=UT_g}^{\vert} T\vert\,(1+UT_g)+\sum_{t=1}^{\vert T\vert-DT_g}(1+DT_g)\right)$
W (2nd stage)	$2G+(N+L+4G)T$ $+\sum_{g\in G_f}(3T-UT_g-DT_g+1)$	$G_f+G+(2G_f+G+N$ $+L+\mathcal{L}+\mathcal{I}+\mathcal{R}+\mathcal{W})T$	$\sum_{g\in G_f}\left(\sum_{t=UT_g}^{\vert} T\vert\,(1+UT_g)+\sum_{t=1}^{\vert T\vert-DT_g}(1+DT_g)\right)$ $+T\sum_{n\in N}\left(L_n^-+L_n^+G_n+IG_n^{\mathcal{L}}+IG_n^{\mathcal{I}}+IG_n^{\mathcal{I}}+IG_n^{\mathcal{R}}+IG_n^{\mathcal{W}}\right)$ $+T(6G+5G_f+3L)+2(G+G_f)$
T (complicating)	$2G_s(T+1)$	$G_s(2T+1)$	$2G_s(T+1)$

*We skip sparsity equations due to the lack of space.

C Constraint type legend from MIPLIB 2010

Table 21: Constraint type legend [21]

Type	Description	Constraint form
AGG	Aggregation	$a_i x_i + a_k x_k = b,\ x_i, x_k$ int. or cont., $a_i, a_k, b \in \mathbb{R}$
VBD	Variable bound	$x_i \leq a_k x_k + b$ or $x_i \geq a_k x_k + b$, x_i, x_k int. or cont., $a_k, b \in \mathbb{R}$
PAR	Set partition	$\sum x_i = 1$, x_i binary
PAC	Set packing	$\sum x_i \leq 1$, x_i binary
COV	Set cover	$\sum x_i \geq 1$, x_i binary
CAR	Cardinality	$\sum x_i = b$, x_i binary, $b \in \mathbb{N}$
EQK	Equality knapsack	$\sum a_i x_i = b$, x_i binary, $a_i, b \in \mathbb{N}$
BIN	Bin packing	$\sum a_i x_i + a_k x_k \leq a_k$, x_i binary, $a_i, a_k \in \mathbb{N}$
IVK	Invariant knapsack	$\sum x_i \leq b$, x_i binary, $b \in \mathbb{N}$
KNA	Knapsack	$a_i x_i \leq b$, x_i binary, $a_i, b \in \mathbb{N}$
IKN	Integer knapsack	$a_i x_i \leq b$, $x_i \geq 0$ integer, $a_i, b \in \mathbb{N}$
M01	Mixed binary	$\sum a_i x_i + \sum p_j s_j \leq$ or $= b$, x_i binary, s_j cont., $a_i, p_j \in \mathbb{R}$
GEN	General	All other constraint types

References

1. S. Ahmed, R. Garcia, N. Kong, L. Ntaimo, G. Parija, F. Qiu, S. Sen. SIPLIB: A Stochastic Integer Programming Test Problem Library. http://www.isye.gatech.edu/ sahmed/siplib, 2015.
2. SPS: Stochastic Programming Society (https://stoprog.org/what-stochastic-programming).
3. Introduction to Stochastic Programming, J. R. Birge, F. Louveaus.
4. S. Ahmed and R. Garcia. "Dynamic Capacity Acquisition and Assignment under Uncertainty," Annals of Operations Research, vol.124, pp. 267-283, 2003.
5. Kibaek Kim and Victor M. Zavala. "Algorithmic Innovations and Software for the Dual Decomposition Method applied to Stochastic Mixed-Integer Programs" Mathematical Programming Computation, 2015.
6. J.-P. Watson, D. L. Woodruff, and W. E. Hart, PySP: modeling and solving stochastic programs in Python, Mathematical Programming Computation, 2012.
7. SMI - Stochastic Modeling Interface. https://github.com/coin-or/Smi
8. Julia: A Fresh Approach to Numerical Computing. Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah (2017) SIAM Review, 59: 6598.
9. JuMP - Julia for Mathematical Optimization, https://jump.readthedocs.io/en/latest/index.html
10. StructJuMP - Parallel algebraic modeling framework for block structured optimization models in Julia, https://github.com/StructJuMP/StructJuMP.jl
11. F. Maggioni, G. Perboli, and R. Tadei, The multi-path traveling salesman problem with stochastic travel socts: Building realistic instances for city ligistics applications, Transportation Research Procedia, 2014
12. G. Perboli, L. Gobbato, and F. Maggioni, A progressive hedging method for the multi-path travelling salesman problem with stochastic travel times, IMA Journal of Management Mathematics, 2017
13. R. Tadei, G. Perboli, and F. Perfetti, The multi-path traveling salesman problem with stochastic travel costs, EURO Journal on Transportation and Logistics, 2017
14. Andre Langevin, Francois Soumis, and Jacques Desrosiers, Classification of travelling salesman problem formulations, Operations Research Letters, 1990
15. Soheila Jorjani, Carlton H. Scott, and David L. Woodruff, Selection of an optimal subset of sizes, International Journal of Production Research, 1999
16. B. Gavish and S.C. Graves, The travelling salesman problem and related problems, Working paper GR-078-78, Operations Research Center, Massachusetts Institute of Technology, 1978
17. Gustavo Angulo, Shabbir Ahmed, and Santanu S. Dey, Improving the integer L-shaped method, 2014.
18. Anthony Papavasiliou and Shmuel S. Oren, Multiarea stochastic unit commitment for high wind penetration in a transmission constrained network, Operations Research, 2013
19. Kibaek Kim, Fan Yang, Victor M. Zavala, and Andrew A. Chien, Data centers as dispatchable loads to harness stranded power, IEEE Transactions on sustainable energy, 2017
20. Lewis Ntaimo and Suvrajeet Sen, The million-variable "March" for stochastic combinatorial optimization, Journal of Global Optimization, 2005
21. Koch et al., MIPLIB 2010, Mathmatical Programming Computation, 2011
22. Lee, C., Liu, C., Mehrotra, S., Shahidehpour, M, Modeling transmission line constraints in two-stage robust unit commitment problem, IEEE Transactions on Power Systems, 2014
23. J. R. Birge, M. A. H. Demster, H. I. Gassman, E. A. Gunn, A. J. King, S. W. Wallace, A Standard Input Format for Multiperiod Stochastic Linear Programs, International Institute for Applied Systems Analysis, 1987.
24. Derek Holmes and John Birge, A portable stochastic programming test set (POSTS), http://users.iems.northwestern.edu/ jrbirge/html/dholmes/post.html, 1993.
25. Derek Holmes and John Birge, Computational Results (POSTS), http://users.iems.northwestern.edu/ jrbirge/html/dholmes/POSTresults.html
26. K. A. Ariyawansa and Andrew J. Felt, On a new collection of stochastic linear programming test problems, INFORMS Journal on Computing, 2004.

27. Western Electricity Coordinating Council (WECC) Data Set, Available from: https://www.wecc.biz/SystemAdequacyPlanning/Pages/Datasets.aspx
28. Claus C. Carøe and Rdiger Schultz, Dual decomposition in stochastic integer programming, Operations Research Letters, 1999
29. R. T. Rockafellar and Roger J.-B. Wets, Scenarios and Policy Aggregation in Optimization Under Uncertainty, Mathematics of Operations Research, 1991
30. Willian E. Hart, Carl Laird, Jean-Paul Watson, and David L. Woodruff, Pyomo Optimization Modeling in Python, Springer Optimization and Its Applications, 2012
31. J. F. Benders, Partitioning procedures for solving mixed-variables programming problems, Numerische Mathematik, 1962.
32. G. Laporte and F. V. Louveaux, The integer L-shaped method for stochastic integer programs with complete recourse, Operations Research Letters, 1993.
33. Claus C. Carøe and Jørgen Tind, L-shaped decomposition of two-stage stochastic programs with integer recourse, Mathematical Programming, 1998.