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

SIPLIB 2.0

Stochastic Integer Programming Library version 2.0

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

Received: date / Accepted: date

Abstract We present a collection of stochastic integer programming problem instances.

Keywords Stochastic Integer Programming · Problem Instances

1 Introduction

(What is SIPLIB?) The SIPLIB [2] 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. The instances are basically given in the standard SMPS format accompanied with additional information including parameter data, size of the instance in terms of the number of rows, columns, and integers, benchmarking information such as best known objective value or bounds, optimality gap, and solution time.

(Motivation) 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 solves the instances. The state-of-art in SIP combined with the speedup in computing machinery, however, makes many instances of SIPLIB trivial so that we have no more basis to use SIPLIB for showing the excellence of

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

new 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 expand the size of instance as much as they want in terms of the number of scenarios included.

(What SIP is and our restriction) 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., [3,4]. SIP is a branch of SP that indicates any type of SP including at least one integer decision variable. The integers can be placed anywhere in general SIP. However, we restrict our focus on two-stage SIP that contains integer variables (including binary) in its second stage 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, but only lower semicontinuous. Thus, the standard decomposition approaches that work nicely for stochastic *linear* programs, break down when the second stage integer variables are present [5]. Hereinafter, we use the term SIP to indicate the two-stage SIP that contains integer variables in its second stage.

(SMPS, Julia) We provide the test sets in two formats: `SMPS` files (*.cor, *.tim, *.stoch) and `Julia` files (*.jl). `SMPS` is widely used to describe stochastic linear and quadratic programs. Once having `SMPS` files of a problem instance, we can directly solve it using various mixed integer linear program (MILP) solvers such as `CPLEX`, `GUROBI`, and `CBC`. One can also use the existing open-source SIP solvers like `DSP` [6], `PySP` [7], and `SMI` [8] given that `SMPS` files. A drawback of `SMPS` format is its low readability by human, which we decided to provide `Julia` files to let users be able to easily read problems and tailor the instances.

(Convenience of Julia) `Julia` is an open source high-level, high-performance dynamic programming language for numerical computing. It is also known as its good performance, approaching that of statically-compiled languages like `C` [9]. 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 [10]) provides a domain-specific modeling language for mathematical optimization embedded in `Julia`. `JuMP` enables us to easily translate mathematical model to `JuMP.Model`-type object. Some structured mathematical models like SIP can also be translated to the `JuMP.Model`-type object combined with the package `StructJuMP` [11]. Once we have a `Julia` code for constructing `JuMP.Model`-type object, it is easy to generate instances whenever we need to modify the original mathematical model. For each problem in `SIPLIB 2.0`, we provide a `Julia` script for constructing `JuMP.Model`-type object. We also provide a `Julia` script (SmpsWriter.jl) for converting any `JuMP.Model`-type objects to `SMPS` files for users' convenience. Those who feel the given instances are not large enough can simply generate more scenario data by just modifying the parameter in `Julia` script corresponding to the number of scenarios.

(Contribution of SIPLIP2.0) By `SIPLIB 2.0`, we provide 1) richer collection of test instances for computational and algorithmic research in SIP with benchmarking computational results, 2) not only `SMPS` files but also `Julia` files that are easily readable/tailorable. Hence, the users can obtain as large-sized instances as they need by generating new scenarios and including them into instances. For those who want to utilize the instances in the legacy `SIPLIB` with strong tailorability provided by `SIPLIB 2.0`, we include the original `SIPLIB` instances as well.

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 given 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 $p_1, \cdots, p_r$, i.e., $p_i := \mathbb{P}[\boldsymbol{\xi} = \xi_i]$ for $i \in \{1, \ldots, r\}$. When the distribution is continuous, we can aproximate it by a suitably discretized distribution. The real-valued map $\phi : \mathbb{R}^{m_2} \to \mathbb{R}$ is the optimal value of the second-stage problem defined by

$$\phi(s; \xi) := \min_{y \in Y} \left\{ q(\xi)^\top y : \; W(\xi) y \geq s \right\}, \; s \in \mathbb{R}^{m_2}, \tag{2}$$

where the r.v. $\boldsymbol{\xi}$ is realized by an arbitrary scenario ξ. 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, respectively. The first-stage problem data comprise A, b, and c. The second-stage data are given by $T(\xi_j)$, $W(\xi_j)$, $h(\xi_j)$, and $q(\xi_j)$ (for dimensional information refer to Table 1). Hereinafter, we use the simplified notations (T_j, W_j, h_j, q_j). The SIP (1) can be rewritten in the extensive form

$$z = \min_{x_j, y_j} \sum_{j=1}^{r} p_j \left(c^\top x_j + q_j^\top y_j \right) \tag{3a}$$

$$\text{s.t.} \sum_{j=1}^{r} H_j x_j = 0 \tag{3b}$$

$$(x_j, y_j) \in G_j, \quad \forall j \in \{1, \dots, r\}, \tag{3c}$$

where the scenario feasibility set G_j is defined as

$$G_j := \{(x_j, y_j) : \ Ax_j \geq b, \ T_j x_j + W_j y_j \geq h_j, \ (x_j, y_j) \in X \times Y\}. \tag{4}$$

The nonanticipativity constraints in (3b) stand for the equations $x_1 = x_r$ and $x_j = x_{j-1}$ for $j = 2, \dots, r$, and H_j is a suitable $rn_1 \times n_1$ matrix. We assume that SIP does not necessarily have relatively complete recourse. We recall that without this property there can exist an $\hat{x} \in X$ satisfying $A\hat{x} \geq b$ for which there does not exist a recourse $y \in \mathbb{R}^{m_2}$ satisfying $(\hat{x}, y) \in G_j$ for some j. In other words, not every choice of the first-stage variables is guaranteed to have feasible recourse for all scenarios.

Table 1 Summary of notations in SIP formulation

Scalas	
$\boldsymbol{\xi}$	the r.v. denoting scenario that can have value in the set $\{\xi_1, \cdots, \xi_r\}$
$z \in \mathbb{R}$	the optimal objective value of the SIP
$r \in \mathbb{N}$	the number of scenarios
$j \in \{1, \cdots, r\}$	the index denoting scenarios
$p_j \in [0, 1]$	the probability that the scenario j happens, i.e., $\mathbb{P}[\boldsymbol{\xi} = \xi_j]$
Sets	
$X \subseteq \mathbb{R}^{n_1}$	the first-stage polyhedral set (real, integer, binary)
$Y \subseteq \mathbb{R}^{n_2}$	the second-stage polyhedral set (real, integer, binary)
G_j	the scenario feasibility set
Vectors	
$x, x_j \in \mathbb{R}^{n_1}$	the first-stage decision vector
$c \in \mathbb{R}^{n_1}$	the first-stage cost vector
$b \in \mathbb{R}^{m_1}$	the first-stage RHS vector
$y, y_j \in \mathbb{R}^{n_2}$	the second-stage decision vector
$q_j \equiv q(\xi_j) \in \mathbb{R}^{n_2}$	the second-stage cost vector
$h_j \equiv h(\xi_j) \in \mathbb{R}^{m_2}$	the second-stage RHS vector
$\mathbf{0} \in \mathbb{R}^{rn_1}$	the vector filled with zeros
Matrices	
$A \in \mathbb{R}^{m_1 \times n_1}$	the first-stage constraint matrix corresponds to the decision vector x_j
$W_j \equiv W(\xi_j) \in \mathbb{R}^{m_2 \times n_2}$	the second-stage constraint matrix corresponds to the decision vector y_j
$T_j \equiv T(\xi_j) \in \mathbb{R}^{m_2 \times n_1}$	the second-stage constraint matrix corresponds to the decision vector x_j
$H_j \equiv H(\xi_j) \in \mathbb{R}^{rm_1 \times n_1}$	nonanticipativity constraints matrix
Functions	
$\phi : \mathbb{R}^{m_2} \to \mathbb{R}$	the second stage program optimal value given the realization of scenario ξ_j
$\mathcal{Q} : \mathbb{R}^{n_1} \to \mathbb{R}$	the recourse function (the expectation of $\phi\left(h(\boldsymbol{\xi}) - T(\boldsymbol{\xi})x\right)$ over the r.v. $\boldsymbol{\xi}$)

2.2 Solution methods

2.2.1 Stage-wise decomposition algorithm

2.2.2 Scenario-wise decompostion algorithm

2.3 Software libraries

2.3.1 Modeling languages

2.3.2 Solvers

3 The test sets

3.1 The type of problems

Table 2 The type of problems in `SIPLIB 2.0`

Problem name	Description	Reference	Is in `SIPLIB`?
`DCAP`	Dynamic capacity planning with stochastic demand	Ahmed and Garcia	Yes
`MPTSPS`	Multi-path traveling salesman problem with stochastic travel costs	[12,13,14]	Yes
`SMKP`	Stochastic multiple knapsack problem	Angulo et al.	Yes
`SIZES`	Optimal product substitution with stochastic demand	Jorjani et al.	Yes
`SSLP`	Stochastic server location problem	Ntaimo and Sen	Yes
`WECC`	Wind power stochastic unit commitment	Papavasiliou and Oren	No

Table 3 The components of problems in `SIPLIB 2.0`

	1st stage		2nd stage	
Problem	Variable	Constraint	Variable	Constraint
DCAP	`real`, `bin`		`bin`	PAR, MO11
MPTSPS	`bin`, `real`	PAR2, GEN1	`bin`	GEN1
SMKP	`bin`	KNA1	`bin`	KNA1
SIZES	`real`, `int`	VBD1, GEN2	`real`, `int`	IKN1
SSLP	`bin`	IVK1, GEN1	`real`, `int`	GEN2
WECC				

3.2 The instance catalog

3.2.1 Instance naming rule

Table 4 Problem-specific instance naming rules

Problem	Instance name	Description
DCAP		
MPTSPS	MPTSPs_Dx_Ny_Sz	MPTSPS with node distribution strategy Dx, number of nodes y, and number of scenarios z
SMKP		
SIZES		
SSLP		
WECC		

4 How to run a test, generate new instance, and convert to SMPS

We explain the structure of SIPLIB 2.0. We explain procedure to generate new instances with user-generated scenario data using Julia scripts. The problem-specific descriptions are given in Section 6. We explain how to convert JuMP.Model-type object to SMPS files.

5 Implementation of SMPS Writer

We describe our Julia implementation, how to model SIP and generate SMPS files..

6 Problem descriptions

In this section, we introduce each problem in SIPLIB 2.0. For each problem, we provide various size of instances as a default. We also explain the scenario data generation procedure for each problem. Due to limited access to the original data and inconsistencies present in reference papers, we selectively choose the methods from the references and modify some of them without harming validity. Also, we guess some parameters about scenario generation to make the procedure clear.

6.1 `MPTSPS`: Mutli-path Traveling Salesman Problem with Stochastic Travel Times

6.1.1 Mathematical formulation

Table 5 Notations for `MPTSPS`

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

$$
\begin{aligned}
(\texttt{MPTSPS})\ \min\ & \sum_{i\in N}\sum_{j\in N} \bar{c}_{ij} y_{ij} + \sum_{s\in S} p_s \sum_{i\in N}\sum_{j\in N}\sum_{k\in K_{ij}} e_{ij}^{ks} x_{ij}^{ks} && \text{(5a)}\\
\text{s.t.}\ & \sum_{j\in N: j\neq i} y_{ij} = 1, \quad \forall i \in N, && \text{(5b)}\\
& \sum_{i\in N: i\neq j} y_{ij} = 1, \quad \forall j \in N, && \text{(5c)}\\
& \sum_{j\in N} \phi_{lj} - \sum_{i\in N: i\neq 1} \phi_{il} = 1, \quad \forall l \in N\backslash\{1\}, && \text{(5d)}\\
& \phi_{ij} \le (|N|-1)\, y_{ij}, \quad \forall i \in N\backslash\{1\},\ \forall j \in N, && \text{(5e)}\\
& \sum_{k\in K_{ij}} x_{ij}^{ks} = y_{ij}, \quad \forall i \in N,\ \forall j \in N,\ \forall s \in S, && \text{(5f)}\\
& x_{ij}^{ks} \in \{0,1\}, \quad \forall i \in N,\ \forall j \in N,\ \forall k \in K_{ij},\ \forall s \in S, && \text{(5g)}\\
& y_{ij} \in \{0,1\}, \quad \forall i \in N,\ \forall j \in N, && \text{(5h)}\\
& \phi_{ij} \ge 0, \quad \forall i \in N,\ \forall j \in N. && \text{(5i)}
\end{aligned}
$$

6.1.2 Scenario data generation

We follow the scenario generation methods described through the references [12,13,14]. For `MPTSPS`, there are three mainly distinguished characteristics for each instance: number of nodes ($N \in \{2, 3, \ldots\}$), number of paths for each edge ($|K_{ij}| \in \{1, 2, 3, \ldots\}$), and nodes partition strategy ($D \in \{D0, D1, D2, D3\}$). Following [14], $|K_{ij}|$ is fixed by 3 as a default. Once we decide D and N, each instance is named by `MPTSPS_D_N`.

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 length of each path between two nodes are affected by the type of arc.

The nodes are generated by 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*.

After determining N and D, 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. Following [14], we fix $r = 7km$.
2. Calculate Euclidean distances between the nodes (EC_{ij}).
3. We fix the deterministic velocity profile by 40 *km/h* for the *central* nodes and 80 *km/h* for the *suburban* nodes: $v^c = 40$ and $v^s = 80$.
4. Generate random travel times (c_{ij}^{ks}) 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 $v^c = 40$ and the remaining paths are generated based on $v^s = 80$.
   - The velocities are distributed by $Unif(\frac{v}{2}, 2v)$ for $v = v^c, v^s$.
   - In summary, if the arc (i, j) is *homogeneous*,

$$c_{ij}^{ks} \sim \begin{cases} \frac{EC_{ij}}{Unif(\frac{v^c}{2}, 2v^c)} & \text{if } i, j \text{ are both } \textit{central}, \\ \frac{EC_{ij}}{Unif(\frac{v^s}{2}, 2v^s)} & \text{if } i, j \text{ are both } \textit{suburban} \end{cases} \quad \forall k \in K_{ij}.$$

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

$$c_{ij}^{ks} \sim \begin{cases} \frac{EC_{ij}}{Unif(\frac{v^c}{2}, 2v^c)} & \text{for } k \in \left\{1, \ldots, \left\lceil \frac{|K_{ij}|}{3} \right\rceil \right\}, \\ \frac{EC_{ij}}{Unif(\frac{v^s}{2}, 2v^s)} & \text{for } k \in \left\{ \left\lceil \frac{|K_{ij}|}{3} \right\rceil + 1, \ldots, |K_{ij}| \right\}. \end{cases}$$

5. Finally, we multiply 3600 for each component of c_{ij}^{ks} to convert the unit from *hours* to *seconds*.

7 Solution report

8 Concluding remarks

References

1. Author1 and Author2, paper paper paper paper, Journal Title 68 (2011), 1207–1221.
2. 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.
3. SPS: Stochastic Programming Society (https://stoprog.org/what-stochastic-programming).
4. Introduction to Stochastic Programming, J. R. Birge, F. Louveaus.
5. S. Ahmed and R. Garcia. "Dynamic Capacity Acquisition and Assignment under Uncertainty," Annals of Operations Research, vol.124, pp. 267-283, 2003.
6. 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.
7. J.-P. Watson, D. L. Woodruff, and W. E. Hart, PySP: modeling and solving stochastic programs in Python, Mathematical Programming Computation, 2012.
8. SMI - Stochastic Modeling Interface. https://github.com/coin-or/Smi
9. Julia: A Fresh Approach to Numerical Computing. Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah (2017) SIAM Review, 59: 6598.
10. JuMP - Julia for Mathematical Optimization, https://jump.readthedocs.io/en/latest/index.html
11. StructJuMP - Parallel algebraic modeling framework for block structured optimization models in Julia, https://github.com/StructJuMP/StructJuMP.jl
12. 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
13. 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
14. R. Tadei, G. Perboli, and F. Perfetti, The multi-path traveling salesman problem with stochastic travel costs, EURO Journal on Transportation and Logistics, 2017