Stochastic Package Queries in Probabilistic Databases

Matteo Brucato* Nishant Yadav*
Azza Abouzied° Peter J. Haas* Alexandra Meliou*
*University of Massachusetts Amherst °NYU Abu Dhabi
{matteo, nishantyadav, phaas, ameli}@cs.umass.edu azza@nyu.edu

ABSTRACT

We provide methods for in-database support of decision making under uncertainty. Many important decision problems correspond to selecting a *package* (bag of tuples in a relational database) that jointly satisfy a set of constraints while minimizing some overall cost function; in most real-world problems, the data is uncertain. We provide methods for specifying—via a SQL extension—and processing *stochastic package queries (SPQs)*, in order to solve optimization problems over uncertain data, right where the data resides. Prior work in stochastic programming uses Monte Carlo methods where the original stochastic optimization problem is approximated by a large deterministic optimization problem that incorporates many *scenarios*, i.e., sample realizations of the uncertain data values. For large database tables, however, a huge number of scenarios is required, leading to poor performance and, often, failure of the solver software. We therefore provide a novel SummarySearch algorithm that, instead of trying to solve a large deterministic problem, seamlessly approximates it via a sequence of smaller problems defined over carefully crafted *summaries* of the scenarios that accelerate convergence to a feasible and near-optimal solution. Experimental results on our prototype system show that SummarySearch can be orders of magnitude faster than prior methods at finding feasible and high-quality packages.

KEYWORDS

package queries; probabilistic databases; stochastic programming; optimization; simulation; prescriptive analytics; decision making; portfolio optimization; data integration

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389765

ACM Reference Format:
Matteo Brucato, Nishant Yadav, Azza Abouzied, Peter J. Haas, and Alexandra Meliou. 2020. Stochastic Package Queries in Probabilistic Databases. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3389765

1 INTRODUCTION

Constrained optimization is central to decision making over a broad range of domains, including finance [24, 28], transportation [13], healthcare [21], the travel industry [15], robotics [19], and engineering [2]. Consider, for example, the following very common investment problem.

Example 1 (Financial Portfolio). *Given uncertain predictions for future stock prices based on financial models derived from historical data, an investor wants to invest* $1,000 *in a set of trades (decisions on which stocks to buy and when to sell them) that will maximize the* expected future gain*, while ensuring that the* loss *(if any) will be lower than* $10 *with probability at least* 95%*.*

Suppose each row in a table contains a possible stock trade an investor can make: whether to buy one share of a certain stock, and when to sell it back, as shown in the left-hand side of Figure 1. The investor wants a "package" of trades—a subset of the input table, with possible repetitions (i.e., multiple shares)—that is *feasible*, in that it satisfies the given constraints (total price at most $1,000 and loss lower than $10 with probability at least 95%), and *optimal*, in that it maximizes an objective (expected future gain). Although the current price of a stock is known—i.e., **price** is a *deterministic* attribute—its future price, and thus the gain obtained after reselling the stock, is *unknown*. In the input table, **Gain** is a *stochastic attribute*. If the future gains were known, Example 1 would be a "package query" [3, 4], directly solvable as an *Integer Linear Program* (ILP) using off-the-shelf linear solvers such as IBM CPLEX [25], and declaratively expressible in the Package Query Language (PaQL). Because **Gain** is stochastic, the investor is solving a *stochastic ILP* instead. In this paper, we introduce *stochastic package queries* (SPQs), a generalization of package queries that allows uncertainty in the data, thereby allowing specification and solution of stochastic ILP problems.

Stock_Investments (Table)

id	stock	price	sell_in	Gain
1	AAPL	234	1 day	?
2	AAPL	234	1 week	?
3	MSFT	140	1 day	?
4	MSFT	140	1 week	?
5	TSLA	258	1 day	?
6	TSLA	258	1 week	?

Stochastic Package Query (sPaQL)

```
SELECT PACKAGE(*) AS Portfolio
FROM Stock_Investments
SUCH THAT
  SUM(price) ≤ 1000 AND
  SUM(Gain) ≥ −10 WITH PROBABILITY ≥ 0.95
MAXIMIZE EXPECTED SUM(Gain)
```

Portfolio (Package)

id	stock	price	sell_in	Gain
3	MSFT	140	1 day	?
3	MSFT	140	1 day	?
6	TSLA	258	1 week	?

"Buy 2 MSFT shares, sell them tomorrow. Buy 1 TSLA share, sell it in 1 week".

Figure 1: Example input table for the Financial Portfolio (left), its stochastic package query expression in sPaQL (center), and an example output package (right) with a description of its meaning for the investor. Stochastic attributes (Gain, in this example) are denoted in small caps and their values are unknown (shown by a question mark). Sample realizations of the uncertain ? values are generated by calls to VG functions.

We first introduce a simple language extension to PaQL, called sPaQL, that allows easy specification of package queries with stochastic constraints and objectives. We show the sPaQL query for Example 1 in Figure 1. The result of the query, on the right-hand side of the figure, is a package that informs the investor about how many trades to buy for each individual stock, and when to plan reselling them to the stock market.

Probabilistic databases [14, 40] enable the representation of random variables in a database. The Financial Portfolio, like many other real-world applications, typically uses complex distributions to model uncertainty. For instance, future stock prices are sometimes forecast using lognormal variates based on "geometric Brownian motion" [36] using historical stock price data; alternatively, forecasts can incorporate complex stochastic predictive simulation or machine learning models. For this reason, we base SPQs on the Monte Carlo probabilistic data model [26, 27], which offers support for arbitrary distributions via user-defined *variable generation* (VG) functions. To generate a sample realization of the random variables in a database, the system calls the appropriate VG functions. Whereas existing probabilistic databases excel at supporting SQL-like queries under uncertainty, they do not support package-level optimization, and therefore cannot answer SPQs. PackageBuilder [3, 4], on the other hand, only supports deterministic package queries and their translation into deterministic ILPs.

The state of the art in solving stochastic ILPs (SILPs) has been developed outside of the database setting, in the field of *stochastic programming* (SP) [1, 11, 23]. SP techniques approximate the given SILP by a large deterministic ILP (DILP) that simultaneously incorporates multiple *scenarios*. In a Monte Carlo database, a scenario is obtained by generating a realization of every random variable in the table, via a call to each associated VG function; this procedure may be repeated multiple times, generating a set of scenarios that are mutually independent and identically distributed (iid). Figure 2 shows an example of three possible scenarios for the input investment table for Example 1. Roughly speaking, expectations in the SILP are approximated by averages over the scenarios and probabilities by relative frequencies to form the DILP, which is then fed to a standard solver (e.g., CPLEX). The obtained solution approximates the true optimal solution for the SILP; the more scenarios, the better the approximation.

The solution of the DILP, however, may not be feasible with respect to the original SILP, especially if the approximation is based on only a small number of scenarios that do not well represent the true uncertainty distribution. For example, a financial package obtained by using too few scenarios might guarantee a loss less than $10 with a probability of only 65%, rather then 95%, incurring more risk than desired.

There is no practical way to know how many scenarios will be needed *a priori*; existing theoretical a-priori bounds—see, e.g., [32]—are usually too conservative to be usable when table sizes are large. For example, if the Stock_Investments table contains N=50,000 rows, then to guarantee that the DILP solution is feasible for the SILP with merely 0.1% probability (which is really no guarantee at all), one would need 690,000 scenarios, resulting in a DILP with 34.5 *billion* coefficients! SP solutions must therefore be "validated" *a posteriori*, using a much larger, and out-of-sample, set of scenarios. In Example 1, for instance, we would generate, say, 10^6 scenarios and verify that the loss is less than $10 in at least 95% of them; such validation is much faster than solving a DILP with 10^6 scenarios.

The state-of-the-art algorithm thus works in a loop: the *optimization phase* creates scenarios, combines them into a DILP, and computes a solution; the *validation phase* validates the solution against the out-of-sample scenarios. If the solution is feasible on the validation scenarios (*validation-feasible*), the algorithm terminates, otherwise it creates more scenarios and repeats. A solution that is validation-feasible is highly likely to be truly feasible for the original SILP. Typically the ultimate number of scenarios used to compute the optimal solution to the DILP is astronomically smaller

Scenario 1

id	...	gain
1	...	0.1
2	...	0.05
3	...	-0.2
4	...	0.2
5	...	0.1
6	...	-0.7

Scenario 2

id	...	gain
1	...	-0.2
2	...	-0.03
3	...	0.5
4	...	0.7
5	...	-0.7
6	...	-0.001

Scenario 3

id	...	gain
1	...	0.01
2	...	0.02
3	...	-0.1
4	...	-0.3
5	...	0.2
6	...	0.3

Figure 2: Three example scenarios for the Stock_Investments **table, each showing only the ids and specific realizations for the stochastic attribute** Gain.

than the number prescribed by the conservative theoretical bounds (though it is still large enough to be extremely computationally challenging).

Unfortunately, this process often breaks down in practice. Uncertainty increases with increasing table size, and large tables typically need a huge number of scenarios to achieve feasibility. Thus the validation phase repeatedly fails, and the scenario set—and hence the DILP—grows larger and larger until the solver is overwhelmed. Even if the solver can ultimately handle the problem, many ever-slower iterations may be required until validation-feasible solutions are found, resulting in poor performance.

In this paper, we present an end-to-end system for SPQs, seamlessly connecting SILP optimization with data management and stochastic predictive modeling. Thus tasks related to efficiently storing data, maintaining consistency, controlling access, and efficiently retrieving and preparing the data for analysis can leverage the full power of a DBMS, while avoiding the usual slow, cumbersome, and error-prone analytics workflow where we read a dataset off of a database into main memory, feed it to stochastic-prediction and optimization packages, and store the results back into the database.

We first introduce a Naïve query evaluation algorithm, which embodies the state-of-the-art optimization/validation technique outlined above, and thoroughly discuss its drawbacks. (Although the Naïve technique is mentioned in the SP literature, to our knowledge this is the first systematic implementation of the approach.) We then introduce our new algorithm, SummarySearch, that is typically faster than Naïve by orders of magnitude and can handle problems that cause Naïve to fail.

Our key observation is that the randomly selected set of scenarios used to form the DILP during an iteration of Naïve tend to be overly "optimistic", leading the solver towards a seemingly good solution that "in reality"—i.e., when tested against the validation scenarios—turns out to be infeasible. This problem is also known as the "optimizer's curse" [39].

To overcome the optimizer's curse, SummarySearch replaces the large set of scenarios used to form the Naïve DILP by a very small synopsis of the scenario set, called a "summary", which results in a "reduced" DILP that is much smaller than the Naïve DILP. A summary is carefully crafted to be "conservative" in that the constraints in the reduced DILP are harder to satisfy than the constraints in the Naïve DILP. Because the reduced DILP is much smaller than the Naïve DILP, it can be solved much faster; moreover, the resulting solution is much more likely to be validation-feasible, so that the required number of optimization/validation iterations is typically reduced. Of course, if a summary is overly conservative, the resulting solution will be feasible, but highly suboptimal. Therefore, during each optimization phase, SummarySearch implements a sophisticated search procedure aimed at finding a "minimally" conservative summary; this search requires solution of a sequence of reduced DILPs, but each can be solved quickly.

Our experiments (Section 6) show that, since its iterations are much faster than those of Naïve, SummarySearch exhibits a large net performance gain even when the number of iterations is comparable; typically, the number of iterations is actually much lower for SummarySearch than for Naïve, further augmenting the performance gain.

In summary, the contributions of our paper are as follows.

- We extend the PaQL language for deterministic package queries (itself an extension of SQL); the resulting language, sPaQL,[1] allows specification of package queries with stochastic constraints and objectives.
- We provide a precise and concrete embodiment, the Naïve algorithm, of the optimization/validation procedure suggested by the SP literature (Section 3).
- We provide a novel algorithm, SummarySearch, that is orders-of-magnitude faster than Naïve, and that can solve SPQs that require too many scenarios for Naïve to handle. This is a significant contribution and fundamental extension to the known state-of-the-art in stochastic programming (Section 4).
- We present techniques that allow SummarySearch to optimize its parameters automatically, and we provide theoretical approximation guarantees on the solution of SummarySearch relative to Naïve (Section 5).
- We provide a comprehensive experimental study, which indicates that SummarySearch always finds validation-feasible solutions of high quality, even when Naïve cannot, with dramatic speed-ups relative to Naïve (Section 6).

Section 7 discusses related work, and we conclude in Section 8. Our SPQ techniques represent a significant step towards data-intensive decision making under uncertainty.

[1] A detailed description of sPaQL can be found in the online appendix [5].

2 PRELIMINARIES

Our work lies at the intersection of package queries, probabilistic databases, and stochastic programming. In this section, we introduce some basic definitions from these areas that we will use throughout the paper.

2.1 Deterministic Package Queries

A *package* P of a relation R is a relation obtained from R by inserting $m_{\mathsf{P}}(t) \geq 0$ copies of t into P for each $t \in \mathsf{R}$; here m_{P} is the *multiplicity function* of P. The goal of a *package query* is to specify m_{P}, and hence the tuples of the corresponding package relation. A package query may include a WHERE clause (tuple-level constraints), a SUCH THAT clause (package-level constraints), a package-level objective predicate and, possibly, a REPEAT limit, i.e., an upper bound on the number of duplicates of each tuple in the package.

A deterministic package query can be translated into an equivalent *integer program* [3]. For each tuple $t_i \in \mathsf{R}$, the translation assigns a nonnegative integer decision variable x_i corresponding to the multiplicity of t_i in P, i.e., $x_i = m_{\mathsf{P}}(t_i)$. If the objective function and all constraints are linear in the x_i's, the resulting integer program is an ILP. A cardinality constraint $\mathsf{COUNT}(*) = 3$ is translated into the ILP constraint $\sum_{i=1}^{N} x_i = 3$. A summation constraint $\mathsf{SUM}(\textbf{price}) \leq 1000$ is translated into $\sum_{i=1}^{N} t_i.\textbf{price}\, x_i \leq 1000$; this translation works similarly for other linear constraints and objectives. A REPEAT l constraint is translated into bound constraints $x_i \leq l + 1, \forall i \in [1..N]$.

2.2 Monte Carlo Relations

We use the Monte Carlo database model to represent uncertainty in a probabilistic database. Uncertain values are modeled as random variables, and a scenario (a deterministic realization of the relation) is generated by invoking all of the associated VG functions for the relation. In the simplest case, where all random variables are statistically independent, each random variable has its own VG function; in general, multiple random variables can share the same VG function, allowing specification of various kinds of statistical correlations. A Monte Carlo database system such as MCDB [26] (or its successor, SimSQL [6]) facilitates specification of VG functions as user-defined functions. We assume that there exists a deterministic key column that is the same in each scenario, so that each scenario contains exactly N tuples for some $N \geq 1$ and the notion of the "ith tuple t_i" is well defined across scenarios. For simplicity, we focus henceforth on the case where a database comprises a single relation. Our results extend to Monte Carlo databases containing multiple (stochastic) base relations in which the SPQ is defined in terms of a relation obtained via a query over the base relations.

2.3 Stochastic ILPs

The field of stochastic programming (SP) [29, 37] studies optimization problems—selecting values of decision variables, subject to constraints, to optimize an objective value—having uncertainty in the data. We focus on SILPs with linear constraints and linear objectives that are deterministic, expressed as expectations, or expressed as probabilities. Probabilistic constraints are also called "chance" constraints in the SP literature.

Linear constraints. Given random variables $\xi_1, \ldots, \xi_N$, decision variables $x_1, \ldots x_N$, a real number $v \in \mathbb{R}$, and a relation $\odot \in \{\leq, \geq\}$, a linear *expectation constraint* takes the form $\mathbb{E}\big(\sum_{i=1}^{N} \xi_i x_i\big) \odot v$, and a linear *probabilistic constraint* takes the form $\Pr\big(\sum_{i=1}^{N} \xi_i x_i \odot v\big) \geq p$, where $p \in [0, 1]$. We refer to the constraint $\sum_{i=1}^{N} \xi_i x_i \odot v$ as the *inner constraint* of the probabilistic constraint. Constraints of the form $\Pr(\cdot) \leq p$ can be rewritten in the aforementioned form by flipping the inequality sign of the inner constraint and using $1 - p$ instead. If for constants $c_1, \ldots, c_N \in \mathbb{R}$ we have $\Pr(\xi_i = c_i) = 1$ for $i \in [1..N]$, then we obtain the deterministic constraint $\sum_{i=1}^{N} c_i x_i \odot v$ as a special case of an expectation constraint.

Objective. Without loss of generality, we assume throughout that the objective has the canonical form $\min_x \sum_{i=1}^{N} c_i x_i$ for deterministic constants $c_1, \ldots, c_N$. Indeed, observe that an objective in the form of an expectation of a linear function can be written in canonical form: $\min_x \mathbb{E}\big(\sum_{i=1}^{N} \xi_i x_i\big) = \min_x \sum_{i=1}^{N} \mathbb{E}(\xi_i)\, x_i$, and thus we take $c_i = \mathbb{E}(\xi_i)$. (This assumes that each expectation $\mathbb{E}(\xi_i)$ is known or can be accurately approximated.) Similarly, an objective in the form of a probability can be written in canonical form using epigraphic rewriting [10]. For example, we can rewrite an objective of the form $\min_x \Pr\big(\sum_{i=1}^{N} \xi_i x_i \odot v\big)$ in canonical form as $\min_{x,y} y$ and add a new probabilistic constraint $\Pr\big(\sum_{i=1}^{N} \xi_i x_i \odot v\big) \leq y$. Here $c_1 = \cdots c_N = 0$ and y is an artificial decision variable added to the problem with objective coefficient $c_y = 1$. Throughout the rest of the paper, we will primarily focus on techniques for minimization problems with a nonnegative objective function; the various other possibilities can be handled with suitable modifications.

In our database setting, we assume for ease of exposition that, in a given constraint or objective, each random variable ξ_i corresponds to a random attribute value $t_i.\mathbf{A}$ for some real-valued attribute $\mathbf{A}$; a different attribute can be used for each constraint, and need not be the same as the attribute that appears in the objective. Our methods can actually support more general formulations: e.g., an expectation objective of the form $\min_x \mathbb{E}\big(\sum_{i=1}^{N} g(t_i) x_i\big)$, where g is an arbitrary real-valued function of tuple attributes; constraints can similarly be generalized. Note that this general form allows categorical attributes to be used in addition to real-valued attributes.

Algorithm 1 Naïve Monte Carlo Query Evaluation

$\mathcal{Q}$: A stochastic package query
$\hat{M}$: Number of out-of-sample validation scenarios (e.g., 10^6)
M : Initial number of optimization scenarios (e.g., 100)
m : Iterative increment to M (e.g., 100)
output: A feasible package solution x, or failure (no solution).

1: $S \leftarrow$ GenerateScenarios($\mathcal{Q}, M$) ▷ *Optimization scenarios*
2: **repeat**
3: $\text{SAA}_{\mathcal{Q},M} \leftarrow$ FormulateSAA($\mathcal{Q}, S$) ▷ *Approximate DILP*
4: $x \leftarrow$ Solve($\text{SAA}_{\mathcal{Q},M}$) ▷ *Solve SAA with M scenarios*
5: $\hat{v}_x \leftarrow$ Validate($x, \mathcal{Q}, \hat{M}$) ▷ *Validate using $\hat{M}$ scenarios*
6: **if** $\hat{v}_x$.is_feasible **is** True **then** ▷ *x is feasible*
7: **return** x
8: ▷ *Otherwise, use more optimization scenarios*
9: $S \leftarrow S \cup$ GenerateScenarios($\mathcal{Q}, m$)
10: $M \leftarrow M + m$

3 NAÏVE SILP APPROXIMATION

Recall that Naïve is the first systematic implementation of the optimization/validation approach mentioned in the SP literature. The pseudocode is given as Algorithm 1. As discussed previously, the algorithm generates scenarios (line 1), combines them into an approximating DILP (line 3), solves the DILP to obtain a solution x (line 4), and then validates the feasibility of x against a large number of out-of-sample validation scenarios (line 5). The process is iterated, adding additional scenarios at each iteration (line 10) until the validation phase succeeds. We now describe these steps in more detail.

As discussed in the Introduction, the optimization phase for the DILP can be very slow, and often the convergence to feasibility requires so many optimize/validate iterations that the DILP becomes too large for the solver to handle, so that Naïve fails. Our novel SummarySearch algorithm in Section 4 uses "summaries" to speed up the optimization phase and reduce the number of required iterations.

3.1 Sample-average approximation

As mentioned previously, we can generate a scenario by invoking all of the VG functions for a table to obtain a realization of each random variable, and can repeat this process M times to obtain a Monte Carlo sample of M iid scenarios. In our implementation, Naïve generates scenarios by seeding the random number generator once for the entire execution, and maintains scenarios in main memory.

We then obtain the DILP from the original SILP by replacing the distributions of the random variables with the empirical distributions corresponding to the sample. That is, the probability of an event is approximated by its relative frequency in the sample, and the expectation of a random variable by its sample average. In the SP literature, this approach is known as *Sample Average Approximation* (SAA) [1, 32], and we therefore refer to the DILP for the stochastic package query $\mathcal{Q}$ as $\text{SAA}_{\mathcal{Q},M}$.

More formally, suppose that we have M scenarios $S_1, \ldots, S_M$, each with N tuples, and that the SILP has K probabilistic constraints. Recall that $t_i.\mathbf{A}$ denotes the random variable corresponding to attribute $\mathbf{A}$ in tuple t_i, and denote by $s_{ij}.\mathbf{A} \in \mathbb{R}$ the realized value of $t_i.\mathbf{A}$ in scenario S_j. Then each expected sum $\mathbb{E}\left(\sum_{i=1}^{N} t_i.\mathbf{A}\ x_i\right) = \sum_{i=1}^{N} \mathbb{E}\left(t_i.\mathbf{A}\right) x_i$ is approximated by $\sum_{i=1}^{N} t_i.\bar{\mu}_{\mathbf{A}}\ x_i$, where $t_i.\bar{\mu}_{\mathbf{A}} = (1/M) \sum_{j=1}^{M} s_{ij}.\mathbf{A}$.

To approximate a probabilistic constraint of the form

$$\Pr\left(\sum_{i=1}^{N} t_i.\mathbf{A}\ x_i \odot v\right) \geq p, \tag{1}$$

we add to the problem a new *indicator variable*, $y_j \in \{0, 1\}$ for each scenario $j \in [1..M]$, along with an associated *indicator constraint*: $y_j = \mathbb{1}\left(\sum_{i=1}^{N} s_{ij}.\mathbf{A}\ x_i \odot v\right)$, where the indicator function $\mathbb{1}(\cdot)$ equals 1 if the inner constraint is satisfied and equals 0 otherwise. We say that solution x "satisfies scenario S_j" (with respect to the constraint) if and only if $y_j = 1$. (Solvers like CPLEX can handle indicator constraints.) Finally, we add the following linear constraint over the indicator variables: $\sum_{j=1}^{M} y_j \geq \lceil pM \rceil$, where $\lceil u \rceil$ is the smallest integer greater than or equal to u. That is, we require that the solution x satisfies at least a fraction p of the M scenarios. The FormulateSAA() function applies these approximations to create the DILP $\text{SAA}_{\mathcal{Q},M}$.

The size of $\text{SAA}_{\mathcal{Q},M}$, measured with respect to the number of coefficients, is $\Theta(NMK)$: we have N coefficients for each expectation constraint and, for each probabilistic constraint, $N + 1$ coefficients (corresponding to $x_1, \ldots, x_N, y_j$) for the jth scenario.

In our implementation, during a precomputation phase, we actually average $\hat{M} \gg M$ scenarios—the same number as the number of validation scenarios—to estimate each $\mathbb{E}\left(t_i.\mathbf{A}\right)$; we then append these estimates, denoted $t_i.\hat{\mu}_{\mathbf{A}}$, to the table. We do this because such averaging is typically very fast to execute, and is space-efficient in that we simply maintain running averages. Thus a solution x returned by a solver is always feasible for every expectation constraint, and hence is feasible overall if and only if, for every probabilistic constraint of the form (1), x satisfies at least a fraction p of the validation scenarios. We can therefore focus attention on the probabilistic constraints, which are the most challenging.

3.2 Out-of-sample validation

After using M scenarios to create and solve the DILP $\text{SAA}_{\mathcal{Q},M}$, we check to see if the solution x is *validation-feasible* in that it is a feasible solution for the DILP $\text{SAA}_{\mathcal{Q},\hat{M}}$

that is constructed using $\hat{M} \gg M$ out-of-sample scenarios. When $\hat{M}$ is sufficiently large, validation feasibility is a proxy for true feasibility, i.e., feasibility for the original SILP; commonly, $\hat{M} = 10^6$ or 10^7. This definition of validation-feasibility is simple, but widely accepted [32]. Although there are other, more sophisticated ways to use validation scenarios to obtain confidence intervals on degree of constraint violation—see, e.g., [10]—these are orthogonal to the scope of this paper. Henceforth, we use the term "feasibility" to refer to "validation feasibility", unless otherwise noted.

The procedure VALIDATE($x, \mathcal{Q}, \hat{M}$) checks the feasibility of x, the solution to $\text{SAA}_{\mathcal{Q},M}$; we describe its operation assuming a single probabilistic constraint $\Pr\left(\sum_{i=1}^{N} t_i.\mathbf{A}\ x_i \odot v\right) \geq p$. It first seeds the system random number generator with a different seed than the one used to generate the optimization scenarios. For each $j \in [1..\hat{M}]$, it generates a realization $\hat{s}_{ij}.\mathbf{A}$ for each $t_i.\mathbf{A}$ such that $x_i > 0$ (i.e., for each tuple that appears in the solution package), and computes the "score" $\sigma_j = \sum_{i:x_i>0} \hat{s}_{ij}.\mathbf{A}\ x_i$. It then sets $y_j = \mathbb{1}(\sigma_j \odot v)$. After all scenarios have been processed, it computes $Y = \sum_{j=1}^{\hat{M}} y_j$ and declares x to be feasible if $Y \geq \lceil p\hat{M} \rceil$. The algorithm purges all realizations from main memory after each scenario has been processed, and only stores the running count of the y_j's, allowing it to scale to an arbitrary number of validation scenarios. Moreover, a package typically contains a realtively small number of tuples, so only a small number of realizations need be generated. A similar procedure can be used to quickly estimate a solution's objective value using limited space: in this case, the procedure simply maintains a running average of the σ_j's.

4 SUMMARY-BASED APPROXIMATION

The NAÏVE algorithm has three major drawbacks. (1) The overall time to derive a feasible solution to $\text{SAA}_{\mathcal{Q},M}$ can be unacceptably long, since the size of $\text{SAA}_{\mathcal{Q},M}$ sharply increases as M increases. (2) It often fails to obtain a feasible solution altogether—in our experiments, the solver (CPLEX) started failing with just a few hundred optimization scenarios. (3) NAÏVE does not offer any guarantees on how close the objective value ω of the solution x to $\text{SAA}_{\mathcal{Q},M}$ is to the true objective value $\hat{\omega}$ of the solution $\hat{x}$ to the DILP $\text{SAA}_{\mathcal{Q},\hat{M}}$ that is based on the validation scenarios. (Recall that we use $\text{SAA}_{\mathcal{Q},\hat{M}}$ as a proxy for the actual SILP.) A feasible solution x that NAÏVE provides can be far from optimal.

Our improved algorithm, SUMMARYSEARCH, which we present in this section, addresses these challenges by ensuring the efficient generation of feasible results through much smaller "reduced" DILPs that each replace a large collection of M scenarios with a very small number Z of scenario "summaries"; in many cases it suffices to take $Z = 1$. We call

Scenario 1

id	...	gain
1	...	0.1
2	...	0.05
3	...	-0.2
4	...	0.2
5	...	0.1
6	...	-0.7

Scenario 3

id	...	gain
1	...	0.01
2	...	0.02
3	...	-0.1
4	...	-0.3
5	...	0.2
6	...	0.3

0.66-Summary

id	...	gain
1	...	0.01
2	...	0.02
3	...	-0.2
4	...	-0.3
5	...	0.1
6	...	-0.7

Figure 3: Using two out of the three scenarios of Figure 2, we derive a 0.66-summary.

such a reduced DILP a *Conservative Summary Approximation* (CSA), in contrast to the much larger sample-average approximation (SAA) used by NAÏVE. The summaries are carefully designed to be more "conservative" than the original scenario sets that they replace: the constraints are harder to satisfy, and thus the solver is induced to produce feasible solutions faster. SUMMARYSEARCH also guarantees that, for any user-specified approximation error $\epsilon \geq \epsilon_{\min}$ (where $\epsilon_{\min}$ is defined in Section 5.4), if the algorithm returns a solution x, then the corresponding objective value ω satisfies $\omega \leq (1+\epsilon)\hat{\omega}$; in this case we say that x is a $(1+\epsilon)$-*approximate* solution. (Recall that we focus on minimization problems with nonnegative objective functions.)

4.1 Conservative Summary Approximation

We first define the concept of an α-summary, and then describe how α-summaries are used to construct a CSA.

Summaries. Recall that a solution x to $\text{SAA}_{\mathcal{Q},M}$ *satisfies* a scenario S_j with respect to a probabilistic constraint of the form of Equation (1) if $y_j = \mathbb{1}\left(\sum_{i=1}^{N} s_{ij}.\mathbf{A}\ x_i \odot v\right) = 1$, where $s_{ij}.\mathbf{A}$ is the realized value of $t_i.\mathbf{A}$ in S_j.

DEFINITION 1 (α-SUMMARY). *Let $\alpha \in [0,1]$. An α-*summary *$S = \{s_i.\mathbf{A} : 1 \leq i \leq N\}$ of a scenario set $\mathcal{S} = \{S_1, \ldots, S_M\}$ with respect to a probabilistic constraint C of the form* (1) *is a collection of N deterministic values of attribute $\mathbf{A}$ such that if a solution x* satisfies *S in that $\sum_{i=1}^{N} s_i.\mathbf{A}\ x_i \odot v$, then x satisfies at least $\lceil \alpha M \rceil$ of the scenarios in $\mathcal{S}$ with respect to C.*

Constructing an α-summary, for $\alpha > 0$, is simple: Suppose that the inner constraint of probabilistic constraint C has the form $\sum_{i=1}^{N} t_i.\mathbf{A}\ x_i \geq v$. Given any subset $G(\alpha) \subseteq \mathcal{S}$ of $\lceil \alpha M \rceil$ scenarios in $\mathcal{S}$, we define S as the tuple-wise minimum over $G(\alpha)$:

$$s_i.\mathbf{A} := \min_{S_j \in G(\alpha)} s_{ij}.\mathbf{A}$$

PROPOSITION 1. *S is an α-summary of $\mathcal{S}$ with respect to C.*

PROOF. Suppose x satisfies S, i.e., $\sum_{i=1}^{N} s_i.\mathbf{A}\ x_i \geq v$. Then for every scenario $S_j \in G(\alpha)$, $\sum_{i=1}^{N} s_{ij}.\mathbf{A}\ x_i \geq \sum_{i=1}^{N} s_i.\mathbf{A}\ x_i \geq v$. Since $|G(\alpha)| = \lceil \alpha M \rceil$, the result follows. □

Figure 3 illustrates an α-summary for the three scenarios in Figure 2, where $\alpha = 0.66$ and $G(\alpha)$ comprises scenarios 1 and 3. The summary is conservative in that, for any choice x of trades, the gain under the summary values will be less than the gain under either of the two scenarios. Thus if we can find a solution that satisfies the summary, it will automatically satisfy at least scenarios 1 and 3. It might also satisfy scenario 2, and possibly many more scenarios, including unseen scenarios in the validation set. Indeed, if we are lucky, and in fact our solution satisfies at least $100p\%$ of the scenarios in the validation set, then x will be feasible with respect to the constraint on **Gain**.

Clearly, for an inner constraint with $\leq$, the tuple-wise *maximum* of $G(\alpha)$ yields an α-summary. While there may be other ways to construct α-summaries, in this paper we only consider minimum and maximum summaries, and defer the study of other, more sophisticated summarization methods to future work. Importantly, a summary need not coincide with any of the scenarios in $\mathcal{S}$; we are exploiting the fact that optimization and validation are decoupled.

CSA formulation. A CSA is basically an SAA in which all probabilistic constraints are approximated using summaries instead of scenarios.[2] The foregoing development implicitly assumed a single summary (with respect to a given probabilistic constraint C) for all of the M scenarios in $\mathcal{S}$. In general, we use Z summaries, where $Z \in [1..M]$. These are obtained by dividing $\mathcal{S}$ randomly into Z disjoint partitions $\Pi_1, \ldots, \Pi_Z$, of approximately M/Z scenarios each. Then the α-summary $S_z = \{s_{iz}.\mathbf{A} : 1 \leq i \leq N\}$ for partition Π_z is obtained by taking a tuple-wise minimum or maximum over scenarios in a subset $G_z(\alpha) \subseteq \Pi_z$, where $|G_z(\alpha)| = \lceil \alpha |\Pi_z| \rceil$.

For each probabilistic constraint C of form (1), we add to the DILP a new indicator variable, $y_z \in \{0, 1\}$, and an associated indicator constraint $y_z := \mathbb{1}\left(\sum_{i=1}^{N} s_{iz}.\mathbf{A}\ x_i \odot v\right)$. We say that solution x "satisfies summary S_z" iff $y_z = 1$. We also add the linear constraint $\sum_{z=1}^{Z} y_z \geq \lceil pZ \rceil$, requiring at least $100p\%$ of the summaries to be satisfied.

We denote the resulting reduced DILP by $\text{CSA}_{\mathcal{Q},M,Z}$. Assuming K probabilistic constraints, the number of coefficients in $\text{CSA}_{\mathcal{Q},M,Z}$ is $\Theta(NZK)$, which is independent of M. Usually, Z takes on only small values, so that the effective size complexity is only $\Theta(NK)$. Our results (Section 6) show that in most cases SummarySearch finds good solutions with only one summary, i.e., $Z = 1$. Because Z is small, the solution to $\text{CSA}_{\mathcal{Q},M,Z}$ can be rapidly computed by a solver. The CSA formulation is also more robust to random fluctuations in the sampled data values, and less prone to "overfit" to an unrepresentative set of scenarios obtained by luck of the draw.

[2] As with the SAA formulation, expectations are approximated as averages over a huge number $\hat{M}$ of independent scenarios.

Algorithm 2 SummarySearch Query Evaluation

$\mathcal{Q}$: A stochastic package query with K probabilistic constraints
$\hat{M}$: Number of out-of-sample validation scenarios (e.g., 10^6)
M : Initial number of optimization scenarios (e.g., 100)
m : Iterative increment to M (e.g., 100)
z : Iterative increment to Z (e.g., 1)
ϵ : User-defined approximation error bound, $\epsilon \geq \epsilon_{\min}$
output: A feasible package solution x, or failure (no solution).

```
▷ Solve probabilistically-unconstrained problem
x^(0) ← Solve(SAA(Q_0, M̂))
Z = 1                                              ▷ Initial number of summaries
repeat
    (x, v̂_x) ← CSA-Solve(Q, x^(0), M, Z)
    if v̂_x.is_feasible then                        ▷ x is feasible
        if v̂_x.upper_bound ≤ ε then                ▷ x is (1 + ε)-approximate
            return x
        else if Z < M then
            z' ← min{z, M − z}
            Z ← Z + z'                             ▷ More summaries to improve optimality
    else
        M ← M + m                                  ▷ More scenarios to improve feasibility
```

An important observation is that as Z increases, $\text{CSA}_{\mathcal{Q},M,Z}$ approaches the $\text{SAA}_{\mathcal{Q},M}$ formulation: at $Z = M$ each partition will contain exactly one scenario, which will also coincide with the summary for the partition. Since $\text{CSA}_{\mathcal{Q},M,Z}$ encompasses $\text{SAA}_{\mathcal{Q},M}$, we can always do at least as well as Naïve with respect to the feasibility and optimality properties of our solution, given M scenarios. We address the issue of how to choose Z, α, and each $G_z(\alpha)$ below and in Section 5, and also discuss how to generate summaries efficiently.

4.2 Query Evaluation with CSA

Algorithm 2 shows query evaluation with SummarySearch. The goal is to find a feasible solution whose objective value is as close as possible to $\hat{\omega}$, the objective value of the SAA based on the $\hat{M}$ validation scenarios. In the algorithm, $\mathcal{Q}_0$ denotes the SPQ obtained from $\mathcal{Q}$ by removing all of the probabilistic constraints. At the first step, SummarySearch computes $x^{(0)}$, the solution to the DILP $\text{SAA}_{\mathcal{Q}_0,\hat{M}}$; the only constraints are deterministic constraints and expectation constraints, with the latter estimated from $\hat{M}$ scenarios in the usual way. This corresponds to the "least conservative" solution possible, and is effectively equivalent to solving a CSA using summaries constructed with $\alpha = 0$, because 0% (i.e., none) of the scenarios are required to be satisfied. For some problems, $x^{(0)}$ might have an infinite objective value, in which case we simply ignore this solution and incrementally increase α until we find a finite solution.

Like Naïve, the SummarySearch algorithm starts with an initial number of optimization scenarios, $M \geq 1$, and iteratively increments it while solutions are infeasible. In the

optimization phase, the algorithm uses a CSA formulation, which replaces the M real scenarios with Z conservative summaries. Initially, the algorithm uses $Z = 1$, replacing the set of M scenarios with a single summary. After feasibility is achieved for a solution x with objective value ω_x, the algorithm tries to check whether the ratio $\epsilon_x = (\omega_x - \hat{\omega})/\hat{\omega}$ is less than or equal to the user-defined error bound ϵ; although $\hat{\omega}$, and hence ϵ_x, is unknown, we can conservatively check whether $\epsilon'_x \leq \epsilon$, where ϵ'_x is an upper bound on ϵ_x that we develop in Section 5.4, If the solution is unsatisfactory, SummarySearch increases Z, and iterates again. The algorithm stops if and when a feasible and $(1+\epsilon)$-approximate solution is found. In practice, because of the conservative nature of summaries, SummarySearch typically finds feasible solutions in drastically fewer iterations than Naïve.

5 OPTIMAL SUMMARY SELECTION

The key component of SummarySearch is CSA-Solve, described in this section. With M and Z fixed, CSA-Solve finds the best CSA formulation, i.e., the one having, for each constraint, the optimal value of α and the best set $G_z(\alpha)$ of scenarios for each summary. CSA-Solve thus determines the best solution x achievable with M scenarios and Z summaries, and also computes metadata $\hat{v}_x$ used by SummarySearch for checking feasibility and optimality.

5.1 CSA-Solve Overview

Algorithm 3 depicts the iterative process of CSA-Solve: at each iteration q it produces a solution $x^{(q)}$ to a problem $\text{CSA}_{\text{Q},M,Z}$ based on an $\alpha_k^{(q)}$-summary for each constraint C_k. Initially, $\alpha_k^{(0)} = 0$ for all k, and thus the solution to $\text{CSA}_{\text{Q},M,Z}$ is simply $x^{(0)}$, which has already been computed by SummarySearch prior to calling CSA-Solve. Then CSA-Solve stops in two cases: (1) if it finds a feasible $(1+\epsilon)$-approximate solution; (2) if it enters a cycle, producing the same solution twice with the same α_k values. In case (2), it returns the "best" solution found so far: if one or more feasible solutions have been found, it returns the one with the best objective value, otherwise it returns an infeasible solution, and SummarySearch will increase M in its next iteration.

5.2 Choosing α

Larger α leads to more conservative α-summaries, as we take the tuple-wise minimum (or maximum) over more and more scenarios. Thus a high value of α increases the chances of finding a feasible solution. On the other hand, if the constraints are more restrictive than necessary, then the solution can have a seriously suboptimal objective value because we are considering fewer candidate solutions, possibly missing the best ones. Thus, CSA-Solve seeks the minimally conservative value of α that will suffice.

How can we measure the true conservativeness of α with respect to a constraint $C \coloneqq \Pr(\sum_{i=1}^{N} t_i.\boldsymbol{A}\, x_i \odot v) \geq p$? As discussed previously, the solution x to a formulation $\text{SAA}_{\text{Q},M}$ based on α-summaries is guaranteed to satisfy at least $100\alpha\%$ of the M optimization scenarios, but the actual true probability of satisfying the constraint—or more pragmatically, the fraction of the $\hat{M}$ validation scenarios satisfied by x—will usually differ from α. Thus, we look at the difference between the fraction of validation scenarios satisfied by x and the target value p. We call this difference the *p-surplus*, and define it as:

$$r = r(\alpha) \coloneqq \left\{ (1/\hat{M}) \sum_{j=1}^{\hat{M}} \mathbb{1}\Big(\sum_{i=1}^{N} \hat{s}_{ij}.\boldsymbol{A}\, x_i \odot v \Big) \right\} - p$$

We expect the function $r(\alpha)$ to be increasing in α with high probability.

Observe that x essentially satisfies the constraint $C' \coloneqq \Pr(\sum_{i=1}^{N} t_i.\mathbf{A}\, x_i \odot v) \geq p + r$. Clearly, if $r < 0$, then x is infeasible for constraint C, whereas if $r > 0$, then x satisfies the inner constraint with a probability that exceeds p, and so is conservative and therefore likely suboptimal. Thus the optimal value α^* satisfies $r(\alpha^*) = 0$. Solutions that achieve zero p-surplus may be impossible to find, and therefore CSA-Solve tries to choose $\alpha = (\alpha_1, \ldots, \alpha_K)$ to minimize the p-surplus for each of the K constraints, while keeping it nonnegative. The search space is finite (hence the possibility of cycles) since $\alpha_k \in \{Z/M, 2Z/M, \ldots, 1\}$ for $k \in [1..K]$.

At each iteration q, CSA-Solve updates $\alpha^{(q-1)}$ to $\alpha^{(q)}$, creates the corresponding CSA problem, and produces a new solution $x^{(q)}$. For simplicity and ease of computation, our initial implementation updates each $\alpha_k^{(q)}$ individually by fitting a smooth curve $R_k^{(q)}(\alpha_k)$ to the historical points $(\alpha_k^{(0)}, r_k^{(0)}), \ldots, (\alpha_k^{(q-1)}, r_k^{(q-1)})$ and then solving the equation $R_k^{(q)}(\alpha_k) = 0$. In our experiments, we observed that (1) fitting an arctangent function provides the most accurate predictions and (2) this artificial decoupling with respect to the constraints yields effective summaries; we plan to investigate other methods for jointly updating $(\alpha_1^{(q-1)}, \ldots, \alpha_K^{(q-1)})$.

5.3 Choosing G_z

So far, we have assumed that the subset $G_z(\alpha_k^{(q)})$ used to build the summary is *any* set containing $n_k^{(q)} = \lceil \alpha_k^{(q)} |\Pi_z| \rceil$ scenarios. SummarySearch employs a simple greedy heuristic to determine $G_z(\alpha_k^{(q)})$: it chooses the $n_k^{(q)}$ scenarios that

Algorithm 3 CSA-Solve

$\mathcal{Q}$: A stochastic package query with K probabilistic constraints
$x^{(0)}$: Solution of probabilistically-unconstrained problem
M : Number of optimization scenarios
Z : Number of summaries, $1 \le Z \le M$
ϵ : User-defined approximation error bound, $\epsilon \ge \epsilon_{\min}$
output: A feasible and $(1+\epsilon)$-approximate solution, or an infeasible solution

1: $q \leftarrow 0$ ▷*Iteration count*
2: $\mathcal{H} \leftarrow \emptyset$ ▷*Initialize validation history*
3: $\alpha^{(q)} = (\alpha_1^{(q)}, \ldots, \alpha_K^{(q)}) \leftarrow (0, \ldots, 0)$ ▷*Initial conservativeness*
4: **repeat**
5: ▷ *If entered a cycle, return best solution from history*
6: **if** $(x^{(q)}, \alpha^{(q)}) \in \mathcal{H}$ **then**
7: **return** Best($\{x : (x, \alpha) \in \mathcal{H}\}$)
8: $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x^{(q)}, \alpha^{(q)})\}$ ▷*Update validation history*
9: $\hat{v}^{(q)} \leftarrow$ Validate($x^{(q)}, \mathcal{Q}, \hat{M}$) ▷*Validate & compute metadata*
10: $\epsilon^{(q)} \leftarrow \hat{v}^{(q)}$.upper_bound ▷*Validation upper bound on ϵ*
11: **for** $k = 1, \ldots, K$ **do**
12: $r_k^{(q)} \leftarrow \hat{v}_k^{(q)}$.surplus ▷*Validation p-surplus*
13: ▷ *Termination with feasible $(1+\epsilon)$-approximate solution*
14: **if** $\epsilon^{(q)} \le \epsilon$ **and** $\forall k : r_k^{(q)} \ge 0$ **then**
15: **return** $(x^{(q)}, \hat{v}^{(q)})$
16: $q \leftarrow q + 1$ ▷*Iterate again with a new set of summaries*
17: $\alpha^{(q)} \leftarrow$ GuessOptimalConservativeness($\mathcal{H}$)
18: **for** $k = 1, \ldots, K$ **do**
19: $\tilde{S}_k \leftarrow$ Summarize($x^{(q)}, \alpha_k^{(q)}, C_k, \mathcal{H}$)
20: $\text{CSA}_{\mathcal{Q},M,Z} \leftarrow$ FormulateSAA($\mathcal{Q}, \{\tilde{S}_1, \ldots, \tilde{S}_K\}$)
21: $x^{(q)} \leftarrow$ Solve($\text{CSA}_{\mathcal{Q},M,Z}$)

produce the summary most likely to keep the previous solution feasible in the current iteration, so that the new solution will likely have a higher objective value. For an inner $\ge$ ($\le$) constraint, this is achieved by sorting the scenarios in Π_z according to their "scenario score" $\sum_{i=1}^{N} s_{ij}.\mathbf{A}\, x_i^{(q-1)}$ and taking the first $n_k^{(q)}$ in descending (ascending) order.

5.4 Approximation Guarantees

If $x^{(q)}$ is feasible, SummarySearch can terminate if it can determine that $x^{(q)}$ is $(1+\epsilon)$-approximate relative to the optimal feasible solution $\hat{x}$ based on the validation scenarios, i.e., that $\omega^{(q)} \le (1+\epsilon)\hat{\omega}$, where $\omega^{(q)}$ and $\hat{\omega}$ are the objective values for $x^{(q)}$ and $\hat{x}$, respectively, and ϵ is an accuracy parameter specified by the user. Without loss of generality, we assume below that the objective function is an expectation; should the objective be deterministic, nesting it within an expectation does not change its value.

This termination check proceeds as follows. During the qth iteration of SummarySearch, the function Validate($x^{(q)}, \mathcal{Q}, \hat{M}$) computes p-surplus values $r_1^{(q)}, \ldots, r_K^{(q)}$, one for each probabilistic constraint in the query. Further, it computes $\epsilon^{(q)}$ (as defined below). We show below that if $\epsilon^{(q)} \le \epsilon$ and $\forall k : r_k^{(q)} \ge 0$, then $x^{(q)}$ is a feasible $(1+\epsilon)$-approximate solution, and SummarySearch can immediately return $x^{(q)}$ and terminate. As usual, we focus on minimization problems with nonnegative objective values, and take the optimal solution $\hat{x}$ and objective value $\hat{\omega}$ of $\text{SAA}_{\mathcal{Q},\hat{M}}$ as proxies for those of the original SILP. We start with the following simple but important result.

Proposition 2 (General Approximation Guarantee). *Let $\epsilon \ge 0$ and let $\underline{\omega}$ be a positive constant such that $\underline{\omega} \le \hat{\omega}$. Set $\epsilon^{(q)} = (\omega^{(q)}/\underline{\omega}) - 1$. If $\epsilon^{(q)} \le \epsilon$, then $\omega^{(q)} \le (1+\epsilon)\hat{\omega}$.*

Proof. Suppose that $\epsilon^{(q)} \le \epsilon$. Since $\hat{\omega}/\underline{\omega} \ge 1$, we have

$$\omega^{(q)} \le \left(\frac{\hat{\omega}}{\underline{\omega}}\right)\omega^{(q)} = \left(1+\left(\frac{\omega^{(q)}}{\underline{\omega}}-1\right)\right)\hat{\omega} = \left(1+\epsilon^{(q)}\right)\hat{\omega} \le (1+\epsilon)\hat{\omega},$$

and the result follows. □

We obtain a specific formula for $\epsilon^{(q)}$ by choosing a specific bound $\underline{\omega}$. Clearly, we would like to choose $\underline{\omega}$ as large as possible, since this maximizes the likelihood that $\epsilon^{(q)} \le \epsilon$. One simple choice that always works is to set $\underline{\omega} = \omega^{(0)}$, where $\omega^{(0)}$ is the objective value of the SAA problem corresponding to the original SILP but with all probabilistic constraints removed—see line 2 of Algorithm 2. If all random variables are lower-bounded by a constant $\underline{s} > 0$ and the size of any feasible package is lower-bounded by a constant $\underline{l} > 0$, then $\sum_{i=1}^{N} \hat{s}_{ij}.\mathbf{A}\, x_i \ge \underline{s}\underline{l}$, $\forall j \in [1..\hat{M}]$, so that

$$\hat{\omega} = \frac{1}{\hat{M}}\sum_{j=1}^{\hat{M}}\sum_{i=1}^{N}\hat{s}_{ij}.\mathbf{A}\,\hat{x}_i \ge \frac{1}{\hat{M}}\sum_{j=1}^{\hat{M}}\underline{s}\underline{l} = \underline{s}\underline{l},$$

which yields an alternative lower bound. Yet another bound can be sometimes obtained by exploiting the relation of the constraints to the objective.

Definition 2 (Objective-Constraint Interaction). *Let the objective be* min $\mathbb{E}(\sum_{i=1}^{N}\xi_i x_i)$*, for random variables* $\{\xi_i\}_{i\in[1..N]}$*. The objective is said to be* supported *by a constraint of the form* $\Pr\left(\sum_{i=1}^{N}\xi_i x_i \le v\right) \ge p$ *and* counteracted *by a constraint of the form* $\Pr(\sum_{i=1}^{N}\xi_i x_i \ge v) \ge p$*. All other forms of constraint are said to be* independent *of the objective.*

Intuitively, a supporting probabilistic constraint "supports" the objective function in the same "direction" of the optimization ($\le$ for minimization, $\ge$ for maximization), whereas a counteracting constraint goes against the optimization. If there exists a counteracting constraint with $v \ge 0$, it can be shown (see the online appendix [5]) that $\hat{\omega} \ge pv$.

Finally, we take $\underline{\omega}$ to be the maximum of all applicable lower bounds. Similar formulas can be derived for other possible cases—maximization problems, negative objective values, and so on; see the online appendix [5].

Note that if $(\hat{\omega}/\underline{\omega}) - 1 > \epsilon$, then $\epsilon^{(q)} = (\omega^{(q)}/\underline{\omega}) - 1 > \epsilon$, $\forall q \geq 0$, so that SummarySearch cannot terminate with a feasible $(1+\epsilon)$-approximate solution. To avoid this problem, we require that $\epsilon \geq \epsilon_{\min}$, where $\epsilon_{\min} = (\overline{\omega}/\underline{\omega}) - 1$. Here $\overline{\omega}$ is any upper bound on $\hat{\omega}$. It can be shown, for example, that if (1) all random variables are upper-bounded by a constant $\overline{s} > 0$, (2) the size of any feasible package is upper-bounded by a constant $\overline{l} > 0$, and (3) there exists a supporting constraint with $v \geq 0$, then $\hat{\omega} \leq v + (1-p)\overline{s}\overline{l}$; see the online appendix [5]. If we have available a feasible solution x with objective value ω_x, then we can take $\overline{\omega} = \omega_x$. We choose $\overline{\omega}$ to be the minimum of all applicable bounds.

5.5 Implementation Considerations

We now discuss several implementation optimizations.

Efficient summary generation. Recall that summarization has two steps: (1) computing the scenario scores to sort scenarios by the previous solution, and (2) computing the tuple-wise minimum (or maximum) of the first α% of the scenarios in sorted order. The fastest way to generate an α-summary is if all M scenarios are generated and kept in main memory at all times. In this case, computing the tuple-wise minimum (or maximum) is trivial. However, the $\Theta(MNK)$ memory requirement for this may exceed the memory limits if M is large. We devise two possible strategies for memory-efficient summary generation with optimal $\Theta(NZK)$ space complexity: *tuple-wise summarization* and *scenario-wise summarization*. Tuple-wise summarization uses a unique random number seed for each tuple ($i = 1, \ldots, N$) and it generates all M realizations, one tuple at a time. Scenario-wise summarization uses a unique seed for each scenario ($j = 1, \ldots, M$), and it generates one realization for all tuples, one scenario at a time.

With tuple-wise summarization, sorting the scenario only requires $\Theta(PM)$ time, where $P = \sum_{i=1}^{N} x_i$ is the size of the current package; usually, $P \ll N$. However, generating the summaries is more costly, as it requires $\Theta(NM)$ time, as all M realizations must be constructed for all N tuples. The total time is $\Theta(M(P+N))$. With scenario-wise summarization, generating summaries has lower time complexity of $\Theta(\alpha NM)$, as it only generates scenarios in $G_z(\alpha)$, but sorting has higher complexity $\Theta(NM)$, with total time $\Theta(NM(\alpha+1))$.

It follows that if $\alpha \geq P/N$, tuple-wise summarization is generally faster than scenario-wise summarization. However, other factors may affect the runtime, e.g., some random number generators, such as Numpy, generate large quantities of random numbers faster if generated in bulk using a single seed. In this case, tuple-wise summarization may suffer considerably in the summary generation phase, as it needs to re-seed the random number generator for each tuple. In our experiments, we observed that tuple-wise summarization is better when the input table is relatively small, but worse than scenario-wise for larger tables. In general, a system should implement both methods and test the two in situ.

Convergence acceleration. When $\alpha_k^{(q)}$ is obtained by *decreasing* $\alpha_k^{(q-1)}$, the solution $x^{(q-1)}$ typically is feasible, and our goal is for $x^{(q)}$ to strictly improve in objective value. CSA-Solve achieves this by slightly modifying the generation of summaries in order to ensure that the previous solution is still feasible for the next CSA problem. This is done by using the tuple-wise maximum (instead of minimum) in the summary generation for all tuples t_i such that $x_i^{(q-1)} > 0$ (tuples in the previous solution). For all other tuples, we set the summary as usual. We have found that ensuring monotonicity of the objective values promotes faster convergence.

6 EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of our techniques for stochastic package queries on three different domains where uncertainty naturally arises: noise in sensor data, uncertainty in future predictions, uncertainty due to data integration [18]. Our results show that: (1) SummarySearch is always able to find feasible solutions, while Naïve cannot in most cases—when both SummarySearch and Naïve can find feasible solutions, SummarySearch is often faster by orders of magnitude; (2) The packages produced by SummarySearch are of high quality (low empirical approximation ratio), sometimes even better than Naïve when they both produce feasible solutions; (3) Increasing M, the number of optimization scenarios, helps SummarySearch find feasible solutions, and the value of M required by SummarySearch to start producing feasible solutions is much smaller than Naïve, explaining the orders of magnitude improvement in running time; (4) Increasing Z, the number of summaries, helps SummarySearch find higher-quality solutions; (5) Increasing N, the number of input tuples, impacts the running time of both algorithms, but SummarySearch is still orders of magnitude faster than Naïve, and finds feasible solutions with better empirical approximation ratios than Naïve.

6.1 Experimental Setup

We now describe the software and runtime environment, and the three workloads we used in the experiments.

Environment. We implemented our methods in Python 2.7, used Postgres 9.3.9 as the underlining DBMS, and IBM CPLEX 12.6 as the ILP solver. We ran our experiments on servers equipped with two 24 2.66GHz cores, 15GB or RAM, and a 7200 RPM 500GB hard drive.

Datasets and queries. We constructed three workloads:

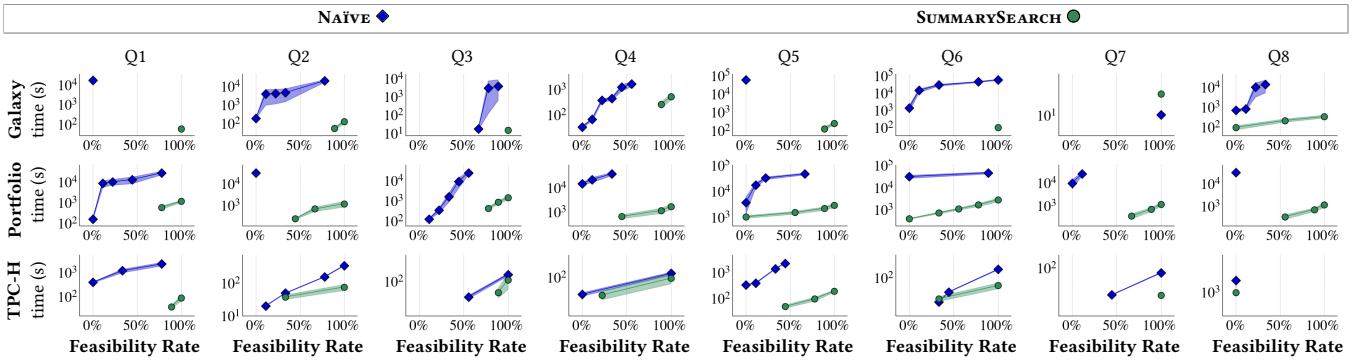

Figure 4: End-to-end results of SummarySearch vs. Naïve. Plotting the average time (and 95% confidence intervals) to reach 100% feasibility rate. Of the 23 feasible queries (TPC-H Q8 is infeasible), SummarySearch always reaches 100% feasibility rate, while Naïve in only 7 queries. In 15 queries, when SummarySearch succeeds, Naïve is still at 0% feasibility. SummarySearch can be orders of magnitude faster even when both reach 100% feasibility.

Noisy sensor measurements: The Galaxy datasets vary between 55,000 and 274,000 tuples, extracted from the Sloan Digital Sky Survey (SDSS) [41]. Each tuple contains the color components of a small portion of the sky as read by a telescope. We model the uncertainty in the telescope readings as Gaussian or Pareto noise.

Financial predictions: The Portfolio dataset contains 6,895 stocks downloaded from Yahoo Finance [43]. The initial price of each stock is set according to its actual value on January 2, 2018, and future prices are generated according to a geometric Brownian motion. We consider selling stocks in one day or in one week, as in Figure 1; the dataset for the short-term (resp., long-term) trades contains 14,000 (resp., 48,000) tuples. For each prediction type, we also extracted a subset corresponding to the 30% most volatile stocks to construct some of the hardest queries. Tuples referring to the same stock are correlated to one another. For example, in Figure 1, tuples 1 and 2 are correlated to each other but are independent of the other tuples.

Data integration: The TPC-H dataset consists of about 117,600 tuples extracted from the TPC-H benchmark [42]. We simulate the result of hypothetically integrating several data sources to form this data set: we model uncertainty in each attribute's value with discrete probability distributions. For each original (deterministic) value in the TPC-H dataset, we generate D possible variations thereof, where D is the number of data sources that have been integrated into one. The mean of these D values is anchored around the original value; each source value is sampled from an exponential, Poisson, uniform or Student's t-distribution.

For each of the three datasets, we constructed a workload of eight sPaQL queries; all 24 queries, except one in TPC-H, are feasible. The workloads span seven different distributions for the uncertain data attributes, including a complex VG function to predict future stock prices. The objective functions are supported by the constraints for the Portfolio queries, independent for the TPC-H queries and either supported or counteracted for the Galaxy queries (see Definition 2 for supported/counteracted/independent objectives). The Portfolio workload tests high- and low-risk, high- and low-VaR (Value at Risk)—i.e., p and v in Equation (1)—as well as short- and long-term trade predictions. The TPC-H workload is split into queries with $D = 3$ and $D = 10$ (number of integrated sources). For all queries there are two constraints, one of which is probabilistic with $p \geq 0.9$. Examples include: (1) for Galaxy, we seek a set of five to ten sky regions that maximizes total expected radiation flux while avoiding total flux levels less than 40 with high probability, and (2) for TPC-H, we seek a set of between one and ten transactions having maximum expected total revenue, while containing less than 15 items total with high probability. A detailed description of the workloads can be found in the online appendix [5].

Evaluation metrics. We measure *response time* (in seconds and logarithmic scale) across 10 i.i.d runs using different seeds for generating the optimization scenarios, and evaluate feasibility and the objective value on an out-of-sample validation set with 10^6 scenarios (10^7 for the Portfolio workload). We plot the average across the 10 runs, and its 95% confidence interval in a shaded area. For each run of an algorithm, we set a time limit of four hours. When the time limit expires, we interrupt CPLEX and get the best solution found by the solver until then. We measure *feasibility rate* as the fraction, out of the 10 runs, in which a method produces a feasible solution (including, for all methods, when the time limit expired). Because the true optimal solution for any of the queries is unknown, we measure *accuracy* by $1 + \hat{\epsilon}$, where $\hat{\epsilon} := \omega/\omega^* - 1$ and ω^* is the objective value of the best feasible solution found by any of the methods.

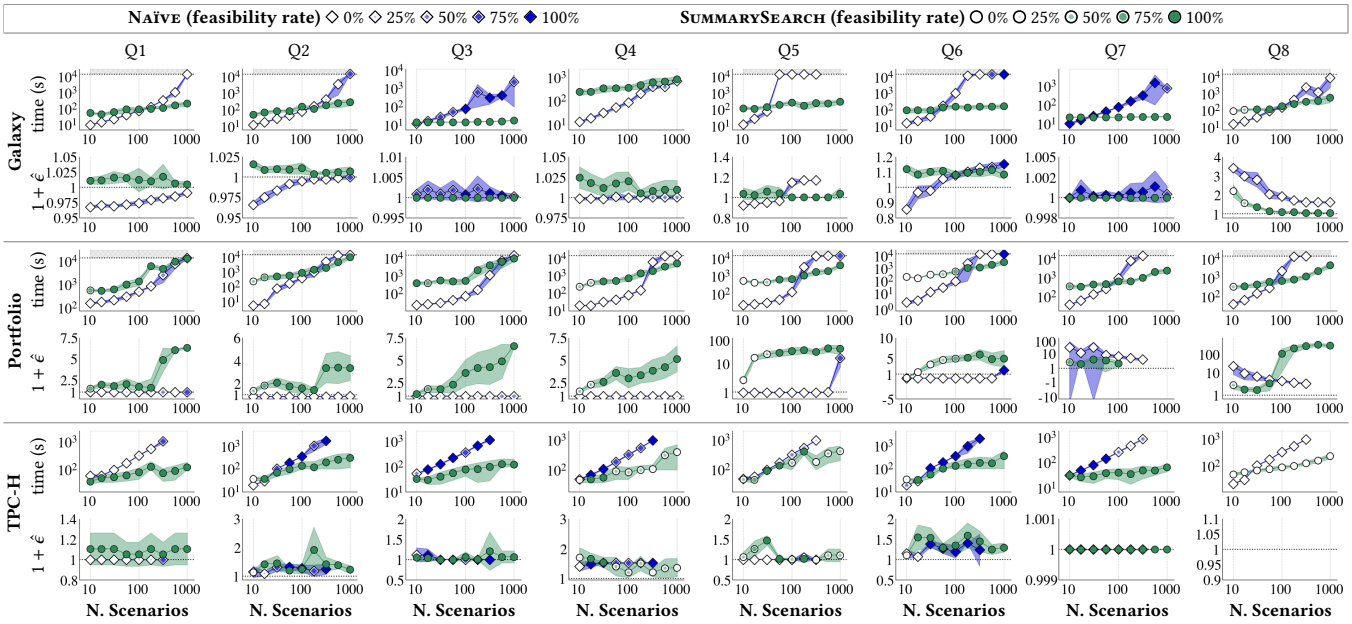

Figure 5: Scalability of Naïve and SummarySearch with increasing number of optimization scenarios. Naïve struggles to find feasible solutions even with a large number of scenarios and often fails completely (missing points in the plot). SummarySearch quickly finds feasible solutions with few scenarios. The approximation ratios of SummarySearch's solutions are generally low when the number of scenarios is small.

6.2 Results and Discussion

We evaluate four fundamental aspects of our algorithms: (1) query response time to reach 100% feasibility rate; (2) scalability with increasing number of scenarios (M); (3) scalability of SummarySearch with increasing number of summaries (Z); (4) scalability with increasing dataset size (N).

6.2.1 Response time to reach 100% *feasibility rate.* Both Naïve and SummarySearch increase M (the number of scenarios) up to when solutions start to be feasible. We report the cumulative time for all iterations the algorithm took to reach a certain feasibility rate, from 0%, up to 100% (when the algorithm produces feasible solutions for all 10 runs). For SummarySearch, Z is fixed (1 for Galaxy and Portfolio, 2 for TPC-H). We set Z to the lowest value (per workload) such that SummarySearch could reach 100% feasibility rate. Figure 4 shows the results of the experiment. For all (23) feasible queries across all workloads, SummarySearch is always able to reach 100% feasibility rate, while Naïve can only reach 100% feasibility for only 7 queries. Even then, SummarySearch is usually orders of magnitude faster than Naïve (e.g., Galaxy Q6, TPC-H Q2, Q6, and Q7). Moreover, in 15 out of the 23 feasible queries, SummarySearch reached 100% feasibility while Naïve was still at 0%. The conservative nature of summaries allows higher feasibility rates for SummarySearch even with fewer scenarios. As the number of scenarios increases, SummarySearch solves a much smaller problem than Naïve, leading to orders-of-magnitude faster response time.

The only case where SummarySearch is slower than Naïve at reaching 100% feasibility rate is Galaxy Q7, which was an easy query for both methods: both solved it with only 10 scenarios. This query has a supported objective function over data with minimal uncertainty described by a Pareto distribution with "scale" and "shape" both equal to 1. For this query, the summarization process and solving a probabilistically-unconstrained problem are overheads for SummarySearch. TPC-H Q8 is an infeasible query. Both methods increase M up to 1000 before declaring infeasibility, but again SummarySearch is faster than Naïve in doing so.

6.2.2 Effect of increasing the number of optimization scenarios. We evaluate the scalability of our methods when the number of optimization scenarios M increases; Z is fixed as described above. For each algorithm, we group feasibility rates into 5 groups: 0%, 25%, 50%, 75% and 100%, and use different shadings to distinguish each case.

Figure 5 gives scalability results for the three workloads. Generally, with low M, Naïve executes very quickly to produce infeasible solutions with low objective values (optimizer's curse); as Naïve increases M, the running time increases exponentially—note the logarithmic scale—up to a

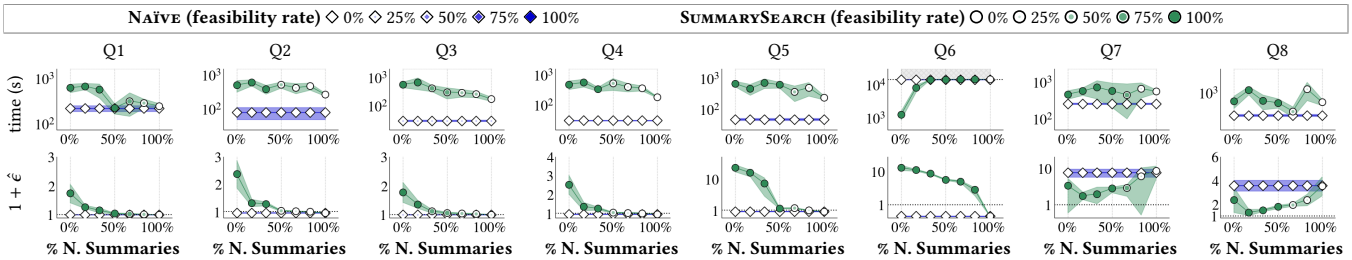

Figure 6: Effects of increasing number of summaries (Z) on the Portfolio workload, as a percentage of the number of scenarios, from 1 summary up to M summaries (100%). Increasing Z improves the approximation ratio of the solution produced by SummarySearch. Increasing Z too far results in infeasible solutions as, when $Z = M$, SummarySearch is identical to Naïve, and it thus overfits, like Naïve, to a bad set of scenarios.

point where it fails altogether (missing Naïve points in the plots). On the other hand, SummarySearch finds feasible solutions even with as little as 10 scenarios.

SummarySearch produces high quality solutions as demonstrated by the low approximation ratio ($1 + \hat{\epsilon}$), close to 1 for most queries. However, with the hardest Portfolio queries (Q5 and Q6), the worst approximation ratio for SummarySearch is quite high for feasible solutions: this is an indicator that the number of summaries, $Z = 1$ is too low and should be increased.

6.2.3 Effect of increasing the number of summaries. In this experiment, we show how increasing the number of summaries (Z) helps improve the approximation ratio in the Portfolio queries. We increase Z from 1 up to M (number of scenarios), where M is set to where the feasibility rate of SummarySearch was 100% in the previous experiment, and we show the running time and approximation ratio compared to Naïve with M scenarios. Figure 6 shows the results of this experiment. First, the response time with increasing Z is in most cases independent of Z. In fact, while increasing Z adds more scenarios to the CSA formulation, each summary becomes less and less conservative, making the problem a bit larger but always easier; in the limit ($Z = M$), each summary is identical to an original scenario, and thus SummarySearch only pays the extra overhead, compared to Naïve, of solving the probabilistically-unconstrained problem first. On the other hand, Naïve is always faster, but its solutions are infeasible. For most queries, the approximation ratio closely approaches 1, while still maintaining a high feasibility rate. Increasing Z too far eventually causes feasibility to drop, reaching that of Naïve in the limit ($Z = M$).

Finally, even though infeasible solutions tend to have better objective values than feasible ones, we find that Naïve's infeasible solutions to Q7 and Q8 have worse objective values. These queries proved quite challenging for Naïve as they involved stock price predictions for a week in the future.

6.2.4 Effect of increasing the dataset size. In this experiment, we increase the Galaxy dataset up to five times from 55,000 tuples to 274,000 tuples. For all queries except Q8 we fix $M = 56$ (for both algorithms) and $Z = 1$. In general, SummarySearch scales well with increasing data set size: it finds feasible solutions with good approximation ratios. Naïve, however, times out for several queries (Q1, Q2, Q5, Q6, & Q8) and its response time sharply increases as dataset size increases (Q1, Q2, Q6, Q8). Except for three queries (Q3, Q4, Q7), most of Naïve's solutions are infeasible; even then, SummarySearch produces feasible solutions in orders of magnitude less time with better approximation ratios.

In Q8, we set $M = 562$ to enable SummarySearch to still produce feasible solutions (75% feasibility at 274K tuples), without causing Naïve to fail. Q8 is a challenging query as each data value is sampled from a Pareto distribution with different parameters leading to high variability across scenarios.

To further increase the data size scalability of SummarySearch, we hope to combine it with partitioning and divide-and-conquer approaches similar to SketchRefine [3].

7 RELATED WORK

Probabilistic databases and package queries. *Probabilistic databases* [14, 40] have focused mainly on modeling discrete data uncertainty; the *Monte Carlo Database* (MCDB) [26] supports arbitrary uncertainty, via VG functions. Probabilistic databases support SQL queries, but lack support for optimization. *Package query engines* [3, 38] offer support only for deterministic optimization.

Stochastic optimization. *Stochastic optimization* [23] studies approximations for stochastic constraints and objectives. Probabilistic constraints are very hard to handle in general, because the feasible region of the inner constraint may be non-convex [1, 8, 11, 17, 23, 33, 35]. In this work, we study stochastic optimization problems with objective functions

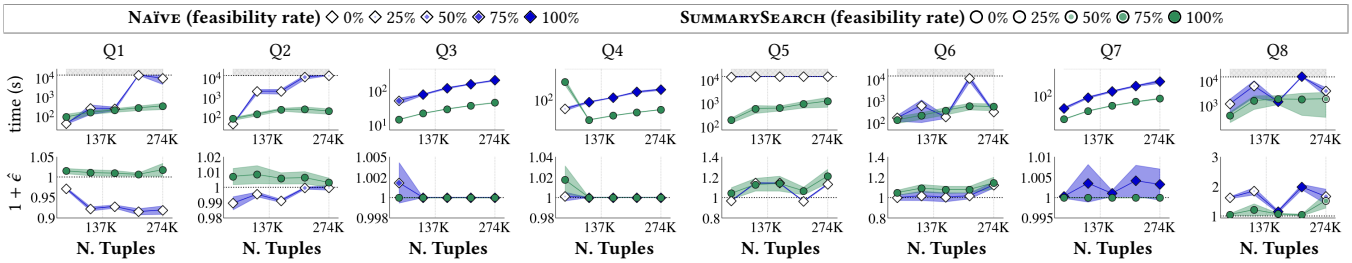

Figure 7: Scalability of Naïve and SummarySearch with increasing dataset size (N) on the Galaxy workload. The running times of both algorithms degrades with increasing N, but SummarySearch scales up well in comparison with Naïve.

and constraints defined in terms of linear functions of the tuple attributes.

Our Naïve method is derived from the numerous "scenario approximations" from the SP literature [8, 9, 11, 23, 29, 32, 34, 35]. Choosing the number of scenarios (M) a priori is one of the most studied problems. Campi et al. [11] show that the optimal solution of a Monte Carlo formulation that satisfies exactly M iid scenarios is feasible with probability at least δ if $M \geq \frac{2}{1-p_j}\left(\ln\left(\frac{1}{1-\delta}\right) + N\right)$. A-priori bounds quickly become impractical in a database setting, where N is also the number of tuples, and thus typically large. For example, with a table of size $N = 50{,}000$, $p_j = 0.9$, $\delta = 0.95$, at least $M \geq 1{,}000{,}060$ scenarios must be generated and all satisfied.

Scenario removal studies techniques for removing scenarios after sampling [7, 9, 20, 30, 32]. Empirically, these methods generally provide a reduction factor of only 50% or less, which is insufficient for our setting. Our α-summary can be viewed as removing $100(1-\alpha)\%$ of the scenarios, where α is usually very small (below 0.01); not only do we remove scenarios, but we replace them with conservative summaries.

Similar to our setting, *distributionally robust optimization* (DRO) [16, 22, 31] attempts to mitigate the optimizer's curse when the uncertainty distribution is unknown but is assumed to lie in some set of candidate distributions; the original probability constraints are replaced with worst-case probability constraints based on this set. In contrast, SummarySearch uses deterministic worst-case constraints, which are simpler and avoid assumptions on the uncertainty distribution. DRO methods also show limited scalability in the number of variables N, e.g., N is at most 20 in the experiments in [31].

The goal of *wait-and-judge optimization* [10, 12] is to perform a-posteriori feasibility analysis. Existing approaches help provide bounds on the quality of a solution, but do not provide algorithms that dynamically adapt in response to poor solutions. SummarySearch, instead, adjusts the conservativeness of the summaries to obtain feasible solutions with minimum computational cost. SummarySearch can potentially use wait-and-judge during out-of-sample validation to decide when to stop increasing the number of scenarios.

8 CONCLUSION AND FUTURE WORK

In this paper, we addressed *single-stage* decision making under uncertainty, in which decisions are made before the values of the random variables become known. In many cases, however, uncertainty is revealed over time, in stages, allowing for remedial actions. We plan to explore these dynamic settings, referred to as stochastic programming with recourse. Another goal is to extend our methods to problems that involve probabilistic constraints where the inner constraints must *jointly* be satisfied with a given probability; such an extension is highly nonntrivial.

We also plan to work on further algorithmic improvements, including (i) developing more sophisticated summarization methods than minimum and maximum summaries; (ii) scaling up SummarySearch to very large datasets (e.g., millions of tuples) by combining summaries with divide-and-conquer approaches like SketchRefine [3]; (iii) parallelizing CSA-Solve and summary generation; and (iv) fully integrating stochastic package queries into a probabilistic database to handle multi-table queries.

We also plan to further develop our theory on SummarySearch to formally prove its convergence to feasible solutions as the number of scenarios increases.

Finally, we plan to explore ways to "open the black box" of optimization software to allow for further performance improvements, in analogy to the way MCDB re-engineered query operations to efficiently handle uncertain tuple attributes.

Acknowledgments. This work was supported by the NYUAD Center for Interacting Urban Networks (CITIES), and funded by: Tamkeen under the NYUAD Research Institute Award CG001, the Swiss Re Institute under the Quantum Cities initiative, and the National Science Foundation under grants IIS-1453543 and IIS-1943971. The authors would like to thank the anonymous reviewers for their valuable insights, and Arya Mazumdar, Nishad Ranade, and Senay Solak for their help and suggestions during various phases of the work.

REFERENCES

[1] S. Ahmed and A. Shapiro. Solving chance-constrained stochastic programs via sampling and integer programming. In *State-of-the-Art Decision-Making Tools in the Information-Intensive Age*, pages 261–269. Informs, 2008.

[2] D. Bienstock, M. Chertkov, and S. Harnett. Chance-constrained optimal power flow: Risk-aware network control under uncertainty. *SIAM Review*, 56(3):461–495, 2014.

[3] M. Brucato, A. Abouzied, and A. Meliou. Package queries: efficient and scalable computation of high-order constraints. *The VLDB Journal*, 27(5):693–718, 2018.

[4] M. Brucato, R. Ramakrishna, A. Abouzied, and A. Meliou. PackageBuilder: From tuples to packages. *PVLDB*, 7(13):1593–1596, 2014.

[5] M. Brucato, N. Yadav, A. Abouzied, P. J. Haas, and A. Meliou. Stochastic package queries in probabilistic databases. *arXiv*, 2020.

[6] Z. Cai, Z. Vagena, L. Perez, S. Arumugam, P. J. Haas, and C. Jermaine. Simulation of database-valued Markov chains using SimSQL. In *ACM SIGMOD*, pages 637–648, 2013.

[7] G. Calafiore and M. Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102(1):25–46, Jan 2005.

[8] G. Calafiore and F. Dabbene. *Probabilistic and Randomized Methods for Design Under Uncertainty*. Springer, 2006.

[9] M. C. Campi and S. Garatti. A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality. *Journal of Optimization Theory and Applications*, 148(2):257–280, 2011.

[10] M. C. Campi and S. Garatti. Wait-and-judge scenario optimization. *Mathematical Programming*, 167(1):155–189, 2018.

[11] M. C. Campi, S. Garatti, and M. Prandini. The scenario approach for systems and control design. *Annual Reviews in Control*, 33(2):149–157, 2009.

[12] M. C. Campi, S. Garatti, and F. A. Ramponi. A general scenario theory for nonconvex optimization and decision making. *IEEE Transactions on Automatic Control*, 63(12):4067–4078, 2018.

[13] G. Clare and A. Richards. Air traffic flow management under uncertainty: application of chance constraints. In *Proc. 2nd Intl. Conf. Application and Theory of Automation in Command and Control Systems*, pages 20–26. IRIT Press, 2012.

[14] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal–The International Journal on Very Large Data Bases*, 16(4):523–544, 2007.

[15] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *HyperText*, pages 35–44, 2010.

[16] E. Delage and Y. Ye. Distributionally robust optimization under moment uncertainty with application to data-driven problems. *Operations research*, 58(3):595–612, 2010.

[17] D. Dentcheva. Optimization models with probabilistic constraints. In *Probabilistic and randomized methods for design under uncertainty*, pages 49–97. Springer, 2006.

[18] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. *The VLDB Journal*, 18(2):469–500, 2009.

[19] N. E. Du Toit and J. W. Burdick. Probabilistic collision checking with chance constraints. *IEEE Transactions on Robotics*, 27(4):809–815, 2011.

[20] J. Dupačová, N. Gröwe-Kuska, and W. Römisch. Scenario reduction in stochastic programming. *Mathematical programming*, 95(3):493–511, 2003.

[21] N. Geng, X. Xie, and Z. Zhang. Addressing healthcare operational deficiencies using stochastic and dynamic programming. *International Journal of Production Research*, 57(14):4371–4390, 2019.

[22] G. A. Hanasusanto, V. Roitch, D. Kuhn, and W. Wiesemann. A distributionally robust perspective on uncertainty quantification and chance constrained programming. *Mathematical Programming*, 151(1):35–62, 2015.

[23] T. Homem-de Mello and G. Bayraksan. Monte Carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19(1):56–85, 2014.

[24] L. J. Hong, Z. Hu, and G. Liu. Monte Carlo methods for value-at-risk and conditional value-at-risk: a review. *ACM Trans. Modeling and Computer Simulation*, 24(4):22, 2014.

[25] IBM CPLEX Optimization Studio. http://www.ibm.com/software/commerce/optimization/cplex-optimizer/.

[26] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo approach to managing uncertain data. In *ACM SIGMOD*, pages 687–700. ACM, 2008.

[27] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. The Monte Carlo database system: Stochastic analysis close to the data. *ACM Trans. Database Syst.*, 36(3):18:1–18:41, 2011.

[28] P. Jorion et al. *Financial Risk Manager Handbook*, volume 406. John Wiley & Sons, 2007.

[29] P. Kall, S. W. Wallace, and P. Kall. *Stochastic Programming*. Springer, 1994.

[30] R. Karuppiah, M. Martin, and I. E. Grossmann. A simple heuristic for reducing the number of scenarios in two-stage stochastic programming. *Computers & Chemical Engineering*, 34(8):1246–1255, 2010.

[31] H. Lam and F. Li. Sampling uncertain constraints under parametric distributions. In *2018 Winter Simulation Conference (WSC)*, pages 2072–2083. IEEE, 2018.

[32] J. Luedtke and S. Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2):674–699, 2008.

[33] J. Luedtke, S. Ahmed, and G. L. Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Mathematical Programming*, 122(2):247–272, Apr 2010.

[34] A. Nemirovski and A. Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2006.

[35] A. Nemirovski and A. Shapiro. Scenario approximations of chance constraints. In *Probabilistic and Randomized Methods for Design Under Uncertainty*, pages 3–47. Springer, 2006.

[36] S. M. Ross. *Introduction to Probability Models*. Academic Press, 2014.

[37] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on Stochastic Programming: Modeling and Theory, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014.

[38] L. Šikšnys and T. B. Pedersen. SolveDB: Integrating optimization problem solvers into SQL databases. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 14. ACM, 2016.

[39] J. E. Smith and R. L. Winkler. The optimizer's curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3):311–322, 2006.

[40] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool, 2011.

[41] The Sloan Digital Sky Survey, data release 12. http://cas.sdss.org/dr12/.

[42] TPC Benchmark™H. http://www.tpc.org/tpch/.

[43] Yahoo! Finance. http://finance.yahoo.com/.