sPaQLTooLs: A Stochastic Package Query Interface for Scalable Constrained Optimization

Matteo Brucato* Miro Mannino° Azza Abouzied° Peter J. Haas* Alexandra Meliou*

*University of Massachusetts Amherst
{matteo, phaas, ameli}@cs.umass.edu

°NYU Abu Dhabi
{miro.mannino, azza}@nyu.edu

ABSTRACT

Everyone needs to make decisions under uncertainty and with limited resources, e.g., an investor who is building a stock portfolio subject to an investment budget and a bounded risk tolerance. Doing this with current technology is hard. There is a disconnect between software tools for data management, stochastic predictive modeling (e.g., simulation of future stock prices), and optimization; this leads to cumbersome analytical workflows. Moreover, current methods do not scale. To handle a broad class of uncertainty models, analysts approximate the original stochastic optimization problem by a large deterministic optimization problem that incorporates many "scenarios", i.e., sample realizations of the uncertain data values. For large problems, a huge number of scenarios is required, often causing the solver to fail. We demonstrate sPaQLTooLs, a system for in-database specification and scalable solution of constrained optimization problems. The key ingredients are (i) a database-oriented specification of constrained stochastic optimization problems as "stochastic package queries" (SPQs), (ii) use of a Monte Carlo database to incorporate stochastic predictive models, and (iii) a new SUMMARYSEARCH algorithm for scalably solving SPQs with approximation guarantees. In this demonstration, the attendees will experience first-hand the difficulty of manually constructing feasible and high-quality portfolios, using real-world stock market data. We will then demonstrate how SUMMARYSEARCH can easily and efficiently help them find very good portfolios, while being orders of magnitude faster than prior methods.

PVLDB Reference Format:
Matteo Brucato, Miro Mannino, Azza Abouzied, Peter J. Haas, Alexandra Meliou. sPaQLTooLs: A Stochastic Package Query Interface for Scalable Constrained Optimization. *PVLDB*, 13(12): 2881-2884, 2020.
DOI: https://doi.org/10.14778/3415478.3415499

1. INTRODUCTION

Constrained optimization is central to decision making over a broad range of domains, including *finance* [7], *transportation* [4], *healthcare* [6], and the *travel industry* [5]. Consider, for example, the following very common investment problem.

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415499

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	?

Scenario$_j$

...	gain
...	0.1
...	0.05
...	−0.2
...	0.2
...	0.1
...	−0.7

Figure 1: Example input table for the FINANCIAL PORTFOLIO (left). The values of a stochastic attribute (**GAIN**, in this example) are unknown (shown by a question mark). Sample realizations of the uncertain ? values are generated by calls to simulation functions. Drawing a realization for each ? value generates a possible world, or scenario (right).

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 a* loss *of more than* $10 *will only happen with probability at most* 5%.

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 of more than $10 with probability at most 5%), 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, 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" [2], directly solvable as an *Integer Linear Program* (ILP) using off-the-shelf linear solvers such as IBM CPLEX [8], and declaratively expressible in the Package Query Language (PAQL). Because **GAIN** is stochastic, the investor is solving a *stochastic ILP* (SILP) instead. *Stochastic package queries* (SPQs) are a generalization of package queries that allow uncertainty in the data, thereby allowing specification and solution of stochastic ILP problems.

In our prior work [3], we introduced SPAQL, a simple language extension to PAQL [2] that allows easy specification of package queries with stochastic constraints and objectives. The SPAQL query for the FINANCIAL PORTFOLIO is:

```
SELECT PACKAGE(*) AS Portfolio
FROM Stock_Investments
SUCH THAT
  SUM(price) ≤ 1000 AND
  SUM(Gain) ≤ −10 WITH PROBABILITY ≤ 0.05
MAXIMIZE EXPECTED SUM(Gain)
```

The result of this query 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. For simplicity, we focus on a one-time decision; in real life, the stock information would be updated daily and new decisions would be made over time.

Our system demonstrates SUMMARYSEARCH [3], our scalable sPAQL evaluation algorithm. SUMMARYSEARCH approximates the given SILP by a deterministic ILP (DILP) that simultaneously incorporates multiple "*scenarios*", or possible worlds, for the future stock market. A scenario is a table where all random variables have been realized. The right-hand side of Figure 1 shows a possible scenario. To generate scenarios, we employ the Monte Carlo probabilistic data model [9], which offers support for arbitrary distributions via user-defined *variable generation* (VG) functions.

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 have a 10% probability of losing more than $\$10$, rather than a 5% probability, incurring more risk than desired. The state-of-the-art techniques attempt to mitigate this by iteratively adding more scenarios into the DILP. We implement this approach in an algorithm that we call NAÏVE; unfortunately, the NAÏVE DILP may quickly become too large for the solver to handle, and this approach often fails.

Our approach, SUMMARYSEARCH, instead facilitates feasible packages by replacing a set of scenarios with a very small synopsis thereof, called a "summary", which results in a "reduced" DILP that is much smaller than the original DILP used by NAÏVE. 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 feasible, so that the required number of 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.

In our demonstration, participants are given an investment budget and asked to use our interactive interface sPaQLTooLs [13] to construct an investment portfolio on real stock market data. They first attempt to build a financial portfolio *manually*, using commonsense techniques, such as looking for low-volatility stocks, and greedily adding one stock at a time to the portfolio package. The system then evaluates their portfolio on a large number of scenarios. Manually constructed portfolios are unlikely to be feasible, and therefore attendees experience first-hand the difficulty of building low-risk portfolios without our automated methods. Finally, the users solve the problem *automatically*, as a sPAQL query, via our advanced SUMMARYSEARCH. The interface shows how the initial solutions found by the system can potentially also be infeasible, and how quickly the system finds feasible solutions and improve on their objective value (expected gain). We also compare the solutions found by SUMMARYSEARCH and (when possible) the NAÏVE algorithm.

2. STOCHASTIC ILP

The field of *stochastic programming* (SP) [11] 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 expectation and probabilistic constraints and objective functions.

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 $\mathbb{E}\left(\sum_{i=1}^{N} \xi_i x_i\right) \odot v$, and a linear *probabilistic constraint* takes the form $\Pr\left(\sum_{i=1}^{N} \xi_i x_i \odot v\right) \geq p$, where $p \in [0, 1]$. In our example, x_i is the number of shares of the ith stock to buy, and ξ_i is the random gain from a share of this stock on the sell date. 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: $\min_x \mathbb{E}\left(\sum_{i=1}^{N} \xi_i x_i\right) = \min_x \sum_{i=1}^{N} \mathbb{E}(\xi_i)\, x_i$, and thus we take $c_i = \mathbb{E}(\xi_i)$. Similarly, an objective in the form of a probability can be written in canonical form using epigraphic rewriting [3].

3. QUERY EVALUATION METHODS

The state of the art in solving SILPs has been developed outside of the database setting [1]. We first describe this approach, which we embody in an algorithm called NAÏVE, and discuss its drawbacks. We then briefly sketch our improved algorithm, SUMMARYSEARCH, which is typically faster than NAÏVE by orders of magnitude and can handle problems that cause NAÏVE to fail. More details about both algorithms can be found in our prior work [3].

3.1 Naïve Query Evaluation

The NAÏVE algorithm follows these steps: First, it generates a sample of independent and identically-distributed (iid) scenarios; It then combines them into an approximating DILP, solves the DILP to obtain a solution x, and then validates the feasibility of x against a large number of out-of-sample *validation scenarios*. The process is iterated, adding more scenarios until the validation phase succeeds.

We 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) [10]. For example, with M scenarios, to approximate a probabilistic constraint of the form $\Pr\left(\sum_{i=1}^{N} \xi_i x_i \odot v\right) \geq p$ 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* $y_j = \mathbb{1}\left(\sum_{i=1}^{N} s_{ij} x_i \odot v\right)$ where the indicator function $\mathbb{1}(\cdot)$ equals 1 if the inner constraint is satisfied and equals 0 otherwise, and where s_{ij} is the realized value of the ith tuple under the jth scenario. Finally, we add the following linear constraint over

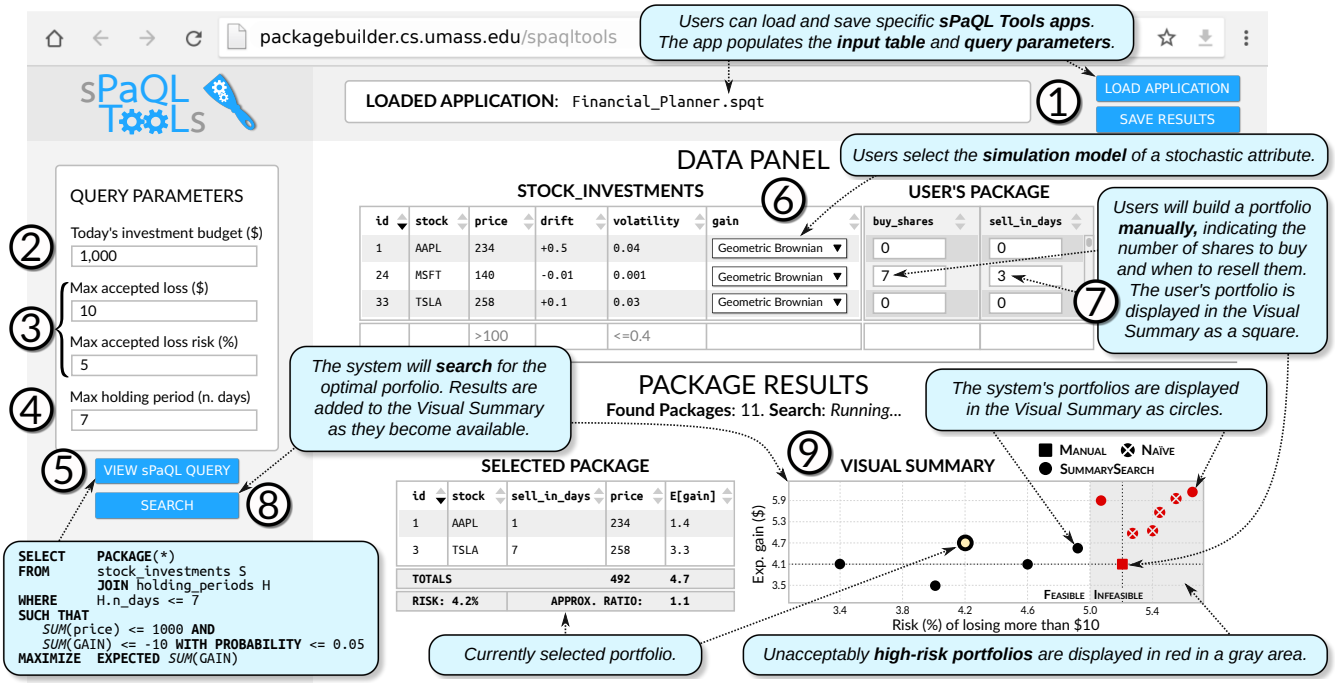

Figure 2: sPaQLTooLs GUI and demo outline: ① application loading, ②③④ query specification, ⑤ sPAQL inspection, ⑥ simulation modeling, ⑦ manual solution, ⑧ automatic solutions, ⑨ result exploration.

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 optimization phase for this SAA formulation can be very slow, and often the convergence to feasibility requires so many optimize/validate iterations that the SAA becomes too large for the solver to handle, so that NAÏVE fails. Our algorithm SUMMARYSEARCH uses "summaries" to speed up the optimization phase and reduce the number of required iterations.

3.2 Summary-based Query Evaluation

The NAÏVE algorithm has two major drawbacks. (1) The time to derive a solution to an SAA can be unacceptably long, since the size of the SAA 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 scenarios.

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" [12].

Our improved algorithm, SUMMARYSEARCH, 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 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 more stringent, and thus the solver is induced to produce feasible solutions faster.

SUMMARYSEARCH starts with an initial number of scenarios, $M \geq 1$, and uses a CSA formulation that replaces the M scenarios with Z conservative summaries, where $Z = 1$ initially. The feasibility of the resulting solution is checked using a large set of validation scenarios. If the current solution is infeasible, SUMMARYSEARCH increments M and tries again, until feasibility is achieved. Then the algorithm checks whether the objective value is within $(1 + \epsilon)$ of the true optimal value, where ϵ is defined by the user; this check uses the validation scenarios together with worst-case bounds developed in [3]. If the current approximation ratio exceeds $(1 + \epsilon)$, then SUMMARYSEARCH increases Z and iterates again. Increasing Z improves the approximation ratio; if this results in a loss of feasibility, then SUMMARYSEARCH increases M and the iterations continue. The algorithm stops if and when a feasible and $(1 + \epsilon)$-approximate solution is found.

Given values for M and Z, the summaries are computed as follows. First suppose that $Z = 1$. Let $\alpha \in [0, 1]$, and consider an inner constraint of the form $\sum_{i=1}^{N} \xi_i x_i \geq v$. An *$\alpha$-summary* of a set of scenarios is a newly constructed scenario $(s_1, \ldots, s_N)$ such that if a solution x satisfies the summary in that $\sum_{i=1}^{N} s_i x_i \geq v$, then x satisfies at least $\lceil \alpha M \rceil$ of the original scenarios. Constructing an α-summary, for $\alpha > 0$, is simple: Given any subset $G(\alpha)$ of $\lceil \alpha M \rceil$ scenarios, we define the summary as the tuple-wise minimum over $G(\alpha)$, i.e., $s_i := \min_{j \in G(\alpha)} s_{ij}$. SUMMARYSEARCH searches through values of α via a root-finding routine until it finds the least conservative one that (based on the validation scenarios) satisfies the inner constraint with probability greater or equal than, but as close as possible to, the required value p. (If no satisfactory α is found, then M is increased.) When $Z > 1$, the search process computes a separate α value for each summary. This overall process is executed independently for each probabilistic constraint.

Our experiments [3] showed 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 compa-

rable; typically, however, the number of iterations is actually much lower for SUMMARYSEARCH than for NAÏVE due to the conservative nature of summaries, further augmenting the performance gain.

4. DEMONSTRATION

We demonstrate our SPAQL query engine on a real dataset of stocks for portfolio optimization. Figure 2 shows a screenshot of the sPaQLTooLs's graphical user interface. During the demonstration, we guide the participants through the following steps.

Step ① (Loading the sPaQLTooLs application). The User loads a sPaQLTooLs application from a `.spqt` configuration file, which points to the input database and tables, as well as the SPAQL query template and input parameters that populate all the graphical interface elements. While the interface can support a variety of applications, we showcase the FINANCIAL PLANNER. This phase populates the `Stock_Investments` table and the query parameters described next.

Step ② (Specifying the budget constraint). The first query parameter indicates how much money ($) the user wants to invest at most.

Step ③ (Specifying the accepted risk constraint). The next two parameters set the risk level the user is willing to accept with the investment. The maximum accepted loss indicates how much money ($) the user is willing to lose at most, and the risk (%) indicates the maximum probability with which this loss can occur.

Step ④ (Specifying the maximum holding period). The user then specifies the maximum number of days d to hold any stock. This will generate d rows in the `Stock_Investments` table for each unique stock. The larger the value of d, the more uncertainty and hence the harder the optimization.

Step ⑤ (SPAQL inspection). Once all query parameters have been specified, the user can inspect the SPAQL query used by the system to search for the optimal portfolio.

Step ⑥ (Specifying the simulation model). For all the stochastic attributes in the input table (in this application, only **GAIN**), the user selects a simulation model from a drop-down menu. Users can select a different simulation model for different tuples, the same for all tuples, or any other combination. Simulation models are defined in the application configuration.

Step ⑦ (Manual financial planning). Attendees are asked to build a financial portfolio manually, in an attempt to compete against our system that uses SUMMARYSEARCH to find the optimal portfolio automatically. For example, a user may filter stocks by their volatility and price, as shown in the figure, and decide to buy a number of shares for some low-volatility stocks. As the user starts building their manual portfolio, the system runs simulations in the background, to estimate the associated risk and expected gain. The constructed portfolio is then shown in the Visual Summary at the bottom as a square, placed in the graph according to its risk and expected gain. The portfolio may be *infeasible*, i.e., it may not not satisfy the risk constraint, in which case it is colored in red and placed in a gray area. Users experience first-hand the difficulty of manually constructing feasible portfolios with high-enough gain.

Step ⑧ (Automatic financial planning). Users then run our system to search for the optimal portfolio according to their needs. We concurrently run both SUMMARYSEARCH and NAÏVE (for comparison).

Step ⑨ (Visual exploration of the results). The results of our system are interactively shown in the Visual Summary as they become available. As the system produces a solution, the interface plots its associated risk and expected gain. As feasible solutions are found, SUMMARYSEARCH starts improving their objective value (expected gain). At the end of the search, the final solution has the highest expected gain, under the acceptable risk. Users are able to compare this result with the portfolio that they manually built. Attendees can also click on any solution in the Visual Summary in order to view the full portfolio details, as a table on the left of the summary.

5. 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.

6. 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] M. Brucato, A. Abouzied, and A. Meliou. Package queries: efficient and scalable computation of high-order constraints. *The VLDB Journal*, Oct 2018.

[3] M. Brucato, N. Yadav, A. Abouzied, P. J. Haas, and A. Meliou. Stochastic package queries in probabilistic databases. In *SIGMOD*, pages 269–283, 2020.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] IBM CPLEX optimization studio. `https://www.ibm.com/analytics/cplex-optimizer`.

[9] 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 *SIGMOD*, pages 687–700, 2008.

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

[11] 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.

[12] 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.

[13] sPaQLTooLs. `http://packagebuilder.cs.umass.edu/spaqltools`.