TOP- K ALGORITHMS FOR SIMQL: A DECISION GUIDANCE QUERY LANGUAGE BASED ON STOCHASTIC SIMULATION

by

Susan Farley
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

__________________________________________
Dr. Alexander Brodsky, Dissertation Director

__________________________________________
Dr. Amihai Motro, Committee Member

__________________________________________
Dr. Larry Kerschberg, Committee Member

__________________________________________
Dr. Lance Sherry, Committee Member

__________________________________________
Dr. Stephen Nash, Senior Associate Dean

__________________________________________
Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: ________________________
Fall Semester 2013
George Mason University
Fairfax, VA
Top- K Algorithms for SimQL: A Decision Guidance Query Language Based on Stochastic Simulation

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Susan Farley
Masters of Engineering
Old Dominion University, 2001
Bachelor of Science
University of Georgia, 1998

Director: Alexander Brodsky, Associate Professor
Department of Computer Science

Fall Semester 2013
George Mason University
Fairfax, VA
THIS WORK IS LICENSED UNDER A CREATIVE COMMONS

ATTRIBUTION-NODERIVS 3.0 UNPORTED LICENSE.
DEDICATION

This is dedicated to my wonderful husband Davis and our beautiful daughter Beatrice.
ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Alex Brodsky for his continuing help and support during this process. I would also like to thank my committee members Dr. Motro, Dr. Kerschburg, and Dr. Sherry.

I extend great gratitude to my dissertation support group, John McDowell, Mark Coletti, and Jeff Bassett who really made this journey a lot easier than it could have been. Thank you for the support during our weekly status meetings, Saturday writing group meetings, and for providing a sound board for my papers and presentations.

I would also like to thank Dennis “Uncle Dad” Johnson for always finding the time to read the latest version of a paper or my dissertation.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>List of Equations</td>
<td>x</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>xi</td>
</tr>
<tr>
<td>Abstract</td>
<td>xii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Research Gap</td>
<td>3</td>
</tr>
<tr>
<td>1.2. Research Questions and Objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.3. Thesis Statement and Summary of Key Contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.4. Dissertation Organization</td>
<td>8</td>
</tr>
<tr>
<td>2. Related Work</td>
<td>10</td>
</tr>
<tr>
<td>2.1. Probabilistic Databases</td>
<td>10</td>
</tr>
<tr>
<td>2.2. Statistical Databases</td>
<td>10</td>
</tr>
<tr>
<td>2.3. Simulation</td>
<td>11</td>
</tr>
<tr>
<td>2.4. Top-K Queries</td>
<td>11</td>
</tr>
<tr>
<td>2.5. Monte Carlo</td>
<td>12</td>
</tr>
<tr>
<td>2.6. Optimal Computing Budget Allocation (OCBA)</td>
<td>12</td>
</tr>
<tr>
<td>2.7. Airline Scheduling Problem to Minimize Passenger Trip Delay</td>
<td>14</td>
</tr>
<tr>
<td>3. Simulation Query Language (SimQL): Data Model, Syntax, and Semantics</td>
<td>16</td>
</tr>
<tr>
<td>3.1. Stochastic Relational Data Model</td>
<td>16</td>
</tr>
<tr>
<td>3.2. SimQL by Example</td>
<td>18</td>
</tr>
<tr>
<td>3.3. Formal Syntax and Semantics of SimQL</td>
<td>26</td>
</tr>
<tr>
<td>4. Algorithms for Top-K SimQL Queries Based on Entire Search Space Exploration and Simulation Budget Optimization</td>
<td>34</td>
</tr>
<tr>
<td>4.1. Top K Algorithm Template (TKAT)</td>
<td>35</td>
</tr>
</tbody>
</table>
4.2. Implementation of Monte Carlo (a.k.a. Equal Opportunity) in SimQL .......... 37
4.3. Implementation of OCBA in SimQL .................................................. 38
5. Algorithm(s) for Top-K Queries Based on Partial Search Space Exploration for Continuous Decision Variables Based on Regression Analysis Heuristics ......................... 40
   5.1. An Efficient Regression Based Algorithm for Top-K Selection .............. 40
   5.2. An Efficient Iterative Regression Based Algorithm for Top-K Selection .... 44
6. Algorithms for Top-K Scheduling Queries Based on Partial Search Space for Discrete Decision Variables Based in Enumeration Heuristics ............................................ 48
   6.1. Deterministic Scheduling Problem and Algorithm ................................. 48
   6.2. Stochastic Scheduling Problem and Algorithm ..................................... 65
   6.3. Specialized Transportation Rescheduling Algorithm for Optimizing Heuristics 79
7. Prototype System .................................................................................... 99
8. Experimental Evaluation ............................................................................. 106
   8.1. Monte Carlo, OCBA, GORBASH and GORBASH+ ......................... 106
   8.2. SMRGOL and BRYAGH ................................................................ 111
9. Conclusions and Future Work ................................................................... 113
Appendix ........................................................................................................... 115
Index ............................................................................................................... 117
References ...................................................................................................... 119
Curriculum Vitae ............................................................................................ 122
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1 Settings in the sim_config table that the user can use to alter the simulation.</td>
<td>26</td>
</tr>
<tr>
<td>Table 2 Functions and Signatures</td>
<td>28</td>
</tr>
<tr>
<td>Table 3 Terminal Symbols for SimSQL</td>
<td>29</td>
</tr>
<tr>
<td>Table 4 Syntax of SimSQL Queries</td>
<td>30</td>
</tr>
<tr>
<td>Table 5 Syntax of SimSQL Views</td>
<td>31</td>
</tr>
<tr>
<td>Table 6 Assign_Budget for Monte Carlo Algorithm</td>
<td>38</td>
</tr>
<tr>
<td>Table 7 Assign_Budget for OCBA Algorithm</td>
<td>39</td>
</tr>
<tr>
<td>Table 8 GORBASH Algorithm</td>
<td>42</td>
</tr>
<tr>
<td>Table 9 GORBASH+ Algorithm</td>
<td>45</td>
</tr>
<tr>
<td>Table 10 Elements of the Scheduling Problem</td>
<td>49</td>
</tr>
<tr>
<td>Table 11 Deterministic Scheduling Problem</td>
<td>54</td>
</tr>
<tr>
<td>Table 12 DALEXS: Deterministic Algorithm for Load and machine reScheduling</td>
<td>55</td>
</tr>
<tr>
<td>Table 13 HardDependencyReschedule</td>
<td>56</td>
</tr>
<tr>
<td>Table 14 SoftDependencyReschedule</td>
<td>58</td>
</tr>
<tr>
<td>Table 15 TARDIS: Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios</td>
<td>60</td>
</tr>
<tr>
<td>Table 16 RescheduleSingleLoad</td>
<td>62</td>
</tr>
<tr>
<td>Table 17 RescheduleActivity</td>
<td>64</td>
</tr>
<tr>
<td>Table 18 Simulation of Stochastic Schedule (SSS)</td>
<td>65</td>
</tr>
<tr>
<td>Table 19 Stochastic Scheduling Problem</td>
<td>69</td>
</tr>
<tr>
<td>Table 20 Monte Carlo Stochastic Scheduling Algorithm</td>
<td>71</td>
</tr>
<tr>
<td>Table 21 ARAGH: All-puRpose Algorithm for Generating a scHedule</td>
<td>72</td>
</tr>
<tr>
<td>Table 22 CreateSchedule</td>
<td>75</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1 OCBA Technique ................................................................. 13
Figure 2 GORBASH Top Q Selection ...................................................... 41
Figure 3 Passenger Trip Delay (solid) is determined by one of the scenarios described in Space-Time diagrams. The dashed line corresponds to the scheduled arrival time of the flight. [26] ........................................................................................................ 83
Figure 4 SMRGOL algorithm used to minimize passenger trip delay by either holding the next flight till a passenger arrives or rescheduling passengers from late connecting flights on the next available flight ................................................................. 88
Figure 5 PCS vs. Number of Simulations for 1 million tuples ..................... 108
Figure 6 PCS vs. simulations for 1 million tuples ........................................ 109
Figure 7 PCS vs. simulations for 1 million tuples ........................................ 110
Figure 8 Hub and spoke network with flights going into and out of the hub ........ 111
# LIST OF EQUATIONS

<table>
<thead>
<tr>
<th>Equation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1: Probability of Correct Selection for GORBASH</td>
<td>41</td>
</tr>
<tr>
<td>Equation 2: PTD</td>
<td>81</td>
</tr>
</tbody>
</table>
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARAGH</td>
<td>All-puRpose Algorithm for Generating a scHedule</td>
</tr>
<tr>
<td>BRYAGH</td>
<td>Basic Reduction Yare Approach for fliGHts</td>
</tr>
<tr>
<td>CA</td>
<td>Composite Activity</td>
</tr>
<tr>
<td>CT</td>
<td>Composite Task</td>
</tr>
<tr>
<td>GORBASH</td>
<td>General Optimal Regression Budget Allocation ScHeme</td>
</tr>
<tr>
<td>DALEXS</td>
<td>Deterministic ALgorithm for EXecutable Scheduling</td>
</tr>
<tr>
<td>HD</td>
<td>Hard Dependency</td>
</tr>
<tr>
<td>LNCT</td>
<td>Load Normalized Composite Time</td>
</tr>
<tr>
<td>OCBA</td>
<td>Optimal Computing Budget Allocation</td>
</tr>
<tr>
<td>PCS</td>
<td>Probability of Correct Selection</td>
</tr>
<tr>
<td>PTD</td>
<td>Passenger Trip Delay</td>
</tr>
<tr>
<td>SSS</td>
<td>Simulation of Stochastic Schedule</td>
</tr>
<tr>
<td>SimQL</td>
<td>Simulation Query Language</td>
</tr>
<tr>
<td>SMRGOL</td>
<td>Schedule Minimization foR Generalized Operational Logistics</td>
</tr>
<tr>
<td>SD</td>
<td>Soft Dependency</td>
</tr>
<tr>
<td>TARDIS</td>
<td>Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios</td>
</tr>
<tr>
<td>WCT</td>
<td>Weighted Composite Time</td>
</tr>
</tbody>
</table>
ABSTRACT

TOP- K ALGORITHMS FOR SIMQL: A DECISION GUIDANCE QUERY LANGUAGE BASED ON STOCHASTIC SIMULATION

Susan Farley, Ph.D.
George Mason University, 2013
Dissertation Director: Dr. Alexander Brodsky

Many applications in diverse areas such as manufacturing systems and sustainable energy systems require making complex decision based on stochastic data. Probabilistic and statistical databases are excellent for declarative queries, but do not support more complex stochastic processes defined through stochastic simulation. Stochastic simulation allows the user to create complex models to simulate an event, but does not support declarative formulation of queries and is time consuming. To support this type of application, this dissertation introduces Simulation Query Language, SimQL, a language extends the database query language SQL with stochastic attributes/ random variables defined by simulation. I also propose algorithms for computing top-k answers based on partial search space exploration for continuous decision variables using regression analysis and algorithms based on enumeration heuristics for scheduling problems. I also
conducted experimentation comparing the algorithms for continuous decision variables and a case study for a class of scheduling problems.
1. INTRODUCTION

Making stochastic decisions is a part of everyday life. Unfortunately, making decisions based on stochastic data where the computation of even one outcome may involve a computationally expensive stochastic simulation. There has been significant research has been done in the areas of probabilistic and statistical databases to store and manipulate uncertain data. However, they deal with explicit probability distributions whereas, many classes of problems require probabilistic distributions that are complex and implicitly defined by stochastic simulation. Stochastic simulations are time consuming because decisions are based on trial and error even after applying heuristics to speed up the process. To bridge the gap, in this dissertation, I introduce SimQL, a decision-guidance query language with attributes representing random variables and defined by a stochastic simulation and developing efficient algorithms for top k queries.

For example, suppose the fictional Camford University is conducting studies on wind turbine locations and turbine models and would like to answer related questions such as:

- Find the list of location and turbine model pairs with their average wind speeds where the wind speed is a random variable.
- Rank the locations in decreasing order of their expected power generation (in kilowatts) where power generation is a random variable that is dependent on the wind speed at the location.
- Return the 10 top ranked locations where the probability that the expected wind power generated being greater than 8600 kilowatts is greater than 80%.

As another example, suppose an airline wants to schedule flights in a way that minimizes passenger trip delay with minimal disruption to the airline’s overall operations. To this end, planes can be delayed so that passengers can make connecting flights, passengers can be rescheduled on another flight, flights can be cancelled, or any combination of these.

Even with a considerable amount of data to help with the decision making, there are still problems to overcome. The data available to solve these problems:

- Can have probabilities attached to it,
- Can be statistical in nature, or
- Can be described as a stochastic process.

In applications such as the one facing Camford and the airline industry, for every decision choice, the corresponding utility or objective may need to be determined using a possibly complex stochastic simulation. This leads to two core challenges:

- Each simulation is expensive and may need to be run multiple times
- Databases do not support arbitrarily complex simulations.
1.1. Research Gap

Traditional relational databases do not process probabilistic data and do not have the additional functionality needed for analyzing statistical data or defining stochastic attributes. To overcome these limitations, considerable research has been done in the field of probabilistic and statistical databases as well as stochastic simulation, but this work also has limitations.

Probabilistic and statistical databases have been popular in solving problems like those described above because they are capable of managing large amounts of uncertain data [1] and support complex queries [2]. The shortcomings with probabilistic and statistical databases is that they deal with explicit probability distributions, whereas in many applications probabilistic distributions are complex and implicitly defined, e.g., by stochastic simulation.

There has been significant work on stochastic simulation, allowing a user to create a complex model to simulate an event and serves as an implicitly defined stochastic model. However, stochastic simulations do not support declarative formulation of queries. Furthermore, it is time consuming to make decisions based on repetitive experiments even after applying heuristics to speed up the process [3]. The user has to repeatedly alter the model until the desired answer is achieved.

To support stochastic simulation, algorithms have been developed to efficiently use the available simulation budget and to increase the probability of correct selection for different types of problems (i.e., top-k). Monte Carlo is a popular algorithm that is easy to implement and assigns an equal portion of the simulation budget to every design [4].
Unfortunately, Monte Carlo does not complete till the entire simulation budget has been consumed. It also does not efficiently spend the number of simulations available; giving the same number of simulations to tuples that have a very low probability of being included in the top k. Optimal Computing Budget Allocation (OCBA) improves on Monte Carlo by optimizing the simulation budget allocation. The OCBA algorithm [5] assigns more of the simulation budget to tuples that have a higher chance of being included in the top-k. Unlike Monte Carlo which always exhausts the simulation budget, OCBA also stops when a certain probability of correct selection has been reached. Unfortunately, even with OCBA, the entire search space is considered so a number of simulations need to be run on all designs. Having to run simulations on the entire search space is expensive and can be prohibitive when the number of tuples is very large.

1.2. Research Questions and Objectives

It is desirable to develop a language, tools, and algorithms to support the accurate and rapid analysis of data containing attributes defined by stochastic simulation. The currently available technology can be built upon to enable easier and more immediate results while not sacrificing accuracy. The research questions this dissertation is addressing are as follows:

What should the language be for supporting declarative queries that contain stochastic attributes?

How to efficiently compute top k queries to questions like the ones facing Camford University and the airline industry?
How to experimentally demonstrate the quality and scalability of algorithms developed for top k queries?

1.3. Thesis Statement and Summary of Key Contributions

1.3.1. Thesis Statement

It is possible to design a stochastic relational model and query language with attributes representing random variables expressed through stochastic simulation and the analysis of these attributes with efficient algorithms for top-k queries.

1.3.2. Key Contributions

- **Introducing a Query Language to Support Stochastic Attributes Defined by Simulation**

  I introduced the Simulation Query Language, SimQL, in [6][7] to make the connection between relational databases and stochastic simulation. SimQL extends the database query language SQL with stochastic attributes/ random variables defined by simulation. It allows the user to make decisions using queries similar to what they are familiar with using in standard relational databases (i.e., SQL), yet with stochastic attributes implicitly defined by a simulation.

- **Algorithms Based on Entire Search Space Exploration and Simulation Budget Optimization**

  I implemented the algorithms Monte Carlo and Optimal Computing Budget Allocation, OCBA [8][5], for selecting top-k designs as part of SimQL syntax [7].
• Algorithm(s) Based on Partial Search Space Exploration for Continuous Decision Variables Based on Regression Analysis Heuristics

Unfortunately, with both Monte Carlo and OCBA, a number of simulations need to be run on all designs which can be prohibitive when the number of tuples is very large. General Optimal Regression Budget Allocation ScHeme, GORBASH, and GORBASH+ both use regression analysis to dodge the necessity of running a number of simulations for each tuple in the search space [9] [10]. GORBASH and GORBASH+ run a number of simulations on a small sample of tuples and then use regression analysis to estimate the stochastic attribute for the tuples not in the sample set. Whereas GORBASH spends the remaining simulation budget on a subset of tuples (Q) for OCBA, GORBASH+ takes a more iterative approach. During each iteration of the OCBA phase, GORBASH+ computes if the probability of correct selection would increase more if the simulation budget for that iteration was spent on an additional tuple Q+1 or if the simulation budget was spent on the existing Q tuples.

• Algorithm Based on Partial Search Space for Discrete Decision Variables Based in Enumeration Heuristics for Scheduling Problems

The heuristics that GORBASH and GORBASH+ use for continuous decision variables would not work for discrete decision variables. For discrete decision variables for a class of scheduling problems, I developed algorithms based on enumeration heuristics and conducted heuristic assessment of their utility as compared to other approaches.
I designed the algorithms Deterministic ALgorithm for EXecutable Scheduling, DALEXS, and Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios, TARDIS for use with deterministic schedules. These algorithms are generalized scheduling algorithms that take an existing deterministic schedule that is no longer feasible along with the current time and system status and generates a new feasible schedule. TARDIS improves upon DALEXS by optimizing the load delay of the schedule.

For stochastic schedules, I developed All-puRpose Algorithm for Generating a schedule, ARAGH. ARAGH is a generalized scheduling algorithm that takes an existing schedule that has uncertainty to it and that is no longer feasible and generates a new schedule that optimizes a utility of the schedule. After generating possible good candidate schedules, OCBA is used to determine the best schedule based on minimizing the delay.

For the specific problem of airline passenger delay, I designed Schedule Minimization foR Generalized Operational Logistics, SMRGOL and Basic Reduction Yare Approach for flighTs, BRYAGH. SMRGOL, takes a flight schedule as input and generates a schedule that minimizes passenger trip delay [11]. This can be done either by rescheduling the passenger or by holding the connecting flight for a period of time to allow for the passenger to make the flight. BRYAGH [12], fine-tuned the approach introduced with SMRGOL to further reduce passenger trip delay.

- **Airline Rescheduling Case Study**

  Using a twenty four hour period of flights through a single hub, I compared the passenger trip delay of SMRGOL and BRYAGH to rescheduling passengers on the next
available flight to their destination. If there was not a new connecting flight for a passenger during the twenty-four hour period, a 900 minute PTD was assumed.

- **Prototype System**

  A prototype of the language SimQL has been integrated with the open source database PostgreSQL using the relational data model with the addition of stochastic attributes and functions to analyze stochastic attributes. The Top K algorithms Monte Carlo, OCBA, GORBASH, GORBASH+, SMRGOL, and BRYAGH have been implemented and are included in the prototype system.

- **Experimental Evaluation**

  I ran a series of experiments increasing the number of tuples to be simulated up to 5 million to show the improvement of simulation budget used for GORBASH and GORBASH+ over Monte Carlo and OCBA. With k=5 and a desired confidence of 95% (for the OCBA, GORBASH, and GORBASH+ algorithms), I compared the simulation budget used along with the probability of correct selection achieved by Monte Carlo, OCBA, GORBASH and GORBASH+. For GORBASH+, I sampled 0.1% of the tuples for the regression line.

1.4. **Dissertation Organization**

  This dissertation is organized as follows. In Chapter 2, I review related work for probabilistic and statistical databases and OCBA. In Chapter 3, I explain the stochastic relational data model and introduce the syntax and semantics of SimQL. In Chapters 4, 5 and 6, I propose solutions for three classes of top-k problems. I explain Monte Carlo and OCBA as used for top-k queries in Chapter 4 and introduce GORBASH in Chapter 5. In
Chapter 6, I describe how SimQL can be used to optimize a specific utility for scheduling problems. I define the deterministic and stochastic scheduling problem and introduce algorithms for optimizing the utility of the schedule. With SMRGOL, I minimize PTD for the flight scheduling problem in Chapter 6. In Chapter 7, I give details on the canonical implementation of SimQL. The results of experimentation are described in Chapter 8. Then I conclude the paper in Chapter 9 with the current status of my research.
2. RELATED WORK

2.1. Probabilistic Databases

Probabilistic databases are databases capable of managing large amounts of uncertain data [1] and support complex queries [2]. Typically, each tuple, or part of a tuple, has a probability associated with it corresponding to its likelihood. Implicitly, this defines a probability distribution over all “possible worlds,” each corresponding to a regular relational database. However, computing the exact confidence of the resulting tuples for a query is typically an NP complete problem. As a result, researchers have to use approximation techniques [13] or “systems that do not scale to the same extent as” relational databases [2]. The shortcomings with probabilistic databases are that it deals with explicit probability distributions, whereas in many applications probabilistic distributions are complex and implicitly defined, e.g., by stochastic simulation. Also, the more precise the probabilities are, the harder the database is to use and scale making working with large amounts of data both cumbersome and time consuming [2].

2.2. Statistical Databases

Statistical databases are usually relational databases containing statistical data that have been extended to allow for advanced statistical analysis techniques. Unfortunately, the quality of data is not always optimal and there are security problems resulting in
either the queries being too restrictive or the users possibly getting private information that they should not have access to [14]. “The attention given to protecting mining output is fairly limited” [15] and still prohibits the user from controlling “precisely the formation of query sets [16].” Relational databases also do not extend well to manage statistical data because of the high redundancy and sparseness of data leads to inefficient retrieval of tuples [17].

2.3. Simulation

Simulation allows a user to create a complex model to imitate a real world system for either a specified amount of time or till a certain event occurs. A simulation serves as an implicitly defined stochastic model. A simulation model is flexible in representing a real system can be used repeatable to analyze different parameters or plans [18]. However, it is time consuming to make decisions based on trial and error, even after applying heuristics to speed up the process.

2.4. Top-K Queries

Many questions can be reduced to a top-k query. For instance: “what is the most efficient use of the local warehouse” becomes “what are the top five selling sprockets” and “what flight route should the extra plane be put on” becomes “what flight route has the most standby passengers.” Though these questions can be answered with a simple query, more complex questions based on stochastic problems, such as the one facing Camford, require running simulations. The difficulty is the computational power required to run these simulations given the number of possible solutions that exist. There are
different algorithms that can be used to assign the computational budget to tuples for top-
k simulations [19]. I have implemented two of these algorithms in SimQL: Monte Carlo
and OCBA. I also introduce two additional algorithms: GORBASH and GORBASH+.

2.5. Monte Carlo

Monte Carlo is “particularly useful for path-dependent options and others for
which no known formula exists.” [20] For top-k queries, the Monte Carlo algorithm
allocates an equal portion of the computational budget to each tuple for a number of
iterations until the budget has been exhausted. The stopping criterion for this algorithm is
when the budget has been exhausted. The probability of correct selection cannot be
computed till the algorithm is completed and is based on the number of times the top-k
tuple was selected as the best divided by the number of iterations.

2.6. Optimal Computing Budget Allocation (OCBA)

Though the Monte Carlo algorithm for selecting top-k tuples is easy to
implement, it is not efficient in its allocation of the simulation budget. Every tuple
receives the same allocation of the simulation budget and the algorithm does not stop till
the entire simulation budget has been used.

The OCBA algorithm [5] allows for the optimal allocation of a computational
budget when selecting the top-k tuples in a query as well as the ability to attain a certain
probability that a row is contained in the top-k with minimal simulations. After an
initialization run, a portion of the computational budget is assigned to each tuple for
additional simulation runs. Using OCBA ensures that a larger portion of the given computational budget is allocated to the tuples that have a higher variance and that tuples that are further away from the top-k “cut-off” have a smaller budget (Figure 1). This ensures that tuples with a higher prospect of “jumping the line” to be included in the top-k results as well as tuples that could be downgraded into not being included in the top-k, get more attention than tuples that do not have any chance of switching from their current designation. After each portion of the computational budget has been utilized, OCBA is used again to allocate more of the computational budget if the desired degree of certainty that the selected rows are indeed in the top-k has not been reached. This process continues until either the computational budget has been depleted or the desired probability has been reached.

Figure 1 OCBA Technique
Both Monte Carlo and OCBA algorithms for top-k selection have been incorporated into SimQL.

2.7. Airline Scheduling Problem to Minimize Passenger Trip Delay

The U.S. government and airline industry are working together to improve the airline transportation system (ATS). The first approach, the Airport Improvement Plan [21], attempts to increase the airport infrastructure (i.e., runways, gates, service facilities) at key positions of the air transportation system to increase throughput. The second approach, NextGen [22], hopes to “improve the productivity and the utilization of existing airspace” [23] with the air traffic control modernization program that will cost $37B.

In [23], the expected impact of the Airport Improvement Plan and the effect of NextGen on passenger trip reliability was described with a probabilistic model. The main results of the analysis using this model are as follows:

- “Flight delays account for approximately 41% of the total passenger trip delays. The remaining passenger trip delays are a result of trip delays experienced by passengers due to cancelled flights and missed connections.” [23]
- “The way airlines design their networks has a significant impact on total passenger trip delay. The ratio between direct and connecting itineraries, the time between banks at the hubs, the frequency of service, and the
selection of aircraft size and target load factor play a significant role in determining passenger trip reliability.”[23]

Though the model showed that the Airport Improvement Plan and NextGen do improve flight on-time performance, it also showed that they do not improve the passenger experience. More needs to be done to improve passenger trip reliability and the associated costs.
3. SIMULATION QUERY LANGUAGE (SIMQL): DATA MODEL, SYNTAX, AND SEMANTICS

I have integrated a prototype of the language SimQL with the open source database PostgreSQL using the relational data model with the addition of stochastic attributes. PostgreSQL was chosen because it is a “more densely featured database system often described as an open-source version of Oracle,” [24] but SimQL can be integrated into any database that allows Java functions. In the following sections, I describe the data model, syntax, and semantics.

3.1. Stochastic Relational Data Model

My data model is the basic relational model extended with probabilistic attributes as described below using the definitions from [25].

“Definition” An n-ary stochastic relational schema, or simply s-schema, is a set of n pairs \( S = \{ A_1 : T_1, \ldots, A_n : T_n \} \) with \( A_i \neq A_j \) if \( i \neq j \), where each \( A_i \) (\( i = 1, \ldots, n \)) is an attribute name, or simply attribute, and each \( T_i \) (\( i = 1, \ldots, n \)) is a type name or domain. The attributes \( A_1, \ldots, A_n \) are partitioned into two sets: the regular attributes \( S_{\text{Reg}} \) and the probabilistic attributes \( S_{\text{Prob}} \).

When \( T_1, \ldots, T_n \) are understood in an s-schema, we may abbreviate the s-schema as \( S = \{ A_1, \ldots, A_n \} \). Semantically, each \( T_i \) is associated with a domain, denoted \( \text{Dom}(T_i) \).
or simply $\text{Dom}(A_i)$. Without loss of generality, we assume $S_{\text{Reg}} = \{A_i, \ldots, A_k\}$ and $S_{\text{Prob}} = \{A_{k+1}, \ldots, A_n\}$ for some $0 \leq k \leq n$. With this notation, $S_{\text{Reg}} = \emptyset$ when $k = 0$, and $S_{\text{Prob}} = \emptyset$ when $k = n$.

**Definition** Given an $n$-ary s-schema $S = \{A_1, \ldots, A_n\}$, an s-instance of $S$ is a finite set of s-tuples over $S$, where each s-tuple $t$ over $S$ consists of two elements $t_{\text{Reg}}$ and $t_{\text{Prob}}$ such that $t_{\text{Reg}}$ is a (regular) tuple over the attributes $S_{\text{Reg}}$ and $t_{\text{Prob}}$ is an $(n - k)$-dimension probability density function (PDF) over the domain $\text{Dom}(A_{k+1}) \times \cdots \times \text{Dom}(A_n)$, where $\{A_{k+1}, \ldots, A_n\} = S_{\text{Prob}}$.

I use $r$ to denote an s-instance of $S$, and $t$ an s-tuple of $r$, consisting of $t_{\text{Reg}}$ and $t_{\text{Prob}}$. In special cases when $S_{\text{Reg}} = \emptyset$ or $S_{\text{Prob}} = \emptyset$, I let $t_{\text{Reg}}$ or $t_{\text{Prob}}$, respectively, be the empty tuple (i.e., 0-ary tuple).

Alternatively, we may write an s-tuple in the form $t = (a_1, \ldots, a_k, pdf)$. Mathematically, the pdf takes $A_{k+1}, \ldots, A_n$ as its random variables. Semantically, an s-tuple of the above form gives the conditional probability density function $pdf(A_{k+1}, \ldots, A_n| A_1 = a_1, \ldots, A_k = a_k)$.

Note that in the above, $\text{Dom}(A_j)$, $k + 1 \leq j \leq n$, are implicitly assumed to be dense domains, but we can easily accommodate discrete domains.” [25]

As described in [7], the data stored in the tables is conformant to the pure relational data model as is what is returned to those tables after queries and analysis are complete. When a view definition contains stochastic attributes, the stochastic attributes are never materialized, but a materialized view is a possible world relational data view. Only queries that return a purely relational answer are materialized, therefore there is
never a problem with storing stochastic data. A view that is of the stochastic relational
type is defined implicitly with a stochastic function.

For instance, a view with a stochastic attribute of the wind generated electricity
based on the average annual wind speed, along with other attributes like the wind turbine
hub height, a deterministic attribute in the table wind_power_info, can be defined as:

```
CREATE OR REPLACE VIEW vwWndPwr AS
    SELECT wind_id, location,
           avg_speed,
           speedSimulation(avg_speed) AS wdout
    FROM wind_power_info;
```

This view uses an imported function speedSimulation that was written as part of a
Java library that was imported for use in the database. Note that the stochastic attribute
wdout defined in the `SELECT` clause is a stochastic attribute which defines a random
variable and has an associated probability distribution. So SimQL would not allow this
view to be materialized. However, if we added the SimQL’s EXP or VAR functions to
the expression for wdout, then the real value would be computed and could be
materialized.

3.2. **SimQL by Example**

SimQL is an extension of standard SQL language that allows a user to define and
analyze unknown attributes that depend upon a stochastic simulation based on known
attributes stored in a standard data table.
To define an attribute that is based on the materialization of a stochastic attribute, it is necessary to define the stochastic attribute using a WITH clause:

```
CREATE OR REPLACE VIEW vwWndPwr_2 AS
SELECT wind_id, location, avg_speed,
       wnd.wdout AS wind_out,
       wnd.wdout*17.5 AS power_out
FROM simql.wind_power_info w
WITH wnd AS
    (SELECT wind_id, speedSimulation(avg_speed) AS wdout
     FROM simql.wind_power_info);
```

This example is similar to an inline view using the speedSimulation function that was defined in Java and imported into PostgreSQL. It is also possible to create a function that returns a tuple based on simulation. For example, a function pwr_funct can be created that takes the average wind speed as an input and returns a tuple with the wind and power out based on a simulation. First we define the tuple type:

```
CREATE TYPE pwr_type AS (wdout double, pwout double);
```

Then we create the function that returns the new tuple type pwr_type. This function can be a very complex simulation written in SQL, Java, or a combination of both.

```
CREATE FUNCTION pwr_funct(avg_speed double) RETURNS pwr_type
AS $BODY$
```
DECLARE answers pwr_type;
BEGIN

SELECT speedSimulation(avg_speed)
INTO answers.wdout;

answers.pwout := answers.wdout*17.5;

RETURN answers;
END;

Finally, the view can be created using this function instead of trying to do all of the computations using inline SQL. This allows for more complex attribute definitions such as a stochastic attribute depending upon the output of another stochastic attribute.

CREATE OR REPLACE VIEW vwWndPwr_2 AS

SELECT wind_id, location, avg_speed, wnd.wdout AS wind_out,
wnd.pwout AS power_out
FROM wind_power_info w
WITH wnd AS
(SELECT pwr_funct (avg_speed));

Six new functions are defined to help analyze the stochastic attributes: EXP, VARIANCE, PROB, PROBABILITY, TOPK, TOPG, and TOPG2. Each runs the simulation for the stochastic variable using the parameters set in the sim_config table.
(described below) with EXP returning the *approximation of expectation* of the stochastic attribute, VARIANCE returning the *variance* of the stochastic attribute, PROB returning the *probability* that the stochastic attribute satisfies the inequality equation, PROBABIL ITY returning *true or false* based of the stochastic attribute satisfying the inequality equation. TOPK, TOPG, and TOPG2 are functions that return a table that can be queried directly or be joined with another table(s) as part of a query. The attributes returned by TOPK, TOPG, and TOPG2 include:

- **Tablepk** (integer): the primary key of the tuple from the table or view used to generate the topk results,
- **Rank** (integer): the ranking of the tuple according to the simulated mean of the stochastic attribute,
- **Mean** (double precision): the simulated mean of the stochastic attribute,
- **Variance** (double precision): the simulated variance of the stochastic attribute,
- **Numsamples** (integer): the number of samples (simulation runs) of the stochastic attribute for the tuple,
- **Probability** (double precision): the statistical confidence that the rank (k position) is correct.

EXP, VARIANCE, and PROB use a canonical evaluation based on Monte Carlo, TOPK uses OCBA, TOPG uses GORBASH, and TOPG2 uses GORBASH+.

Once a view has been created, the user can query the view like a standard table, materializing the stochastic attribute into a deterministic attribute (possible world).
The functions EXP, VARIANCE, PROB, and PROBABILITY added by SimQL can be used as part of the SELECT statement or as part of the WHERE clause. The functions TOPK, TOPG, and TOPG2 return a table so it can be used as either part of the FROM clause or as a JOIN to a table. Each function requires you to pass in the name of the view. EXP, VARIANCE, TOPK, TOPKG, and TOPG2 also require the name of the desired stochastic attribute as more than one stochastic attribute can be defined in a single view. PROB and PROBABILITY require an inequality expression passed in as a string. Excluding TOPK, TOPKG, and TOPG2, these functions also require the user to pass in the primary key of the view.

The query below uses the function EXP to return the approximation of the expected wind output after the simulation has been run for the allocated budget along with the location and annual average wind speed as part of the SELECT clause.

```sql
SELECT location, avg_speed, exp('vwWndPwr', 'wndout', wind_id)
FROM vwWndPwr;
```

A query that will return the variance of the output after the simulation has been run until the desired stopping point has been reached would look something like the following:

```sql
SELECT location, avg_speed, variance('vwWndPwr', 'wndout', wind_id)
```
FROM vwWndPwr;

These functions can also be used in the WHERE clause to eliminate tuples that do not fit the desired criterion. The following query would only return the tuples where the probability that the expected output is greater than 8600 is more than eighty percent.

```
SELECT location, avg_speed
FROM vwWndPwr
WHERE prob('vwWndPwr', 'wndout>8600', wind_id) > .8);
```

To return true or false based on whether the tuple is going to satisfy a boolean equation based on the output, a user would issue the following query:

```
SELECT location, avg_speed,
    probabilities('vwWndPwr', 'wndout>8600', wind_id);
FROM vwWndPwr;
```

To return the probability that the tuple is going to be in the top ten locations based on the wind output, the following query would be used:

```
SELECT location, avg_speed, rank, mean
FROM vwWndPwr w
JOIN topk('wndout', 'vwWndPwr') tk
    ON tk.tablepk = w.wind_id
LIMIT 10;
```
A user can also do more complex queries combining these functions. For instance, using a trivial simulation for power generation, a user can create a simple view that defines the stochastic attribute `power_out` and `power_availability`:

```
CREATE OR REPLACE VIEW PowerRequirements AS

SELECT cl.location, t.tubine, t.average_power,
     powerSimulation (t.average_power) AS power_out,
     power_availability
FROM cand_locations cl, turbines t
WHERE cl.location <> 'GEORGIA';
```

Using the new functions defined in SimQL and returning to the fictional university of Camford, suppose we want to solve the following query:

Find the list of location and turbine model pairs with their average wind speeds where the wind speed is a random variable. Rank the locations in decreasing order of their expected power generation (in kilowatts) where power generation is a random variable that is dependent on the wind speed at the location. Return the 10 top ranked locations where the probability that the expected generated power with 80% probability, that the availability of power is at least 45%.

Using the view `PowerRequirements`, the user can create the following query:

```
SELECT p.location, p.turbine, p.average_speed,
     EXP(p.power_out), EXP(p.power_availability)
```
FROM PowerRequirements p
WHERE PROB(p.power_availability>=.45) >= .8
ORDER BY EXP(p.power_out) DESC
LIMIT 10;

A generalized form for the stochastic top-k query would be:

CREATE OR REPLACE VIEW stochastic_view AS

SELECT RA1, …, RAM
Simulation (simulation_attributes) AS SA1, …, SAR
FROM T1 t1, …, TM tm
WHERE regular_attribute_condition

WHERE RA is a regular attribute and SA is a stochastic attribute. The tables are regular relational tables and Simulation is a function that takes regular attributes from a tuple as input and returns a random variable of type reals defined by simulation.

SELECT stoch_expr1 AS SA1, …, stoch_exprw AS SAw,
RA1, …, RAM
FROM stochastic_view sv
WHERE stochastic_attr_condition
ORDER BY EXP(SA1)
LIMIT k;
The functions ARRANGE_PAX, SMRGOL, BRYAGH were created to support flight schedule analysis. All three functions can only be used as part of a `SELECT` statement. ARRANGE_PAX places passengers who miss their flight on the next available flight. SMRGOL and BRYAGH look at delaying a flight till one or more passengers have arrived before rescheduling them on the next available flight. All three functions return a new flight schedule and the PTD for that schedule. These three functions require the tables described in Table 23, Table 24, Table 25, and Table 29. A query for returning a flight schedule where all passengers who missed a flight will be placed on the next flight would be:

```
SELECT arrange_pax();
```

### 3.3. Formal Syntax and Semantics of SimQL

Sim_config is a configuration table that allows the user to change parameters that will affect the simulations in SimSQL. For instance, the user can alter the number of times the simulation is run per tuple. All five functions can run the simulation either a set number of times or until a certain statistical confidence is reached. The entries in this table are described below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sim_Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sim_runs</td>
<td>The number of times the user wants to run the simulation or the number of runs at which the simulations will stop even if the desired statistical confidence has not been reached.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Sim_Value</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Run_type</td>
<td>The type of simulation to be run.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>1</td>
<td>Runs the simulation for the number of times entered in sim_runs regardless of statistical confidence.</td>
</tr>
<tr>
<td>2</td>
<td>Runs the simulation till either the entered variance or the number of simulation runs in sim_runs is reached, whichever occurs first.</td>
</tr>
<tr>
<td>3</td>
<td>Runs the simulation till either the entered standard deviation or the number of simulation runs in sim_runs is reached, whichever occurs first.</td>
</tr>
<tr>
<td>Std_dev</td>
<td>The standard deviation desired for the simulation to stop. If this standard deviation is not reached before the number in sum_runs is reached, the simulation will still stop.</td>
</tr>
<tr>
<td>Variance</td>
<td>The variance desired for the simulation to stop. If this variance is not reached before the number in sum_runs is reached, the simulation will still stop.</td>
</tr>
<tr>
<td>Init_samples</td>
<td>The number of simulation runs to use for each tuple during the initialization phase of algorithms such as OCBA.</td>
</tr>
<tr>
<td>Budget</td>
<td>The total computing budget that can be used for simulation. All algorithms will stop when the budget has been exhausted.</td>
</tr>
<tr>
<td>K</td>
<td>The number of tuples desired for algorithms such as OCBA.</td>
</tr>
<tr>
<td>Probability</td>
<td>The statistical confidence desired before returning the results from an algorithm such as OCBA.</td>
</tr>
</tbody>
</table>
Table 2 Functions and Signatures

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP</td>
<td>( EXP(\text{view_name, stochastic_attribute_name, view_primary_key}) ) Returns double precision</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>( VARIANCE(\text{view_name, stochastic_attribute_name, view_primary_key}) ) Returns double precision</td>
</tr>
<tr>
<td>PROB</td>
<td>( PROB(\text{view_name, inequality_expression, view_primary_key}) ) Returns double precision</td>
</tr>
<tr>
<td>PROBABILITY</td>
<td>( PROBABILITY(\text{view_name, inequality_expression, view_primary_key}) ) Returns boolean</td>
</tr>
<tr>
<td>TOPK</td>
<td>( TOPK(\text{stochastic_attribute_name, view_name}) ) Returns table</td>
</tr>
<tr>
<td>TOPG</td>
<td>( TOPG(\text{stochastic_attribute_name, view_name}) ) Returns table</td>
</tr>
<tr>
<td>TOPG2</td>
<td>( TOPG2(\text{stochastic_attribute_name, view_name}) ) Returns table</td>
</tr>
<tr>
<td>ARRANGE_PAX</td>
<td>( ARRANGE_PAX () ) Returns table, integer</td>
</tr>
<tr>
<td>SMRGOL</td>
<td>( SMRGOL () ) Returns table, integer</td>
</tr>
<tr>
<td>BRYAGH</td>
<td>( BRYAGH () ) Returns table, integer</td>
</tr>
</tbody>
</table>
The following is a table of the terminal symbols used for describing the syntax of SimSQL. All but stochastic_attribute_name and user_defined_function should be familiar to SQL developers.

Table 3 Terminal Symbols for SimSQL

<table>
<thead>
<tr>
<th>Terminal Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias_name</td>
<td>Alias for the table or view</td>
</tr>
<tr>
<td>column_name</td>
<td>Name of a column defined in a table or view</td>
</tr>
<tr>
<td>comparison_operator</td>
<td>Standard operators for comparison (&gt;, &lt;, ≤, =, ≥, ≤)</td>
</tr>
<tr>
<td>inequality_expression</td>
<td>A string of one or more comparison expressions</td>
</tr>
<tr>
<td>literal</td>
<td>A fixed value such as a string or integer</td>
</tr>
<tr>
<td>operator</td>
<td>Standard operators for value manipulation; (+, -, ×, ÷)</td>
</tr>
<tr>
<td>stochastic_attribute_name</td>
<td>Name of a stochastic attribute that is defined in a view</td>
</tr>
<tr>
<td>table_name</td>
<td>Name of a table defined in the database</td>
</tr>
<tr>
<td>user_defined_function</td>
<td>Function created by the user in the database that returns type double precision. This function can be a distribution function such as Gaussian or Poisson.</td>
</tr>
<tr>
<td>view_name</td>
<td>Name of a view defined in the database</td>
</tr>
<tr>
<td>view_primary_key</td>
<td>Primary key for the view</td>
</tr>
</tbody>
</table>

Table 5 beginning with the start symbols QUERY and VIEW. Terminals are listed in the Table 2, non-terminal are upper case strings, SQL specific keywords are in
bold, and SimSQL specific keywords are in bold italics. Table 4 focuses on the semantics of a SimSQL query.

Table 5 focuses on the semantics of a view that is used to define one or more stochastic attributes.

Table 4 Syntax of SimSQL Queries

<table>
<thead>
<tr>
<th>Non-Terminal</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY</td>
<td>SELECT [ALL</td>
</tr>
<tr>
<td></td>
<td>FROM_CLAUSE [WHERE_CLAUSE]</td>
</tr>
<tr>
<td>SELECT_LIST</td>
<td>[ATTRIBUTE_LIST][AGGREGATE_FUNCTION]</td>
</tr>
<tr>
<td></td>
<td>[STOCHASTIC_FUNCTION]</td>
</tr>
<tr>
<td>ATTRIBUTE_LIST</td>
<td>list of ATTRIBUTE</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>alias_name.column_name</td>
</tr>
<tr>
<td>AGGREGATE_FUNCTION</td>
<td>AGG_FUNCTION_NAME ([ALL][DISTINCT]</td>
</tr>
<tr>
<td></td>
<td>alias_name.column_name</td>
</tr>
<tr>
<td>AGG_FUNCTION_NAME</td>
<td>COUNT</td>
</tr>
<tr>
<td>STOCHASTIC_FUNCTION</td>
<td>STOCHASTIC_FUNCTION_NAME (view_name,</td>
</tr>
<tr>
<td></td>
<td>alias_name.stochastic_attribute_name,</td>
</tr>
<tr>
<td></td>
<td>view_primary_key)</td>
</tr>
<tr>
<td>STOCHASTIC_FUNCTION_NAME</td>
<td>EXP</td>
</tr>
<tr>
<td>FROM_CLAUSE</td>
<td>FROM VIEW_REFERENCE [, VIEW_REFERENCE]</td>
</tr>
</tbody>
</table>
### Table 5 Syntax of SimSQL Views

<table>
<thead>
<tr>
<th>Non-Terminal</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE VIEW</td>
<td>view_name AS SELECT</td>
</tr>
<tr>
<td>VIEW</td>
<td>CREATE VIEW view_name AS SELECT</td>
</tr>
</tbody>
</table>

Non-Terminal | Production
--- | ---
[.TABLE_REFERENCE][.TOPK][.TOPG 2] | view_name alias_name
TABLE_REFERENCE | table_name alias_name
WHERE_CLAUSE   | WHERE WHERE_CONDITIONS
WHERE_CONDITIONS | WHERE_CONDITION [AND|OR] WHERE_CONDITION
WHERE_CONDITION | ATTRIBUTE comparison_operator VALUE|
| PROBABILITY_FUNCTION | comparison_operator VALUE
| PROBABILITY_FUNCTION_NAME | PROBABILITY_FUNCTION_NAME (VIEW REFERENCE, inequality_expression, view_primary_key)
| PROBABILITY_FUNCTION NAME | PROB|PROBABILITY
<table>
<thead>
<tr>
<th>Non-Terminal</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT_LIST</td>
<td>SELECT_LIST FROM_CLAUSE</td>
</tr>
<tr>
<td>ATTRIBUTE_LIST</td>
<td>[ATTRIBUTE_LIST][AGGREGATE_FUNCTION] [STOCHASTIC_ATTR]</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>alias_name column_name</td>
</tr>
<tr>
<td>AGGREGATE_FUNCTION</td>
<td>AGG_FUNCTION_NAME ([ALL</td>
</tr>
<tr>
<td>AGG_FUNCTION_NAME</td>
<td>COUNT</td>
</tr>
<tr>
<td>CASE_CONDITIONS</td>
<td>WHEN CASE_CONDITION THEN</td>
</tr>
<tr>
<td>CASE_CONDITION</td>
<td>alias_name.column_name comparison_operator VALUE [AND</td>
</tr>
<tr>
<td>VALUE</td>
<td>Literal</td>
</tr>
<tr>
<td>STOCHASTIC_VAR_DEFINIT</td>
<td>VALUE operator user_defined_function</td>
</tr>
</tbody>
</table>

|

32
<table>
<thead>
<tr>
<th>Non-Terminal</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>ION</td>
<td>user_defined_function</td>
</tr>
<tr>
<td>FROM_CLAUSE</td>
<td>TABLE_REFERENCE</td>
</tr>
<tr>
<td>TABLE_REFERENCE</td>
<td>table_name alias_name</td>
</tr>
</tbody>
</table>
4. ALGORITHMS FOR TOP-K SIMQL QUERIES BASED ON ENTIRE SEARCH SPACE EXPLORATION AND SIMULATION BUDGET OPTIMIZATION

Using the generalized stochastic top-k query from the previous section we can partially materialize the view

Partially_materialized_table:

\[
\text{SELECT } R_{A_1}, \ldots, R_{A_z} \\
\text{FROM } T_1 \ t_1, \ldots, T_m \ t_m \\
\text{WHERE regular_attribute_condition}
\]

so that the query being processed by SimQL becomes:

stochastic_query (PST):

\[
\text{SELECT Simulation (PMT.Sim_attributes) AS } S_{A_t}, \ldots, S_{A_r}, \\
R_{A_1}, \ldots, R_{A_z} \\
\text{FROM Partially_materialized_table PMT} \\
\text{WHERE stochastic_attr_condition} \\
\text{ORDER BY EXP(S_{A_z})} \\
\text{LIMIT k;}
\]

By partially materializing the view, all of the possible inputs for the simulation are gained.
4.1. **Top K Algorithm Template (TKAT)**

**Input:**

PMT – partial materialized table

Simulation – simulation defined in the stochastic view

K – Number of top tuples to return

PS – Desired probability of correct selection

B – Available simulation budget

iteration_budget – the simulation budget to be spent on each iteration

**Output:**

Regular relational Table containing:

1. Simulated mean for stochastic attribute
2. Probability of correct selection of the top k tuples

**Structures:**

B – portion of the budget consumed so far

num_tuples – number of tuples

tupMean – The mean of the stochastic value as simulated so far

tupBudget – The simulation budget used on the tuple so far

tupStandDev – The standard deviation of the stochastic value for the tuple

per_tuple_budget – Simulation budget for the tuple for the iteration

PS – the statistical confidence that the current top-k selection is correct
Initialize:

per_tuple_budget = iteration_budget/num_tuples;

budget_consumed = iteration_budget;

/* Each tuple (S_1, S_2, ..., S_t) is sampled (i.e., simulation run on it) with a CONSTANT computational budget (which is part of C) */

For Each tuple In PMT Do

   For j = 1 To per_tuple_budget Do

      temp_sim[j] = Simulation(tuple);

   End For;

   tupMean[tuple] = Σ_{x=1}^j temp_sim[x]/per_tuple_budget;

   tupBudget[tuple] = per_tuple_budget;

   tupStandDev[tuple] = Σ_{x=1}^j (tupMean[tuple] − temp_sim[x])²/per_tuple_budget

End For;

Do

simBudgetTable = Assign_Budget (PMT, iteration_budget, tupMean, tupStandDev);

For Each tuple In PMT Do

   For j = 1 To simBudgetTable[tuple] Do

      temp_sim[j] = Simulation(tuple);

   End For;

   tupMean[tuple] = \frac{(tupMean[tuple] \times tupBudget[tuple]) + Σ_{x=1}^j temp_sim}{simBudgetTable[tuple] + tupBudget[tuple]};

End For;
tupStandDev[tuple] = tupStandDev[tuple] x tupBudget [tuple]

tupBudget [tuple]= tupBudget [tuple] + simBudgetTable[tuple];

tupStandDev[tuple] = (tupStandDev[tuple] + 
\[\sum_{x=1}^{i} (tupMean[tuple] − temp_sim[x])^2\]) / tupBudget [tuple]

End For;

SELECT top k tuples;

PCS = Calculate_PCS ();

budget_consumed = budget_consumed + iteration_budget ;

While budget_consumed < B And PCS < PS;

Return PCS, top k tuples;

4.2. Implementation of Monte Carlo (a.k.a. Equal Opportunity) in SimQL

Monte Carlo is “particularly useful for path-dependent options and others for which no known formula exists.” [20] The Monte Carlo algorithm allocates an equal portion of computational budget to each tuple until the budget has been exhausted. The stopping criterion for this algorithm is when the budget has been exhausted. The probability of correct selection cannot be computed till the algorithm is completed.

The Monte Carlo implementation in SimQL follows the TKAT template from above with the Assign_Budget function defined below:
Table 6 Assign_Budget for Monte Carlo Algorithm

<table>
<thead>
<tr>
<th>Input:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PMT – Partial materialized table</td>
<td></td>
</tr>
<tr>
<td>iteration_budget – The simulation budget to be spent on each iteration</td>
<td></td>
</tr>
<tr>
<td>tupMean – The mean of the stochastic value as simulated so far</td>
<td></td>
</tr>
<tr>
<td>tupStandDev – The standard deviation of the stochastic value as simulated so far</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>simBudgetArray – The array with the simulation budget for each tuple for the next iteration</td>
<td></td>
</tr>
</tbody>
</table>

For Each tuple In tupMean Do

```
simBudgetArray [tuple] = iteration_budget/num_tuples;
```

End For;

Return simBudgetArray;

4.3. Implementation of OCBA in SimQL

The Monte Carlo algorithm [4] does not allocate the simulation budget to the tuples effectively nor does it stop until the entire simulation budget has been exhausted. Consequently, tuples that do not stand a chance of being in the top-k get the same amount of budget as the tuples that may be in the top-k. The OCBA addresses both of these limitations. Following the TKAT template, the assign_budget function is described in Table 7.
### Table 7 Assign_Budget for OCBA Algorithm

<table>
<thead>
<tr>
<th>PMT – Partial materialized table</th>
</tr>
</thead>
<tbody>
<tr>
<td>iteration_budget – The simulation budget to be spent on each iteration</td>
</tr>
<tr>
<td>tupMean – The mean of the stochastic value as simulated so far</td>
</tr>
<tr>
<td>tupStandDev – The standard deviation of the stochastic value as simulated so far</td>
</tr>
</tbody>
</table>

**Output:**

simBudgetArray – The array with the simulation budget for each tuple for the next iteration

\[ b = \left(\frac{tupMean[k] + tupMean[k + 1]}{2}\right)^2 \]

**For Each** tuple **In** tupMean **Do**

\[
\text{simBudgetArray [tuple]} = \frac{\text{tupStandDev [tuple]^2}}{\text{num_tuples}} \times \text{iteration_budget;}
\]

**End For:**

**Return** simBudgetArray;
5. ALGORITHM(S) FOR TOP-K QUERIES BASED ON PARTIAL SEARCH SPACE EXPLORATION FOR CONTINUOUS DECISION VARIABLES BASED ON REGRESSION ANALYSIS HEURISTICS

5.1. An Efficient Regression Based Algorithm for Top-K Selection

Unfortunately, OCBA requires an initialization phase where a portion of the budget must be run on every tuple. Even with the best scenario where OCBA reaches a PCS of the desired probability after the initialization phase, OCBA would still require $n \times t_0$ simulation runs where $n = \text{number of tuples and } t_0 = \text{sample budget}$.

This can become prohibitive when dealing with large numbers of tuples. Because the stochastic simulation can be time consuming and the number of tuples very large, running the Initialization phase of OCBA for each tuple is expensive.

Suppose the user were willing to possibly sacrifice the exact answer and be satisfied with a close enough answer if it considerably improved performance. One way to accomplish this is to run the Initialization phase on a small sample of the tuples and then use regression analysis to estimate the stochastic attribute for the tuples not in the sample set instead of running it on every tuple. I am assuming that regression analysis is cheaper than the additional simulation runs. Then, to further budget compression, choose a subset of the tuples based on the mean of the stochastic attribute, the top Q, where Q is very small in comparison to the total number of designs, but is large enough to not miss
the top k are sent into the iterative phase of OCBA. The stopping criteria are still either when the desired PCS is reached or the simulation budget has been exhausted.

Figure 2 GORBASH Top Q Selection

Though the probability of correct selection for the subset of tuples run through OCBA will still have the same bound as OCBA, it is necessary to consider the probability that the top-k were contained in Q. Since the two events are independent, the probability of correct selection of k is:

**Equation 1: Probability of Correct Selection for GORBASH**

$$PCS = P(S_k \subseteq S_Q) \times P(\text{top } k \text{ selected from } Q)$$

where $S_k = \text{set of } k \text{ tuples and } S_Q = \text{set of } Q \text{ tuples}$.

**Claim:** $P(S_k \subseteq S_Q) \geq \left[P(v_k - v_{Q+1} \geq 0)\right]^{n-Q}$. 

41
where $v_i = mean$ for tuple $i$,

$n = total$ number of tuples,

and $0 \leq k \leq Q \leq n$.

**Small Proof**: Assume $k \leq Q$.

$$P(v_1, ..., v_Q$ contains top $k)$$

$$\geq P\left(\bigwedge_{j=Q+1}^{n} \left(v_k - v_j \geq 0\right)\right)$$

$$= \prod_{j=Q+1}^{n} P(v_k - v_j \geq 0)$$

$$\geq [P(v_k - v_{Q+1} \geq 0)]^{n-Q}.$$

GORBASH follows the template of TKAT and is described in Table 8.

---

**Table 8 GORBASH Algorithm**

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>C – Total computation budget,</td>
</tr>
<tr>
<td>DB – A budget delta, i.e., constant part of the budget to be used in a single iteration,</td>
</tr>
<tr>
<td>$P_o$ – Desired statistical confidence (e.g., 95%) that the top-$k$ selection is correct,</td>
</tr>
<tr>
<td>K – Desired number of tuples for selection,</td>
</tr>
<tr>
<td>N – Number of tuples to sample during initialization,</td>
</tr>
<tr>
<td>$t_0$ – Initialization budget for each tuple</td>
</tr>
</tbody>
</table>
Structures:

B – Portion of the budget consumed so far

T – Relation, set of tuples

Q – Number of designs to choose for OCBA after initialization,

P_B – Statistical confidence that the current top-k selection is correct

P_T – Statistical confidence that the tuple is in the top-k

Phase I:

Step 1: Sample N tuples for \( t_0 \)

Step 2: Run the simulation for each of the N tuples M times and estimate value of outcome B.

Step 3: Use regression analysis to express the utility stochastic attribute \( U \) as a function \( F \):

\[
U = F(p, A_1, ..., A_g)
\]

Step 4: Make estimates for other samples using the regression parameters.

Phase II:

Step 5: Compute \( Q = \) the number of tuples where the probability of the top k being contained is

\[
P_T = \sqrt{P_D}
\]

Step 6: Select the min Q tuples.
Phase III:

While $P_B \times P_T < P_D$ and $B < C$ Loop

Step 7: Allocate the delta budget $DB$ to each tuple proportional to:

$$\text{If } n = 0 \text{ Then } \epsilon_0$$

$$\text{Else } \frac{\text{variance}_n}{(b-\text{mean})^2}$$

Step 8: Run simulations for each tuple within allocated budget.

Step 9: Re-compute $P_B$ and $B$.

End Loop

Step 10: Return the top-k tuples and $P_B \times P_T$.

5.2. An Efficient Iterative Regression Based Algorithm for Top-K Selection

Whereas GORBASH spends the remaining simulation budget on a subset of tuples for OCBA, $Q$, GORBASH+ takes a more iterative approach. During each iteration of Phase III, GORBASH+ computes if the PCS would increase more if the simulation budget for that iteration was spent on an additional tuple, $Q+1$, or if the simulation budget was spent on the existing $Q$ tuples.
Table 9 GORBASH+ Algorithm

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>C – Total computation budget,</td>
</tr>
<tr>
<td>DB – A budget delta, i.e., constant part of the budget to be used in a single iteration,</td>
</tr>
<tr>
<td>( P_D ) – Desired statistical confidence (e.g., 95%) that the top-k selection is correct,</td>
</tr>
<tr>
<td>K – Desired number of tuples for selection,</td>
</tr>
<tr>
<td>N – Number of tuples to sample during initialization,</td>
</tr>
<tr>
<td>( t_0 ) – Initialization budget for each tuple</td>
</tr>
<tr>
<td>Q – Number of designs to choose for OCBA after initialization</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Structures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>B – Portion of the budget consumed so far</td>
</tr>
<tr>
<td>T – Relation, set of tuples</td>
</tr>
<tr>
<td>( P_B ) – Statistical confidence that the current top-k selection is correct</td>
</tr>
<tr>
<td>( \hat{P}_B ) – Estimated statistical confidence that the current top-k selection will be correct after additional simulations</td>
</tr>
<tr>
<td>( P_T ) – Statistical confidence that the current Q tuples contain the top-k</td>
</tr>
<tr>
<td>( \hat{P}_T ) – Statistical confidence that the current Q+1 tuples contain the top-k</td>
</tr>
</tbody>
</table>

**Phase I:**

- **Step 1:** Sample \( N \) tuples for \( t_0 \)
- **Step 2:** Use regression to find parameters of function.
- **Step 3:** Make estimates for other samples using the regression parameters.
Phase II:

Step 4: Compute $Q$ = the number of tuples where the probability of the top $k$ being contained is

$$P_T = \sqrt{P_D}$$

Step 5: Select the min $Q$ tuples.

Phase III:

While $P_B \times P_T < P_D$ and $B < C$ Loop

Step 6: Compute $\hat{P}_T$ for min $Q+1$ tuples

Step 7: Compute $\hat{P}_B$ for $Q$ tuples

If $\hat{P}_T \times P_B > \hat{P}_B \times P_T$ Then

Step 8: Run the simulation for the $Q+1$ tuple for $t_0$.

Step 9: Re-compute $P_T$.

Step 10: $Q=Q+1$

Else

Step 8: Allocate the delta budget $DB$ to each tuple proportional to:

If $n = 0$ Then $t_0$

Else $\frac{\text{variance}^2/n}{(b-\text{mean})^2}$

Step 9: Run simulations for each tuple within allocated budget.

Step 10: Re-compute $P_B$.

End If
Step 11: Re-compute B.

End Loop

Step 12: Return the top-k tuples and $P_B \times P_T$. 
6. ALGORITHMS FOR TOP-K SCHEDULING QUERIES BASED ON PARTIAL SEARCH SPACE FOR DISCRETE DECISION VARIABLES BASED IN ENUMERATION HEURISTICS

6.1. Deterministic Scheduling Problem and Algorithm

One important class of top-k problems that can be represented and analyzed with SimQL is scheduling problems. With this class of problems, there is a set of tasks where some tasks are dependent on the completion of one or more other tasks. Tasks can be delayed because of the task taking longer than expected to complete, mechanical problems, supply problems, or some other uncontrollable issue. If one or more tasks are not completed in time, the next task cannot begin and there is a ripple effect in the schedule rendering the schedule invalid.

For this class of problems, I assume that I have an existing schedule that is no longer feasible. I also assume that I have a current “snapshot” or state of the system at the point when the scheduling problem has occurred as well as the list of tasks and their dependencies. In Table 10, the definitions of the different elements of the problem are described in further detail.
Table 10 Elements of the Scheduling Problem.

<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
<th>Formal Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time ($T$)</td>
<td>The time period (beginning and end) for the schedule.</td>
<td>$T = [0, TH]$ where $TH$ is Time Horizon, is a time interval for scheduling.</td>
</tr>
<tr>
<td>Tasks ($F$)</td>
<td>A set of tasks that are part of the schedule. There is a unique task id associated with each task. An example of a task is to move 30 widgets from a shipping point in Huntsville to a shipping point in Atlanta.</td>
<td>$F = {1, 2, 3, \cdots, n}$ is a set of tasks where $n$ is total number of tasks.</td>
</tr>
</tbody>
</table>
| Activities ($A$) | A task with the start times and duration associated with the task. For instance, a flight from NYC to WCA may occur at several times during the day. | A set of activities where each activity $a \in A$ is a tuple where $a = (id, t, st, d, lc)$ where $id$ is the identity of the activity, $t$ is the task in $F$ the activity accomplishes, $st$ is a start time where $st \in T$, $d$ is the duration of the activity, and $lc$ is the load capacity of the activity. We denote id of activity $a$ by $ID(a)$, start time $st$ of activity $a$ by $B(a)$, duration $d$ by $D(a)$, task $t$ by $task(a)$, and load capacity $lc$ by $LC(a)$.

49
<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
<th>Formal Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite Activities (CA)</td>
<td>Every schedule has a set of composite activities which has a load, $L$, (e.g., the number of passengers or weight of cargo) and a sequence of one or more activities with some connection between the activities.</td>
<td>$CA$ is a set of composite activities where each composite activity $ca \in CA$ is a pair $\langle L_i, (a_1, \ldots, a_k) \rangle$ where $a_1, \ldots, a_k$ is a series of activities in $A$ and $L_i$ is the load of the composite activity, such that $(\forall i = 1, \ldots, k) L_i \leq LC(a_i)$ where $LC(a_i)$ is the load capacity of $a_i$.</td>
</tr>
<tr>
<td>Hard Dependency (HD)</td>
<td>Based on timing, this is the inter-dependencies of activities where an activity is dependent on the same physical resource used by another activity. Therefore, the work on the activity cannot begin until the completion of the previous activity. For example, if truck 1 has to be used to move widgets to Atlanta and then to move cogs to Denver, the work on the activity cannot begin until the completion of the previous activity.</td>
<td>A hard constraint is an expression of the form: $i &lt;_{\Delta} j$ where $i, j \in A$ and $\Delta \in \mathbb{R}$ is a time delay after the completion of $i$ until the start of $j$. We denote a set of hard constraints by HD.</td>
</tr>
<tr>
<td>Elements</td>
<td>Description</td>
<td>Formal Definition</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>cogs to Denver delivery cannot begin till the truck returns from the widget delivery to Atlanta.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schedule ((S))</td>
<td>The set of tasks, time needed for the tasks to be completed so that the next dependent task can begin, and the hard dependencies of the set of tasks.</td>
<td>A schedule is a tuple ((T, F, A, CA, HD)) where (T) is the time interval of the schedule, (F) is a set of tasks, (A) is the set of activities with tasks from (F), (CA) is the set of composite activities with activities from (A), and (HD) is the set of hard dependencies associated with the set of activities (A).</td>
</tr>
</tbody>
</table>

**Definition 1**  
A soft dependency \(SD(S)\) is an expression of the form  
\[
i <_{\delta} j, \text{ where } i, j \in A \text{ and } \delta \in \mathbb{R} \text{ is a delay after the completion of } i \text{ before the start of } j.\]  
Given a schedule \(S = \langle T, F, A, CA, HD \rangle\), the set of soft dependencies \(SD(S)\) associated with \(S\) is defined as follows:  
\[
SD(S) = \left\{ i <_{\delta} j \mid i, j \in A \text{ and } \exists \text{ a composite activity } ca = \langle L, (i_1, \ldots, i_j, \ldots, i_k) \rangle \in CA \right\}
\]

**Definition 2**  
Given a schedule \(S = \langle T, F, A, CA, HD \rangle\), we say that it satisfies the hard dependencies \(HD\) if,  
\[
\forall i <_{\Delta} j \in HD, B(i) + D(i) + \Delta \geq B(j), \text{ where } \Delta \text{ is a }
\]
scheduling constant representing preparation time needed before the next task can
begin.

**Definition 3** Given a schedule \( S = \langle T, F, A, CA, HD \rangle \), we say that it satisfies the soft
dependencies \( SD(S) \) if, \( \forall i < j \in SD, B(i) + D(i) + \delta \leq B(j) \) where \( \delta \) is a
scheduling constant representing preparation time needed before the next task can
begin.

**Definition 4** We say that a given schedule \( S = \langle T, F, A, CA, HD \rangle \) is feasible if it satisfies
the hard constraints \( HD \) and the soft constraints \( SD(S) \).

**Definition 5** Composite task, denoted \( ct(ca) \), associated with a composite activity
\( ca = (a_1, \cdots, a_k) \in CA \) is \( (t_1, \cdots, t_k) \) where \( t_1 = task(a_1), \cdots, t_k = task(a_k) \). The
set of composite tasks \( CT(CA) \) associated with the set of composite activities \( CA \) is
defined as \( CT(CA) = \{ct(ca) | ca \in CA\} \).

**Definition 6** For a composite task \( ct = (t_1, \cdots, t_k) \), where \( t_1, \cdots, t_k \in F \), and \( CA \) is a set
of composite activities, we define the load \( L(ct, CA) \) as follows:

\[ L(ct, CA) = \sum_{ca\in CA} (L(ca)) \]

where \( L(ca) \) is the load \( L \) of the composite activity
\( ca = \langle L, (a_1, \cdots, a_k) \rangle \) and

\[ CA' = \{ca | ca = \langle L, (a_1, \cdots, a_k) \rangle \in CA \land (task(a_1) = t_1 \land \cdots \land task(a_k) = t_k)\} \]

(i.e., \( CA' \) is the set of all composite activities in \( CA \) that perform the composite task
\( ct \)).

**Definition 7** Let \( S = \langle T, F, A, CA, HD \rangle \) be a feasible schedule. The weighted composite
time of \( S \), denoted \( WCT(S) \), is defined as:
\[
WCT(S) = \sum_{ca \in CA} (L(ca) \times D(ca))
\]

where \(L(ca)\) is the load \(L\) of the composite activity \(ca = (L, (a_1, \ldots, a_k))\)

and \(D(ca)\) is the duration of \(ca\) defined as \(D(ca) = B(a_k) + D(a_k) - B(a_1)\). The load normalized composite time, denoted \(LNCT(S)\), is the normalized completion time defined as:

\[
LNCT(S) = \frac{WCT(S)}{\sum_{ca \in CA} L(ca)}
\]

where \(L(ca)\) is the load \(L\) of the composite activity \(ca = (L, (a_1, \ldots, a_k))\).

**Definition 8** Let \(S_1 = (T, F, A_1, CA_1, HD_1)\) be a feasible schedule. We say that \(S_2 = (T, F, A_2, CA_2, HD_2)\) is a delay of schedule \(S_1\) if \(S_2\) is feasible and there exists a mapping \(m: A_1 \rightarrow A_2\) such that:

1. \(m\) is a one-to-one and onto,
2. \(HD_2 = \{m(i) \prec_\Delta m(j) \mid i \prec_\Delta j \in HD_1\}\) (i.e., \(m\) preserves hard dependencies), and
3. \(\forall a \in A_1, B(m(a)) \geq B(a)\) (i.e., \(S_2\) can only “delay” activities).

We say that \(S_2\) is task-load equivalent delay of \(S_1\) if:

1. \(S_2\) is a delay of \(S_1\),
2. \(CT(CA_1) = CT(CA_2)\) (i.e., \(S_1\) and \(S_2\) have the same set of composite tasks), and
3. \(\forall ct \in CT, L(ct, CA_1) = L(ct, CA_2)\).

With these elements, the problem then becomes (Table 11): given an "original" schedule, \(S_{curr}\), where an event or delay has occurred that makes the schedule no longer
feasible, find an updated schedule, $S_{new}$, that is task-load equivalent to $S_{curr}$ that minimizes the load normalized composite time.

The expected delay of $S_{curr}$ will be computed by rescheduling the loads that have been delayed to the next available activity that meets the requirements of the one missed. As a result of the delays, if part of a load cannot make it to the next activity when the activity is supposed to begin, the partial load will be scheduled to the next activity that has available space and that matches the task of the activity missed. If all of the partial load cannot be assigned to the new activity, the available space will be filled in the new activity and the rest of the load will wait till the next available activity. If part of a load cannot be scheduled by the end of the 24 hour schedule run, then a delay of 900 minutes will be assessed as the delay.

Table 11 Deterministic Scheduling Problem

<table>
<thead>
<tr>
<th>Input:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $S_{curr} = (T, F, A_{curr}, CA_{curr}, HD_{curr})$ – the current feasible schedule such that($\forall f \in F)(\exists a = (id, t, st, d, lc) \in A_{curr})$ such that:</td>
<td></td>
</tr>
<tr>
<td>a. $t = f$</td>
<td></td>
</tr>
<tr>
<td>b. $st = \text{Penalty (a penalty late time)}$</td>
<td></td>
</tr>
<tr>
<td>c. $lc = \infty$</td>
<td></td>
</tr>
<tr>
<td>2. Time – the current time of the system.</td>
<td></td>
</tr>
<tr>
<td>3. $aStatus: A_{curr} \rightarrow {\text{not_started}} \cup {\text{started}} \times T$, i.e., for every activity $a \in A_{curr}, aStatus(a)$ gives a status which is either “not_started” or “started&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Output:

\[ S_{\text{new}} = \langle T, F, A_{\text{new}}, C_{\text{new}}, H_{\text{new}} \rangle \] - a new schedule such that:

1. \( S_{\text{new}} \) is a task-load equivalent delay of \( S_{\text{curr}} \) such that:
   
   a. For every \( a \in A_{\text{curr}} \) if \( \text{aStatus}(a) = \text{"started @ time t"}, t \in T, B(m(a)) = t \) and
   
   b. For every \( a \in A_{\text{curr}}, \) if \( \text{aStatus}(a) = \text{"not_started"} \) then \( B(m(a)) \geq Time \).

2. \( \forall S \in S_{\text{Total}}, \) where \( S_{\text{Total}} \) is the set of all new possible schedules, if \( S \) satisfies (1), then \( \text{LNCT}(S) \geq \text{LNCT}(S_{\text{new}}) \) (i.e., \( S_{\text{new}} \) is a schedule with minimal \( \text{LNCT} \) among schedules that satisfy (1)).

---

**Table 12 DALEXS: Deterministic Algorithm for Load and machine reScheduling**

**Input:** Same as the input for the deterministic scheduling problem in Table 11

**Output:** Same as the output for the deterministic scheduling problem in Table 11, but does not guarantee number 2 for optimality.

**Structures:**
(1) **ActivityQueue** – sequence of activities in $S$ ordered by $B(a)$

(2) **NewActivityQueue** – sequence of new activities in $S_{new}$

/* Enforce all hard dependencies*/

$$\text{NewActivityQueue} = \text{HardDependencyReschedule}(S, \text{Time}, a\text{Status});$$

/* Enforce all soft dependencies*/

$$S_{new} = \langle T, F, \text{NewActivityQueue}, CA_{curr}, HD_{curr}\rangle;$$

$$S_{new} = \text{SoftDependencyReschedule}(S_{new});$$

---

**Table 13 HardDependencyReschedule**

**Input:**

1. $S_{curr} = \langle T, F, A_{curr}, CA_{curr}, HD_{curr}\rangle$ – the current schedule such that $(\forall f \in F)(\exists a = \langle id, t, st, d, lc\rangle \in A_{curr}$ such that:
   a. $t = f$
   b. $st = \text{Penalty}$ (a penalty late time)
   c. $lc = \infty$

2. **Time** – the current time of the system.

3. **aStatus**: $A_{curr} \rightarrow \{\text{not started}\} \cup \{\text{started} \times T\},$ i.e., for every activity $a \in A_{curr}$,
   $a\text{Status}(a)$ gives a status which is either “not started” or “started @ time $t$, $t \in T$” such that all started activities satisfy hard and soft constraints of $S_{curr}$.

**Output:**
NewActivityQueue – sequence of new activities in $S_{new}$

Structures:

ActivityQueue – sequence of activities in $S$ ordered by $B(a)$

For $a = (id, t, st, d, lc)$ In ActivityQueue Do

/* All activities are examined in order of increasing start time*/

If $aStatus(a) = "$ started @ time start_t", start_t ∈ T$ Then

/* The activity has already started, the start time cannot be changed.*/

$a_{new} = (id, t, start_t, d, lc);$

$m(a) = a_{new};$

Insert $a_{new}$ Into NewActivityQueue;

Else If $aStatus(a) = "not_started"$ Then

/* For all activities that have not been started, the activities are examined in order of increasing start time*/

StartTime

$= \max \left( \{B(a)\} \right)$

$\cup \left\{ st + d + \Delta \left| \left( \exists (a_p \prec_{\Delta} a) \in HD \right) \land \left( a_p = (id, t, st, d, lc) \in NewActivityQueue \right) \right. \right\};$

$a_{new} = (id, t, StartTime, d, lc);$

$m(a) = a_{new};$

Insert $a_{new}$ Into NewActivityQueue;

End If;

End For;
Table 14 SoftDependencyReschedule

**Input:** \( S = \langle T, F, A, CA, HD \rangle \)

**Output:** \( S = \langle T, F, A, CA, HD \rangle \) - a new schedule where we denote \( CA \) by \( S.CA \)

/* While there is a composite activity that violates a soft constraint.*/

\[ \textbf{While} \ (\exists ca = \langle L, (a_1, a_2, \ldots, a_k) \rangle \in S.CA)(\exists i, 2 \leq i \leq k) \textbf{ such that } B(a_i) < B(a_{i-1}) + D(a_{i-1}) + \delta \textbf{ Do} \]

Let \( ca \) be any composite activity that satisfies the while condition.

Remove \( ca \) From \( S.CA \); /* Remove the ca from CA*/

Let \( x = i \) be the minimal \( i, 2 \leq i \leq k \), such that \( B(a_x) < B(a_{x-1}) + D(a_{x-1}) + \delta \)

\( \text{LoadToReschedule} = L; /* The load of ca that needs to be rescheduled*/ \)

/* List of activities that occur after the current activity and accomplish the same task.*/

\( \text{SameTaskActivities} = \{ b \in S.CA \wedge \text{task}(b) = \text{task}(a_x) \} ; \)

/* Loop through all of the available activities and schedule the load where there is room.*/

\( \text{For Every } b = (id, t, st, d, lc) \in \text{SameTaskActivities Ordered By st Do} \)

\[ \text{If LoadToReschedule} > 0 \text{ Then} \]

\[ \text{currentLoad} = \sum_{ca \in CA} L(ca); /* load of the activity*/ \]

\[ \text{If currentLoad} < lc \text{ Then} \]
Note that Table 12 shows a heuristic algorithm and does not guarantee optimality. One way to decrease the LNCT is to delay an activity till the previous activity has completed. The algorithm shown in Table 15 looks at each soft constraint to determine if it is more efficient to delay the start time of the activity till the previous activity arrives or to reschedule the load of the activity.
Table 15 TARDIS: Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios

**Input:**

1. $S_{curr} = \langle T, F, A_{curr}, CA_{curr}, HD_{curr} \rangle$ – the current feasible schedule such that

   $(\forall f \in F)(\exists a = \langle id, t, st, d, lc \rangle) \in A_{curr}$ such that:
   
   a. $t = f$
   
   b. $st = Penalty$ (a penalty late time)

   c. $lc = \infty$

2. Time – the current time of the system.

3. $aStatus: A_{curr} \rightarrow \{not\_started\} \cup \{started\} \times T$, i.e., for every activity $a \in A_{curr}$, $aStatus(a)$ gives a status which is either “not\_started” or “started @ time t”, $t \in T$ such that all started activities satisfy hard and soft constraints of $S_{curr}$.

**Output:**

$S_{new} = \langle T, F, A_{new}, CA_{new}, HD_{new} \rangle$ – a new feasible schedule

**Structures:**

1. $TempSchedule = \langle T, F, A_{temp}, CA_{temp}, HD_{temp} \rangle$ – The current best schedule. We denote $TempSchedule.CA_{temp} = S_{curr}.CA_{curr}$

2. $NewActivityQueue$ – sequence of new activities in $S_{new}$

3. $TempActivityQueue$

4. $LoadRescheduleSchedule$ – Schedule where the load of the current activity is reschedule

5. $TempSchedule1$ – temporary for saving schedule where missed loads are redistributed

6. $TimeRescheduleSchedule$ – temporary schedule where the activity is delayed till
the previous activity completes.

(7) TempSchedule2 – temporary for saving schedule where the activity is delayed till the previous activity completed instead of redistributing the tardy loads.

/* Ensure hard dependencies are met*/
NewActivityQueue = HardDependencyReschedule(Scurr, Time, aStatus);
TempSchedule = (T, F, A, CA, HS) where A = NewActivityQueue, CA = CA_{curr}, and HD = HD_{curr}; /* may not be a feasible schedule*/

/* Loop through all activities while there is a soft dependency violation. Check to see if it is better to delay the activity till the previous activity arrives or to reschedule the load of the activity to the next available task*/

While (∃ca = (L,(a_1,a_2, ..., a_k)) ∈ TempSchedule.CA)(∃i, 2 ≤ i ≤ k) such that
B(a_i) < B(a_{i-1}) + D(a_{i-1}) + δ Do

Let ca be any composite activity that satisfies the while condition.

Remove ca From CA; /* Remove the ca from CA*/

Let x = i be the minimal i, 2 ≤ i ≤ k, such that B(a_x) < B(a_{x-1}) + D(a_{x-1}) + δ

/*Reschedule the load for this activity.*/
LoadRescheduleSchedule = RescheduleSingleLoad (TempSchedule, ca, x);
TempSchedule1 = SoftDependencyReschedule (LoadRescheduleSchedule);

/*Delay next activity till the previous activity completes. Then enforce the hard and
soft constraints */

\[ TimeRescheduleSchedule = RescheduleActivity (TempSchedule, ca, x) \]

\[ TimeRescheduleSchedule = HardDependencyReschedule (TimeRescheduleSchedule, Time, aStatus); \]

\[ TempSchedule2 = SoftDependencyReschedule (TimeRescheduleSchedule); \]

/* Check to see which new schedule has the best delay*/

\[ \text{If } LNCT (TempSchedule1) < LNCT (TempSchedule2) \text{ Then} \]

\[ TempSchedule = LoadRescheduleSchedule; \]

\[ \text{Else} \]

\[ TempSchedule = TimeRescheduleSchedule; \]

\[ \text{End If}; \]

\[ \text{End While}; \]

---

Table 16 RescheduleSingleLoad

**Input:**

1. \( S = (T, F, A, CA, HD) \), the current schedule
2. \( ca \) — the original composite activity
3. \( x \) — the activity of violation
Output:

\[ TempSchedule_1 = (T, F, A_{temp}, CA_{temp}, HD_{temp}) \] - the new schedule that may violate hard or soft constraints

\[ LoadToReschedule = L; /* The load of ca that needs to be rescheduled */ \]

/* List of activities that occur after the current activity and accomplish the same task */

\[ SameTaskActivities = \{ b \mid b \in A \land \text{task}(b) = \text{task}(a_x) \land B(b) \geq B(a_{x-1}) + D(a_{x-1}) + \delta \}; \]

/* Reschedule load to activities that accomplish the same task and have space */

**For Every** \( b = (id, t, st, d, lc) \in SameTaskActivities **Ordered By** st **Do**

**If** \( LoadToReschedule > 0 \) **Then**

\[ \text{currentLoad} = \sum_{ca \in CA} L(ca); /* load of the activity */ \]

**If** \( \text{currentLoad} < lc \) **Then**

\[ \text{availSpace} = lc - \text{currentLoad}; /* Check for space */ \]

\[ \text{toBePlaced} = \min\{\text{availSpace}, \text{LoadToRescheduled}\}; \]

\[ \text{LoadToReschedule} = \text{LoadToReschedule} - \text{toBePlaced}; \]

\[ \text{newLoad} = \text{currentLoad} + \text{toBePlaced}; \]

/* Create the new composite activity with the new load and insert it into CA */

\[ \text{ca}_{temp} = (\text{newLoad}, \ldots, a_{x-1}, b, a_{x+1} \ldots a_k)); \]

Insert \( ca_{temp} \) into \( CA_{temp} \);
In section 6.3, I describe an experimental study that shows this algorithm to minimize passenger trip delay in an airline rescheduling problem.
6.2. Stochastic Scheduling Problem and Algorithm

Though the new schedule has the estimated duration times for each activity, there is still randomness to the actual duration of an activity based on a random distribution. In this section, I will define a stochastic schedule and how it compares to the deterministic schedule. A simple algorithm for rescheduling the load and activities for a schedule that is no longer feasible will be presented along with a more efficient algorithm that minimizes the load normalized composite time, LNCT.

The stochastic schedule \( SS = (T, F, A, CA, HD) \) is the same as the deterministic schedule except that for each activity \( a \in A \) where \( a = (id, t, st, d, lc) \), \( d \) is based on a probabilistic distribution instead of being a constant.

Assume there exists a function \( SimDuration \) that runs a simulation of the duration of an activity that may or may not be dependent on previous activities.

Table 18 Simulation of Stochastic Schedule (SSS)

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) ( SS = (T, F, A, CA, HD) ) – the current stochastic schedule such that (( \forall f \in F )) (( \exists a = (id, t, st, d, lc) ) ( \in A ) such that:</td>
</tr>
<tr>
<td>a. ( t = f )</td>
</tr>
<tr>
<td>b. ( st = Penalty ) (a penalty late time)</td>
</tr>
</tbody>
</table>

65
c. \( lc = \infty \)

(2) Time – the current time of the system.

(3) \( aStatus: A \rightarrow \{ \text{not\_started} \} \cup \{ \text{started} \} \times T \cup \{ \text{completed} \} \times T^2 \), i.e., for every activity \( a \in A \), \( aStatus(a) \) gives a status which is: “not\_started” or “started @ time \( start_t \)” or “completed @ time \( comp_t \), started @ time \( start_t \)”, \( start_t, comp_t \in T \) such that all started activities satisfy hard and soft constraints of \( SS \).

**Output:** \( S = \langle T, F, A_{new}, C_{A_{new}}, HD_{new} \rangle \) – deterministic schedule realized from the stochastic schedule that satisfies the hard and soft dependencies

**Structures:**

1. \( ActivityQueue \) – sequence of activities in \( SS \) ordered by \( B(a) \)

2. \( NewActivityQueue \) – sequence of new activities in \( S \)

/* Ensure all hard dependencies are satisfied*/

For \( a = \langle id, t, st, d, lc \rangle \) In \( ActivityQueue \) Ordered By \( st \) Do

/* All activities are examined in order of increasing start time*/

If \( aStatus(a) = \text{"completed @} time \ comp_t, \text{started @} time \ start_t \text{"} \),

\( start_t, comp_t \in T \) Then

/* The activity has already started and completed, the start time and duration cannot be changed.*/

\[ d := \text{comp}_t - \text{start}_t; \]
\[ a_{new} = \langle id, t, \text{start}_t, d, lc \rangle; \]
\[ m(a) = a_{new}; \]
Insert $a_{new}$ Into $NewActivityQueue$;

**Else If** aStatus($a$) = "started @ time $start_t$", $start_t \in T$ **Then**

/* The activity has already started but not completed, the start time cannot be changed.*/

$$d := SimDuration();$$

$$a_{new} = (id, t, start_t, d, lc);$$

$$m(a) = a_{new};$$

Insert $a_{new}$ Into $NewActivityQueue$;

**Else If** aStatus($a$) = "not_started" **Then**

/* For all activities that have not been started, the activities are examined in order of increasing start time*/

$\text{StartTime}$

$$= \max \left( \{ B(a) \} \cup \left\{ st + d + \Delta \left| (\exists a_p <_a a) \in HD \land (a_p = (id, t, st, d, lc) \in NewActivityQueue) \right. \right\} \right);$$

$$d = SimDuration();$$

$$a_{new} = (id, t, StartTime, d, lc);$$

$$m(a) = a_{new};$$

Insert $a_{new}$ Into $NewActivityQueue$;

**End If:**
Using the simulation in Table 18, a deterministic schedule is obtained from the stochastic schedule. With the deterministic schedule, the load normalized composite time, denoted $LNCT(S)$, can be computed. Since $LNCT(SS)$ is computed on one possible outcome of the stochastic schedule which is a deterministic schedule, it is defined as the expected $LNCT(SS)$:

$$\text{Exp}_L \text{NCT}(SS) = \frac{WCT(SS)}{\sum_{ca \in CA} L(ca)}$$

**Definition 9** Let $SS_1 = \langle T, F, A_1, CA_1, HD_1 \rangle$ be a stochastic schedule. We say that $SS_2 = \langle T, F, A_2, CA_2, HD_2 \rangle$ is a delay of schedule $SS_1$ if there exists a mapping $m: A_1 \to A_2$ such that:

1. $m$ is a one-to-one and onto,
2. $HD_2 = \{ m(i) \preceq \Delta m(j) | i \preceq \Delta j \in HD_1 \}$ (i.e., $m$ preserves hard dependencies), and
3. $\forall a \in A_1, B(m(a)) \geq B(a)$ (i.e., $SS_2$ can only “delay” activities).

We say that $SS_2$ is task-load equivalent delay of $SS_1$ if:
4. \( SS_2 \) is a delay of \( SS_1 \).

5. \( CT(CA_1) = CT(CA_2) \) (i.e., \( SS_1 \) and \( SS_2 \) have the same set of composite tasks), and

\[ \forall ct \in CT, L(ct, CA_1) = L(ct, CA_2). \]

The stochastic scheduling problem can then be formally described in Table 19.

**Table 19 Stochastic Scheduling Problem**

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) ( SS = (T, F, A, CA, HD) ) – stochastic schedule</td>
</tr>
<tr>
<td>(2) Time – the current time of the system.</td>
</tr>
</tbody>
</table>
| (3) \( aStatus: A \rightarrow \{\text{not\_started}\} \cup \{\text{started}\} \times T \cup \{\text{completed}\} \times T^2 \) \), i.e., for every activity \( a \in A \), \( aStatus(a) \) gives a status which is: “not\_started” or “started \@ time \( \text{start}_t \)” or “completed \@ time \( \text{comp}_t \), started \@ time \( \text{start}_t \)”, \( \text{start}_t, \text{comp}_t \in T \).

<table>
<thead>
<tr>
<th>Output: ( S = (T, F, \hat{A}, \hat{CA}, \hat{HD}) ) – stochastic schedule such that:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( S ) is a task-load equivalent delay of ( SS ) such that:</td>
</tr>
<tr>
<td>a. For every ( a \in A ) if ( aStatus(a) = \text{“completed @ time comp}_t ), started @ time ( \text{start}_t )” then</td>
</tr>
<tr>
<td>b. ( \text{comp}_t, \text{start}_t \in T )</td>
</tr>
<tr>
<td>ii. ( B(m(a)) = \text{start}_t )</td>
</tr>
<tr>
<td>iii. ( D(m(a)) = \text{comp}_t - \text{start}_t )</td>
</tr>
</tbody>
</table>
b. For every \( a \in A \) if \( a\text{Status}(a) = \text{“started @ time start}_t\text{”} \) then

i. \( \text{start}_t \in T \)

ii. \( B(m(a)) = \text{start}_t \)

iii. \( D(m(a)) = \text{SimDuration} \)

c. For every \( a \in A \), if \( a\text{Status}(a) = \text{“not_started”} \), then \( B(m(a)) \geq \text{Time} \).

2. \( \forall S \in S_{\text{Total}} \) where \( S_{\text{Total}} \) is the set of all new possible schedules,

if \( S \) satisfies (1), then \( \text{Exp}_\text{LNCT}(SS) \geq \text{Exp}_\text{LNCT}(S) \) (i.e., \( S \) is a schedule with minimal \( \text{Exp}_\text{LNCT} \)among schedules that satisfy (1)).

Now I describe algorithms to obtain the best stochastic schedule. First I describe a sub-problem: if I am given a set of candidate stochastic schedules, I can use Monte Carlo to find the best estimated stochastic schedule. Then, in Table 21, I describe creating multiple stochastic schedules candidates from an existing stochastic schedule given the schedule’s current status.

Using Monte Carlo, the schedules could each be run a number of times to determine the expected \( \text{Exp}_\text{LNCT}(S) \) as shown in Table 20.
Table 20 Monte Carlo Stochastic Scheduling Algorithm

**Input:** Same as the input for the stochastic scheduling problem in Table 19

- `numIterations` – the number of iterations for the algorithm to be run
- `arrayS` – an array of stochastic schedules

**Output:** $S_{New}$ – the schedule with the minimum LNCT

**Structures:** $expLNCT[\text{numIterations}]$ – The expected LNCTS of the schedule; is initialized to 0

```plaintext
For i = 1 To numIterations Do

    j := 0;

    For Every S In arrayS Do

        $S_{temp} := SSS(S)$;

        $Exp\_LNCTS[j] := exp\_LNCTS + LNCTS(S_{temp})$;

        j := j + 1;

    End Loop;

End Loop;

minLNCTS := $\infty$;

For j := 0 To arrayS.Count Do

    $expLNCTS[j] := expLNCTS[j] / numIterations$;

    If $expLNCTS[j] < minLNCTS$ Then

        minLNCTS := $expLNCTS[j]$;

        $S_{New} := arrayS[j]$;

    End If;

End Loop;

Return $S_{New}$;
```
Assuming that the actual simulation for determining the duration of a task is complex and time consuming, it is preferable to not iterate over every possible schedule to find the best one. All-puRpose Algorithm for Generating a scHedule, ARAGH, takes a stochastic schedule as input, generates deterministic candidates, replaces with stochastic candidates, and uses known algorithms for obtaining top-k such as OCBA to determine the best schedule.

Table 21 ARAGH: All-puRpose Algorithm for Generating a scHedule

<table>
<thead>
<tr>
<th>Input: Same as the input for the stochastic scheduling problem in Table 19.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $SS = \langle T, F, A, CA, HD \rangle$ – stochastic schedule</td>
</tr>
<tr>
<td>2. Time – the current time of the system.</td>
</tr>
<tr>
<td>3. $aStatus: A \rightarrow {not_started} \cup {started} \times T \cup {completed} \times T^2$, i.e., for every activity $a \in A$, $aStatus(a)$ gives a status which is: “not_started” or “started @ time $start_t$” or “completed @ time $comp_t, started @ time start_t$”, $start_t, comp_t \in T$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output: Same as the output for the stochastic scheduling problem in Table 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \langle T, F, \hat{A}, \hat{CA}, \hat{HD} \rangle$ – deterministic schedule realized from the stochastic schedule that satisfies the hard and soft dependencies</td>
</tr>
</tbody>
</table>
**Heuristic Parameters:**

- $C$ – Total computation budget,
- $DB$ – A budget delta, i.e., constant part of the budget to be used in a single iteration,
- $P_{D}$ – Desired statistical confidence (e.g., 95%) that the top-k selection is correct,
- $K$ – Desired number of tuples for selection,
- $t_{0}$ – Initialization budget for each tuple
- $numIterations$ – the number of iterations for the algorithm to be run
- $m$ – number of new schedules to generate

**Structures:**

- $B$ – Portion of the budget consumed so far
- $P_{B}$ – Statistical confidence that the current top-k selection is correct
- $expLNCT[m]$ – The expected LNCTS of the schedule; is initialized to 0
- $NewSchedules[m]$ – array of new schedules
- $maxD[A.count]$ – array of maximum durations for each activity that doesn’t violate the hard or soft constraints
- $S_{New}$ – the schedule with the minimum $LNCT$

**Phase I:**

**Step 1:** //*Generate new schedules*/

\[
\text{For each } a = \langle id, t, st, d, lc \rangle \text{ In } A \text{ Do}
\]

\[
maxD_{HD} = \min \left\{ st - \Delta \left| \begin{array}{c}
(\exists a_{p} \prec_{\Delta} a) \in HD \\
\land (a_{p} = (id, t, st, d, lc) \in A)
\end{array} \right. \right\}
\]
Loop:

/* Create deterministic schedule with the new delay */

For \( j = 0 \) to \( m \) Do

For each \( a = (id, t, st, d, lc) \) In \( A \) Do

\[ \delta = (\max D(a) - \text{Exp}(d))/(m - 1); \]

\( d_{\text{new}} = \text{Exp}(d) + (\delta \times j); \)

\( a_{\text{new}} = (id, t, st, d_{\text{new}}, lc); \)

Insert \( a_{\text{new}} \) into \( A_{\text{new}}; \)

Loop:

\( \text{NewSchedules}[j] = (T, F, A_{\text{new}}, CA, HD); \)

Loop:

Step 2: Optimize each of the new deterministic schedules

For \( j = 0 \) to \( m \) Do

\( \text{NewSchedules}[j] = \text{CreateSchedule}((\text{NewSchedules}[j], \text{Time}, \text{aStatus}); \)

Loop:

Step 3: Replace \( d \) for all activities \( a = (id, t, st, d, lc) \) with the corresponding stochastic simulation.
Phase II:

While \( P_B < P_D \) and \( B < C \) Loop

Step 4: Allocate the delta budget \( DB \) to each Schedule proportional to:

\[
\text{If } n = 0 \text{ Then } t_0
\]

\[
\text{Else } \frac{\text{variance}^2/n}{(b-\text{mean})^2}
\]

Step 5: Run simulations for each Schedule within allocated budget.

Step 6: Re-compute \( P_B \) and \( B \).

End Loop

Step 7: Return the top Schedule and \( P_B \).

Table 22 CreateSchedule

Input:

1. \( S_{curr} = (T, F, A_{curr}, CA_{curr}, HD_{curr}) \) – the current schedule such that \( (\forall f \in F)(\exists \ a = (id, t, st, d, lc)) \in A_{curr} \) such that:
   a. \( t = f \)
   b. \( st = \text{Penalty} \) (a penalty late time)
   c. \( lc = \infty \)

2. Time – the current time of the system.

3. \( aStatus: A_{curr} \rightarrow \{\text{not\_started}\} \cup (\{\text{started}\} \times T) \), i.e., for every activity \( a \in A_{curr}, aStatus(a) \) gives a status which is either “not\_started” or “started @ time t”.
such that all started activities satisfy hard and soft constraints of \( S_{\text{curr}} \).

Output:

\[ S_{\text{new}} = \langle T, F, A_{\text{new}}, CA_{\text{new}}, HD_{\text{new}} \rangle \] – a new feasible schedule

Structures:

1. \( \text{TempSchedule} = \langle T, F, A_{\text{temp}}, CA_{\text{temp}}, HD_{\text{temp}} \rangle \) – The current best schedule. We denote \( \text{TempSchedule}.CA_{\text{temp}} = S_{\text{curr}}.CA_{\text{curr}} \)

2. \( \text{NewActivityQueue} \) – sequence of new activities in \( S_{\text{new}} \)

3. \( \text{TempActivityQueue} \)

4. \( \text{LoadRescheduleSchedule} \) – Schedule where the load of the current activity is reschedule

5. \( \text{TempSchedule1} \) – temporary for saving schedule where missed loads are redistributed

6. \( \text{TimeRescheduleSchedule} \) – temporary schedule where the activity is delayed till the previous activity completes.

7. \( \text{TempSchedule2} \) – temporary for saving schedule where the activity is delayed till the previous activity completed instead of redistributing the tardy loads.

/* Ensure hard dependencies are met*/

\[ \text{NewActivityQueue} = \text{HardDependencyReschedule}(S_{\text{curr}}.\text{Time}, a\text{Status}); \]

\[ \text{TempSchedule} = \langle T, F, A, CA, HS \rangle \] where \( A = \text{NewActivityQueue}, CA = CA_{\text{curr}}, \) and \( HD = HD_{\text{curr}}; \) /* may not be a feasible schedule*/
/* Loop through all activities while there is a soft dependency violation. Check to see if it is better to delay the activity till the previous activity arrives or to reschedule the load of the activity to the next available task */

While (\(\exists ca = (L, (a_1, a_2, \ldots, a_k)) \in TempSchedule.CA(\exists i, 2 \leq i \leq k)\) such that \(B(a_i) < B(a_{i-1}) + D(a_{i-1}) + \delta\) Do

Let \(ca\) be any composite activity that satisfies the while condition.

Remove \(ca\) From \(CA\); /* Remove the ca from CA*/

Let \(x = i\) be the minimal \(2 \leq i \leq k\), such that \(B(a_x) < B(a_{x-1}) + D(a_{x-1}) + \delta\)

/*Reschedule the load for this activity.*/

LoadRescheduleSchedule = RescheduleLoad (TempSchedule, ca, x);

TempSchedule1 = SoftDependencyReschedule (LoadRescheduleSchedule);

/*Delay next activity till the previous activity completes. Then enforce the hard and soft constraints*/

TimeRescheduleSchedule = RescheduleActivity (TempSchedule, ca, x)

TimeRescheduleSchedule = HardDependencyReschedule

(TimeRescheduleSchedule, Time, aStatus);

TempSchedule2 = SoftDependencyReschedule (TimeRescheduleSchedule);

/* Check to see which new schedule has the best delay*/

If \(LNCT(TempSchedule1) < LNCT(TempSchedule2)\) Then

TempSchedule = LoadRescheduleSchedule ;
Else

\[ TempSchedule = TimeRescheduleSchedule; \]

End If;

End While;
6.3. Specialized Transportation Rescheduling Algorithm for Optimizing Heuristics

One type of scheduling problem is passenger airline rescheduling. For the problem of minimizing passenger trip delay, we have relation tables for flights (Table 23), flight schedules (Table 24), and itineraries (Table 25) that are described below:

Table 23 Flight Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>flight_id</td>
<td>Sequence. Primary key for the table.</td>
</tr>
<tr>
<td>plane_id</td>
<td>The number of plane 1. This is equivalent to the VIN of a car.</td>
</tr>
<tr>
<td>flight_num</td>
<td>The flight number for the flight. When combined with the date and airline, the flight number identifies a particular flight.</td>
</tr>
<tr>
<td>origin</td>
<td>The four digit international civil aviation organization (ICAO) code of the airport where the flight is originating.</td>
</tr>
<tr>
<td>destination</td>
<td>The four digit ICAO code of the airport that is the final destination of the flight.</td>
</tr>
<tr>
<td>num_seats</td>
<td>Total number of seats available on the plane.</td>
</tr>
</tbody>
</table>
### Table 24 Flight Schedule Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>flight_schedule_id</td>
<td>Sequence. Primary key for the table.</td>
</tr>
<tr>
<td>flight_id</td>
<td>Foreign key to the flight table.</td>
</tr>
<tr>
<td>estimated_dept_time</td>
<td>The scheduled departure time of the flight from its origin.</td>
</tr>
<tr>
<td>estimated_arrival_time</td>
<td>The scheduled arrival time of the flight at its destination.</td>
</tr>
<tr>
<td>actual_dept_time</td>
<td>The actual departure time of the flight from its origin.</td>
</tr>
<tr>
<td>actual_arrival_time</td>
<td>The actual arrival time of the flight at its destination.</td>
</tr>
<tr>
<td>scheduled_seats</td>
<td>The number of seats with scheduled passengers.</td>
</tr>
</tbody>
</table>

### Table 25 Itineraries Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>itinerary_id</td>
<td>Sequence. Primary key for the table.</td>
</tr>
<tr>
<td>flight_schedule_id1</td>
<td>Foreign key to the flight schedule table</td>
</tr>
<tr>
<td>flight_schedule_id2</td>
<td>Foreign key to the flight schedule table. If the passenger only has one flight, this column will be null.</td>
</tr>
<tr>
<td>Pax</td>
<td>Number of passengers with this itinerary</td>
</tr>
</tbody>
</table>

In [26], Passenger Trip Delay is defined as the difference between the actual time of arrival of the passenger and the ticketed time of arrival allowing for fifteen minutes delay:
Equation 2: PTD

*Passenger Trip Delay (PTD)*

\[ \text{PTD} = \text{Actual Time of Arrival} - \text{Ticketed Time of Arrival} - 15 \text{ minutes} \]

As depicted in Figure 3, Passenger Trip Delay can occur as a result of one or more of the following scenarios:

1. In scenario 1 of Figure 3, there is not PTD because the flight arrived no later than 15 minutes after its expected arrival time.

2. In scenario 2 of Figure 3, the flight is delayed more than 15 minutes. In this scenario, the PTD would be the number of passengers multiplied by the amount of delay – 15 minutes. As an example, if the flight had 32 passengers and was supposed to arrive at 12:30, but arrived at 13:30, the PTD = 32*(13:30-12:30-15 minutes) = 32*(45 minutes) = 1,440 minutes.

3. In scenario 3 of Figure 3, passengers arrive late because the flight had to be diverted to another airport before continuing to the final destination. The PTD computation would not be different in this scenario that the computation used in scenario 2.

4. In scenario 4 of Figure 3, passengers arrive late because their original flight was either cancelled or overbooked and the passengers had to be rebooked on another
flight. For this scenario, the PTD is computed using the expected arrival of their original flight. For example, 12 passengers were booked on a flight that was supposed to arrive at 12:30, but these passengers had to be rebooked on another flight because it had been overbooked. The new flight arrives at 14:30. The expected arrival of the new flight does not need to be taken into account. The PTD = 12 * (14:30-12:30 – 15 minutes) = 12 * (105 minutes) = 1,260 minutes.

These scenarios can occur on one or more legs of a passenger’s itinerary, creating a potential for severe delays.
Figure 3 Passenger Trip Delay (solid) is determined by one of the scenarios described in Space-Time diagrams. The dashed line corresponds to the scheduled arrival time of the flight. [26]
The trip delays experienced by passengers on late flights and on diverted flights are proportional to the magnitude of the delay of these flights. The trip delays experienced by passengers that have to be re-booked due to cancelled flights, denied boarding, or missed connections are a function of the frequency and load factors (i.e. seats available) on other flights to the ticketed destination. As the frequency of the flights decreases and/or the load factor of candidate re-booked flights increases, the “reservoir” of seat capacity is reduced and the trip delay experienced by these passengers increases non-linearly.

The data used for developing the algorithm and conducting the experiments consists of a hub and spoke network in Dallas, Texas. This table of itineraries has the origin and final destination of a passenger along with any possible transfer points (hubs). The table is described in Table 25 with two sample itineraries in Table 26.

In Table 26, the first itinerary is a direct flight from LAX to DFW with a total of eight passengers having this itinerary. Looking at the scheduled arrival and actual arrival for this flight, the flight is only twelve minutes late. Since twelve minutes is under the allowed fifteen minute buffer, the flight is not considered late and there is no PTD.

In Table 26, the second itinerary is a flight from LAX to ATL through DFW with a total of five passengers having this itinerary. This itinerary has the same first leg as itinerary 1. When the itinerary has more than one leg, a check must be made to ensure that the first leg of the itinerary arrives before the second leg of the itinerary must depart. Thirty minutes is assumed for a passenger to change planes. If the first flight does not arrive at least 30 minutes before the second flight departs, the passengers cannot make
the second flight and must be booked on the next available flight. In this itinerary, the first flight does arrive late, but still with enough time for the passengers to make the second flight.

In Table 26, the third itinerary is a flight from LAX to AUD through DFW with a total of three passengers having this itinerary. The first flight is late, causing these three passengers to miss the second flight. They will have to be rescheduled on the next flight that has available seats.

Table 26 Three sample itineraries, including flight and flight schedule information, for passengers for a single day of travel from, to, or through Dallas, Texas.

<table>
<thead>
<tr>
<th></th>
<th>Itinerary 1</th>
<th>Itinerary 2</th>
<th>Itinerary 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>LAX</td>
<td>LAX</td>
<td>LAX</td>
</tr>
<tr>
<td>destination</td>
<td>DFW</td>
<td>ATL</td>
<td>AUD</td>
</tr>
<tr>
<td>hub</td>
<td>DFW</td>
<td>DFW</td>
<td>DFW</td>
</tr>
<tr>
<td>plane_1_id</td>
<td>1234</td>
<td>1234</td>
<td>1236</td>
</tr>
<tr>
<td>plane_2_id</td>
<td>1235</td>
<td>1234</td>
<td>1234</td>
</tr>
<tr>
<td>fl_1_num</td>
<td>2401</td>
<td>2401</td>
<td>2402</td>
</tr>
<tr>
<td>fl_1_sch_dep</td>
<td>00:25:00</td>
<td>00:25:00</td>
<td>00:55:00</td>
</tr>
<tr>
<td>fl_1_sch_arr</td>
<td>05:10:00</td>
<td>05:10:00</td>
<td>05:40:00</td>
</tr>
<tr>
<td>fl_1_act_dep</td>
<td>00:31:00</td>
<td>00:31:00</td>
<td>01:22:00</td>
</tr>
<tr>
<td>fl_1_act_arr</td>
<td>05:22:00</td>
<td>05:22:00</td>
<td>06:31:00</td>
</tr>
<tr>
<td>fl_2_num</td>
<td>342</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>fl_2_sch_dep</td>
<td>06:50:00</td>
<td>06:10:00</td>
<td></td>
</tr>
<tr>
<td>fl_2_sch_arr</td>
<td>09:50:00</td>
<td>07:05:00</td>
<td></td>
</tr>
<tr>
<td>fl_2_act_dep</td>
<td>06:59:00</td>
<td>06:11:00</td>
<td></td>
</tr>
<tr>
<td>fl_2_act_arr</td>
<td>10:10:00</td>
<td>07:10:00</td>
<td></td>
</tr>
<tr>
<td>fl_1_lf</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td>fl_1_avg_seat</td>
<td>165</td>
<td>165</td>
<td>165</td>
</tr>
<tr>
<td>fl_2_avg_seat</td>
<td>139</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>fl_2_lf</td>
<td>0.92</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>Pax</td>
<td>8</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
6.3.1. Rescheduling Passengers Utility Computation

Currently, when one of the flights of a passenger is late, the passenger may miss the connecting flight(s) and have to be rescheduled on subsequent flight(s). This domino effect can quickly add to the passenger’s frustration and increase his trip delay. Our method for assigning passengers who missed a flight is shown in the Reschedule_Pax algorithm, Table 27. When a passenger misses a flight, the passenger is assigned to the next flight going to the needed destination that is not full. In this instance, a new itinerary will be created and inserted into the itineraries table with a new itinerary id. The original arrival information will be maintained for computing passenger trip delay. Though this does handle the problem of ensuring that the passenger arrives at his/her destination eventually, what if slightly delaying the connecting flight would enable the passenger to not miss it?

Table 27 Reschedule_Pax algorithm for rescheduling passengers who missed flights

| Input: | itineraries – the itineraries of passengers who have missed a flight |
|        | Flight_schedule – the schedule of flights |
| Output:| itineraries – new passenger itineraries |
| Structures: | new_flight – next flight to the passenger’s destination |
6.3.2. SMRGOL: Schedule Minimization for Generalized Operational Logistics

Though the method in Table 27 does handle the problem of ensuring that the passenger arrives at his/her destination eventually, what if slightly delaying the connecting flight would enable the passenger to not miss it?

SMRGOL looks at how delaying a flight might help minimize the passenger’s trip delay. It takes a flight schedule as input and generates a new schedule that minimizes passenger trip delay by either rescheduling the passenger or holding the connecting flight for a period of time to allow for the passenger to make the flight.
Figure 4: SMRGOL algorithm used to minimize passenger trip delay by either holding the next flight till a passenger arrives or rescheduling passengers from late connecting flights on the next available flight.
A flight is dependent on another flight if there is a dependency on the physical plane. For instance, flight 2 is dependent on flight 1 if flight 2 needs the physical plan of flight 1. Based on this dependency, flights are broken into tiers, with each tier depending on the previous tier, except for tier zero. Tier zero is not dependent on any other tier because the physical plan is already at the needed location for the flights. Tier zero assumes that all flights have arrived and there are no existing delays. Tier one is dependent only on tier zero. The flights of tier two are dependent on tier one and tier zero.

There are also two groups: soft constraint and hard constraint. The hard constraint group consists of flights where the flight is dependent on the physical plane of the previous flight. For instance, plane 1234 is needed for the first leg of itinerary 1 and 2 and the second leg of itinerary 3. So, if the first flight of itineraries 1 and 2 does not arrive by 05:50, then the second flight of itinerary 3 cannot depart on time. With the soft constraint group, the dependency is on the passengers from previous flights trying to make a connection. Violation of the soft constraint does not prevent the flight from departing on time as the hard constrain does. For instance, looking back at Table 26, flight 343 in itinerary 3 is dependent on flight 2402 because there are passengers on flight 2402 that are scheduled for flight 343. Flight 343 can still depart on time; it will just be missing the three passengers from flight 2402.
Table 28 SMRGOL Algorithm

**Input:** initineraries – Initial itineraries

  flight_schedule – Initial flight schedule

**Output:** best_itineraries - New flight itineraries minimizing PTD

  best_flight_schedule – New flight schedule minimizing PTD

  PTD – New passenger trip delay

**Structures:**

  min_PTD – minimum passenger trip delay

  best_schedule – the schedule with the minimum PTD

**For Every** tier = 1 to num_tiers **Do**

  flight_schedule := soft_reschedule_tier (flight_schedule, itineraries, tier);

  flight_schedule := hard_reschedule (flight_schedule);

  itineraries := reschedule_pax(itineraries, flight_schedule)

  PTD := compute_PTD (itineraries, flight_schedule);

  **If** PTD < min_PTD **Then**

    best_flight_schedule := flight_schedule;

    best_itineraries := itineraries;

    min_PTD := PTD;

  **End If**

**End For**
### Table 29 vwFlights

<table>
<thead>
<tr>
<th>Column</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre_flight</td>
<td>The flight id of a flight that affects the flight</td>
</tr>
<tr>
<td>pre_flight_arrival_time</td>
<td>The estimated arrival time of the preflight.</td>
</tr>
<tr>
<td>flight</td>
<td>The flight id</td>
</tr>
<tr>
<td>tier</td>
<td>The tier of the flight</td>
</tr>
<tr>
<td>dept_time</td>
<td>The estimated departure time of the flight</td>
</tr>
<tr>
<td>arrival_time</td>
<td>The estimated arrival time of the flight</td>
</tr>
<tr>
<td>duration</td>
<td>The amount of time expected for the flight to reach its destination</td>
</tr>
<tr>
<td>pax</td>
<td>The number of passengers on the flight that will be affected by the pre flight</td>
</tr>
<tr>
<td>constraint</td>
<td>If this preflight and flight combination is a hard constraint or a soft constraint.</td>
</tr>
<tr>
<td>next_departure_time</td>
<td>The next departure time of the plane used for the flight. This is the time that the physical plane must be at the next airport to prevent violating the hard constraint.</td>
</tr>
</tbody>
</table>

### Table 30 soft_reschedule_tier Algorithm

**Input:** flight_schedule – The flight schedule
initineraries – Initial itineraries

tier_num – the current tier to be considered

Output: flight_schedule – The updated flight schedule

Structures:

new_departure – max arrival time of pre-flights of a flight

For Each flight In vwFlights Where constraint = ‘soft’ and tier = tier_num Do

new_departure := flight.departure_time;

For Each pre_flight In flight Do

If pre_flight.arrival_time > new_departure Then

If flight.next_departure_time >

(flight.pre_flight_arrival_time + flight.duration + 30 minutes) Then

new_departure := flight.pre_flight_arrival_time;

End If

End If

End If

End For

flight.departure_time := new_departure + 30 minutes;

End For
Table 31 hard_reschedule Algorithm

| Input: flight_schedule – the flight schedule |
| Output: flight_schedule – new flight schedule |

Structures:
- new_departure – max departure time for a plane so that it arrives at the destination before its next flight

For Each plane In schedule Where constraint = ‘hard’ Do
- new_departure := flight.departure_time;

For Each plane In vwFlights Where constraint = ‘hard’ Order By tier Do
- If plane.arrival_time > plane.next_departure_time Then
  - new_departure := plane.arrival_time + plane.duration + 30 minutes;
- End If
End For
plane.departure_time = new_departure;
End For

6.3.3. BRYAGH: Basic Reduction Yare Approach for flights

BRYAGH improves upon SMRGOL by taking a more fine grain approach to rescheduling of flights. Though it still iterates over tiers, BRYAGH also iterates over individual flights with a soft constraint. This enables a single flight to be delayed instead of all flights in a tier. For instance, what is delaying the flights in a tier overall does not
improve PTD, but there is a single flight that has several passengers on a delayed flight. Assuming flight A that is a connecting flight for flight B is delayed and has thirty passengers for flight B. Depending on the delay of flight A, holding flight B for a few minutes would mean that the thirty passengers would incur a short delay overall, thereby lowering the overall PTD. BRYAGH would still delay flight B, but not the additional flights in the tier.

Table 32 BRYAGH Algorithm

<table>
<thead>
<tr>
<th><strong>Input:</strong></th>
<th>initneraries – Initial itineraries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>flight_schedule – Initial flight schedule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Output:</strong></th>
<th>best_itineraries - New flight itineraries minimizing PTD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>best_flight_schedule – New flight schedule minimizing PTD</td>
</tr>
<tr>
<td></td>
<td>PTD – New passenger trip delay</td>
</tr>
</tbody>
</table>

**Structures:**

<table>
<thead>
<tr>
<th>min_PTD – minimum passenger trip delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>best_schedule – the schedule with the minimum PTD</td>
</tr>
</tbody>
</table>

**Initialization:**

<table>
<thead>
<tr>
<th>flight_schedule := hard_reschedule (flight_schedule);</th>
</tr>
</thead>
<tbody>
<tr>
<td>best_flight_schedule:= flight_schedule;</td>
</tr>
<tr>
<td>best_itineraries := itineraries;</td>
</tr>
</tbody>
</table>
Soft_reschedule_flight iterates over a tier looking at flights with a soft constraint.
Starting with the flights that have the greatest number of passengers coming from connecting flights (pre-flights), the flight is delayed till the last pre-flight arrives and PTD is computed. Soft_reschedule_flight also uses a view called vwFlights, Table 29, that organizes the flights into tiers with the pre-flights constraints and if the constraint is a soft constraint or a hard constraint.
Hard_reschedule uses an additional view called vwPlanes, Table 34 that organizes the flights according to the physical plans. Hard_reschedule ensures that there is a physical plane for the next flight.

Table 33 soft_reschedule_flight

<table>
<thead>
<tr>
<th>Input: flight_schedule – The flight schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>initineraries – Initial itineraries</td>
</tr>
<tr>
<td>tier_num – the current tier to be considered</td>
</tr>
</tbody>
</table>

| Output: flight_schedule – The updated flight schedule |

<table>
<thead>
<tr>
<th>Structures: new_departure – max arrival time of pre-flights of a flight</th>
</tr>
</thead>
<tbody>
<tr>
<td>best_PTD – new passenger trip delay</td>
</tr>
<tr>
<td>best_schedule - The flight schedule with the minimum PTD</td>
</tr>
</tbody>
</table>

For Each flight In vwFlights Where constraint = ‘soft’ and tier = tier_num Order By pax Descending Do

new_departure := flight.dept_time;

For Each pre_flight In flight Do

If pre_flight.arrival_time > new_departure Then

If flight.next_departure_time > (flight.pre_flight_arrival_time + flight.duration + 30 minutes) Then

new_departure := flight.pre_flight_arrival_time;
flight.departure_time = new_departure + 30 minutes;

itineraries := reschedule_pax(itineraries, flight_schedule)

PTD := compute_PTD (itineraries, flight_schedule);

If PTD < new_PTD Then

    new_PTD := PTD;

    best_schedule := flight_schedule;

End If

End For

Table 34 vwPlanes

<table>
<thead>
<tr>
<th>Column</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain_id</td>
<td>The id of the physical plane.</td>
</tr>
<tr>
<td>pre_flight</td>
<td>The flight id of a flight that affects the flight</td>
</tr>
<tr>
<td>flight</td>
<td>The flight id</td>
</tr>
<tr>
<td>tier</td>
<td>The tier of the flight</td>
</tr>
<tr>
<td>dept_time</td>
<td>The estimated departure time of the flight</td>
</tr>
<tr>
<td>arrival_time</td>
<td>The estimated arrival time of the flight</td>
</tr>
<tr>
<td>duration</td>
<td>The amount of time expected for the flight to reach its</td>
</tr>
</tbody>
</table>
**Table 35 hard_reschedule**

**Input:** flight_schedule – the flight schedule

**Output:** flight_schedule – new flight schedule

**Structures:**

new_departure – max departure time for a plane so that it arrives at the destination before its next flight

new_departure := flight.departure_time;

**For Each** plane In vwPlanes **Order By** tier **Do**

If plane.arrival_time > plane.next_departure_time **Then**

new_departure := plane.arrival_time + plane.duration + 30 minutes;

**End If**

plane.departure_time := new_departure;

**End For**
7. PROTOTYPE SYSTEM

SimQL is a combination of functions written in Java that were imported into PostgreSQL using PL/Java and functions written in PL/pgSQL, the native language of PostgreSQL. A function can be written in Java and packaged into a jar created using a development environment such as Eclipse. Then, using PL/Java, this jar can be installed so that PostgreSQL can recognize it:

```sql
SELECT sqlj.install_jar('file:C:/simql/Funcs.jar', 'SimqlFunctions', true);
```

Then a function is created in PostgreSQL referencing a Java function in the jar file.

```sql
CREATE FUNCTION chi2cdf(integer, integer)
RETURNS double precision AS 'edu.gmu.simql.SimqlFunctions.chi2cdf'
LANGUAGE 'javau' VOLATILE;
```

Two tables, `sim_config` and `sim_metadata`, were also added to help generalize the functionality and enable a user to quickly add new stochastic functions and attributes or change configuration parameters such as simulation budget.

The configuration table, `sim_config`, allows the user to change parameters that will affect the simulations in SimQL. These parameters include:
- **Budget**: The total computing budget that can be used for simulation. All algorithms stop when this budget has been exhausted. This attribute is required.

- **Probability**: The desired statistical confidence that the results returned are correct. This is used as a stopping condition for algorithms such as OCBA. This attribute is required for using OCBA, GORBASH, and GORBASH+.

- **Sim_runs**: The number of iterations to run of algorithms such as OCBA. During each iteration, a portion of the budget is assigned for further simulation runs. This attribute is required for OCBA, GORBASH, and GORBASH+.

- **Init_samples**: The number of simulation runs to use for each tuple during the initialization phase of algorithms. This attribute is required for OCBA, GORBASH, and GORBASH+.

- **K**: the number of tuples desired for return from algorithms such as OCBA. This attribute is required for OCBA, GORBASH, and GORBASH+.

*Sim_metadata* is the table that the SimQL functions use to determine the primary key and stochastic attributes for a table or view. This table was created to make generalization easier. The attributes in this table are all required.

- **Table_name**: The name of the table or view with the stochastic attribute(s),

- **Table_pk**: The primary key of the table,
- **Attribute_name**: The name of the stochastic attribute in the table. If there are multiple stochastic attributes, then there will be one entry per stochastic attribute.

An additional table was added called `temp_data` that is used by the OCBA, GORBASH, and GORBASH+ algorithms. This is the table where the data generated by these algorithms is stored to be returned to the user. The attributes, in addition to the attributes described in Section 3.2 include:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>text</td>
<td>The name of the table containing the data to be used for the stochastic attribute</td>
</tr>
<tr>
<td>TablePK</td>
<td>integer</td>
<td>The primary key of the tuple from the table or view used to generate the top-k results</td>
</tr>
<tr>
<td>TableValue</td>
<td>double precision</td>
<td>The latest value of the stochastic attribute computed through simulation</td>
</tr>
<tr>
<td>Rank</td>
<td>integer</td>
<td>The ranking of the tuple according to the simulated mean of the stochastic attribute</td>
</tr>
<tr>
<td>Mean</td>
<td>double precision</td>
<td>The simulated mean of the stochastic attribute</td>
</tr>
<tr>
<td>Variance</td>
<td>double precision</td>
<td>The simulated variance of the stochastic attribute</td>
</tr>
<tr>
<td><strong>Column Name</strong></td>
<td><strong>Column Type</strong></td>
<td><strong>Purpose</strong></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NumSamples</td>
<td>integer</td>
<td>The number of samples (simulation runs) of the stochastic attribute</td>
</tr>
<tr>
<td>Allocation</td>
<td>double precision</td>
<td>The percentage of the budget to be allocated for the tuple for the next iteration</td>
</tr>
<tr>
<td>Budget</td>
<td>integer</td>
<td>The budget to be allocated for the tuple for the next iteration</td>
</tr>
<tr>
<td>Probability</td>
<td>double precision</td>
<td>The statistical confidence that the rank (k position) is correct</td>
</tr>
<tr>
<td>TotalProbability</td>
<td>double precision</td>
<td>The total statistical confidence of correct selection for the current simulation run</td>
</tr>
<tr>
<td>LastUpdated</td>
<td>date</td>
<td>To be used for optimization, this is the date that this data was computed. This can be used to determine if the data is “fresh” enough that the simulations do not have to be re-run for another query</td>
</tr>
<tr>
<td>RunType</td>
<td>varchar</td>
<td>The type of algorithm (TOPK, TOPG, or TOPG2) that was used to compute the simulation budget.</td>
</tr>
</tbody>
</table>

Currently, this “temporary” data is wiped for each new query when the LastUpdated date is more than an hour old. This was to make it easier for a user to make
further queries (i.e., add an additional attribute to the SELECT statement) without the need to run the simulations again. More attributes can be added to this table as needed for further functionality/algorithms.

For SMRGOL and BRYAGH, two tables were added for holding temporary data: missed_flight_iten (Table 37) and ok_flight_iten (Table 38). missed_flight_iten is used when rescheduling passengers. It contains all passengers who have missed their connecting flight and information regarding rescheduling of the passengers or failed attempts at rescheduling. ok_flight_iten is the table that stores all of the passengers who will make all flights. When a passenger in the table missed_flight_iten has been rescheduled, he will be moved to the ok_flight_iten. The rescheduling algorithm allows for partial rescheduling of passengers. In other words, if 5 passengers have missed a flight and 2 of them can be rescheduled on the next flight, those two passengers will be rescheduled and the other 3 will be rescheduled on a later flight.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>missed_pk</td>
<td>integer</td>
<td>The primary key for the table</td>
</tr>
<tr>
<td>iten_num</td>
<td>integer</td>
<td>A foreign key to the itineraries table described in Table 25.</td>
</tr>
<tr>
<td>misplaced_pax</td>
<td>integer</td>
<td>Number of passengers for this itinerary that did not make their flight because of their connecting flight arriving late.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Column Type</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hub</td>
<td>varchar</td>
<td>Transfer point for the passenger. This is the current location of a passenger who missed his flight.</td>
</tr>
<tr>
<td>destination</td>
<td>varchar</td>
<td>The final destination of the passenger.</td>
</tr>
<tr>
<td>original_arrival_time</td>
<td>date</td>
<td>The time that the passenger would have arrived at his destination if he had made his connecting flight.</td>
</tr>
<tr>
<td>new_arrival_time</td>
<td>date</td>
<td>The expected time that the passenger will arrive on the new flight that he has been scheduled on.</td>
</tr>
<tr>
<td>actual_arrival_time</td>
<td>date</td>
<td>The actual arrival time of the new flight that the passenger has been scheduled on.</td>
</tr>
<tr>
<td>attempted_reschedule</td>
<td>integer</td>
<td>Either 1 or 0 to signify if an attempt to reschedule the passengers has been made. 1 for yes and 0 for no. If an attempt has been made, another attempt is not made and the passengers are assumed to have the max delay of 900 minutes.</td>
</tr>
</tbody>
</table>
### Table 38 ok_flight_iten table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>it_num</td>
<td>integer</td>
<td>A foreign key to the itineraries table described in Table 25.</td>
</tr>
<tr>
<td>new_flight_num</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>hub</td>
<td>varchar</td>
<td>Transfer point for the passenger. This is the current location of a passenger who missed his flight.</td>
</tr>
<tr>
<td>destination</td>
<td>varchar</td>
<td>The final destination of the passenger.</td>
</tr>
<tr>
<td>original_arrival_time</td>
<td>date</td>
<td>The time that the passenger would have arrived at his destination if he had made his connecting flight.</td>
</tr>
<tr>
<td>new_arrival_time</td>
<td>date</td>
<td>The expected time that the passenger will arrive on the new flight that he has been scheduled on.</td>
</tr>
<tr>
<td>actual_arrival_time</td>
<td>date</td>
<td>The actual arrival time of the new flight that the passenger has been scheduled on.</td>
</tr>
<tr>
<td>pax</td>
<td>integer</td>
<td>Number of passengers with this itinerary</td>
</tr>
</tbody>
</table>

These tables can also be used to compute the PTD.
8. EXPERIMENTAL EVALUATION

Initial experiments have been run comparing Monte Carlo, OCBA, GORBASH, and GORBASH+ for top-k selection as well as SMRGOL for airline rescheduling.

8.1. Monte Carlo, OCBA, GORBASH and GORBASH+

The same environment and experimental setup was used for running experiments with the algorithms Monte Carlo, OCBA, GORBASH, and GORBASH+. Data was randomly generated.

Looking back at our problem with finding the best location for our wind farm on the Camford campus, we ran a series of experiments increasing the number of locations (tuples) to be simulated. With k=1 and a desired confidence of 95% (for the OCBA, GORBASH, and GORBASH+ algorithms), the size of the data table and available simulations (computational budget) are listed in the table below.

Table 39 Database table size and the corresponding computational budget for experiments.

<table>
<thead>
<tr>
<th># Tuples</th>
<th>Computational Budget (# of available simulations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10,000</td>
</tr>
<tr>
<td>5000</td>
<td>50,000</td>
</tr>
<tr>
<td>10,000</td>
<td>100,000</td>
</tr>
</tbody>
</table>
8.1.1. OCBA vs. Monte Carlo

On average, the percent improvement of OCBA over Monte Carlo was 42.8%. As the number of tuples increased, the amount of budget that was saved using OCBA also increased with the percent improvement reaching as high as 48.2% once the database was over one million tuples.
8.1.2. GORBASH vs. OCBA and Monte Carlo

For GORBASH, we sampled 0.1% of the tuples for the regression line and Q was computed with a desired statistical confidence of $\sqrt{0.95} = 0.9747$. 

Figure 5 PCS vs. Number of Simulations for 1 million tuples
Figure 6 PCS vs. simulations for 1 million tuples

GORBASH reached the desired probability of correct selection of 95% an average of 99.8% faster than OCBA and 99.9% faster than Monte Carlo.

8.1.3. GORBASH+ vs. OCBA and Monte Carlo

For GORBASH+, we sampled 0.1% of the tuples for the regression line and began with Q as computed with a desired statistical confidence of $\sqrt{0.95} = 0.9747$. 
Figure 7 PCS vs. simulations for 1 million tuples

GORBASH+ reached the desired probability of correct selection of 95% an average of 99.8% faster than OCBA and 99.9% faster than Monte Carlo.
8.2. SMRGOL and BRYAGH

For our experiments, we used a 131 airport hub-and-spoke network with Dallas, Texas as the hub and 130 spoke airports gleaned from airline transportation system statistics from 2007. Using data for a 24 hour period, we compared the PTD of SMRGOL to rescheduling passengers on the next available flight to their destination. If there was not a new connecting flight for a passenger during the twenty-four hour period, a 900 minute PTD was assumed. For rescheduling of passengers, the total PTD for the 24 hour period was 13358 hours. With SMRGOL, the PTD was reduced to 11836 hours giving an 19% improvement.
Table 40 Rescheduling passengers on next available flight versus SMRGOL and BRYAGH

<table>
<thead>
<tr>
<th></th>
<th>PTD (in hours)</th>
<th>% Improvement over Reschedule Pax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reschedule Pax</td>
<td>12737</td>
<td></td>
</tr>
<tr>
<td>SMRGOL</td>
<td>10315</td>
<td>19.0154668%</td>
</tr>
<tr>
<td>BRYAGH</td>
<td>10197</td>
<td>19.9419015%</td>
</tr>
</tbody>
</table>
9. CONCLUSIONS AND FUTURE WORK

SimQL allows the user to define computationally complex simulations that can be run for stochastic attributes while allowing the user to alter the parameters. SimQL leverages the power of the relational database to investigate multiple possible decisions in rapid succession without the need to create or update a complex model using MatLab or other traditional stochastic modeling techniques. SimQL can easily be applied to many areas where decision making is based on traditional simulation techniques, such as emergency responder scenarios and manufacturing. With the addition of simulation budget optimization algorithms such as OCBA, GORBASH and GORBASH+, the user is able to pose top-k queries on very large amounts of tuples efficiently.

GORBASH and GORASH+ reach the desired probability while managing to extensively save the amount of simulation budget needed to achieve the PCS. When dealing with an extremely large number of tuples, the cost savings of GORBASH and GORBASH+ will likely outweigh the slight increase in PCS by OCBA.

As a special case of top-k queries, the scheduling problem has been explored. Deterministic Algorithm for Load and machine reScheduling, DALEXS, Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios, TARDIS, are generalized scheduling algorithms that take an existing deterministic schedule that is no longer feasible along with the current time and system status and generates a new feasible
schedule. TARDIS takes the rescheduling one step further by optimizing the schedule.

All-puRpose Algorithm for Generating a scHedule, ARAGH, is a generalized scheduling algorithm that takes a stochastic schedule that is no longer feasible and generates a new schedule that optimizes a utility of the schedule.

As a first step in implementing this class of top-k query, the minimization of passenger trip delay for the airline industry has been investigated. SMRGOL and BRYAGH minimize PTD, but they are just a first step. In future iterations, extending the optimization with stochastic simulation to increase accuracy, model trade-offs between passenger delay and airline costs, implement a heuristic more precise than PTD for passenger cost, and extend the class of scheduling problems for which to develop SimQL [7], [9] algorithms based on the optimization of model approximation can be done.
APPENDIX


“A Simulation Query Language for Defining and Analyzing Uncertain Data” presented at The 15th IASTED International Conference on Software Engineering and Applications, Dallas, TX USA, 2011.


### INDEX

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-purPose Algorithm for Generating a schedule</td>
<td>18, 83, 132</td>
</tr>
<tr>
<td>ARAGH. <em>See</em> All-purPose Algorithm for Generating a schedule</td>
<td></td>
</tr>
<tr>
<td>Basic Reduction Yare Approach for flights</td>
<td>xi, 18, 108</td>
</tr>
<tr>
<td>BRYAGH, xi, 18, 36, 39, 108, 129, 130, <em>See</em> Basic Reduction Yare Approach for flights</td>
<td></td>
</tr>
<tr>
<td>Composite Activities, 59</td>
<td></td>
</tr>
<tr>
<td>Composite task, 61</td>
<td></td>
</tr>
<tr>
<td>DALEXS. <em>See</em> Deterministic Algorithm for Load and machine reScheduling</td>
<td></td>
</tr>
<tr>
<td>delay of schedule, 62, 79</td>
<td></td>
</tr>
<tr>
<td>Deterministic Algorithm for Load and machine reScheduling, vii, 65</td>
<td></td>
</tr>
<tr>
<td>feasible, 61</td>
<td></td>
</tr>
<tr>
<td>General Optimal Regression Budget Allocation Scheme, xi, 17</td>
<td></td>
</tr>
<tr>
<td>Hard Dependency, 59</td>
<td></td>
</tr>
<tr>
<td>LNCT. <em>See</em> load normalized composite time, <em>See</em> load normalized composite time</td>
<td></td>
</tr>
<tr>
<td>load normalized composite time, 62, 63, 76, 79</td>
<td></td>
</tr>
<tr>
<td>Monte Carlo Stochastic Scheduling Algorithm, 82</td>
<td></td>
</tr>
</tbody>
</table>

117
Allocation, See 2.5. Optimal Computing Budget Allocation
Optimal Computing Budget Allocation, xi, 15, 16, 23
PCS, xi, 49, 50, 53, 125, 127, 128
probability of correct selection, 15, 17, 19, 23, 45, 50, 127, 129, See PCS
PTD, xi, 19, 93, 94, 96, 103, 108, 110, 129, 130
Schedule Minimization for Generalized Operational Logistics
SMRGOL, xi, 18, 100
Simulation of Stochastic Schedule, xi, 76
SMRGOL. See Schedule Minimization for Generalized Operational Logistics
soft dependency, 60
SSS. See Simulation of Stochastic Schedule
Stochastic Scheduling Problem, 76, 80
TARDIS. See Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios
task-load equivalent delay, 62, 63, 80
Tuned Algorithm for Rescheduling Dependency Interrelationship Scenarios
WCT. See weighted composite time
weighted composite time, 62
REFERENCES


[34] I. F. Ilyas, G. Beskales, and M. A. Soliman, “A Survey of Top-k Query Processing Techniques in Relational Database Systems.”
CURRICULUM VITAE

Susan Farley graduated from Augusta Preparatory Day School, Augusta, Georgia, in 1993. She received her Bachelor of Science from University of Georgia in 1998 and her Master of Engineering in Modeling and Simulation from Old Dominion University in 2001. She was employed as a scientist for the Naval Surface Warfare Center in Dahlgren for 6 years and currently works as a scientist for Booz Allen Hamilton.