# Non-dominated Sorting Genetic Algorithm for Multiple Objectives Query Optimization in Spark SQL

Trung-Dung Le(✉)

Thuyloi University, Hanoi 11515, Vietnam
dung_lt@tlu.edu.vn
https://tlu.edu.vn/

**Abstract.** Cloud computing allows building applications to store and analyze heterogeneous data sources under a pay-as-you-go paradigm. This elasticity leads to significant challenges for Multi-Objective Query Processing (MOQP), requiring Query Execution Plans (QEPs) that balance diverse user priorities, such as execution time, monetary cost, etc. Spark SQL and Spark SQL's Catalyst Optimizer provide a solution using Cost-based optimization, which can process data from multiple heterogeneous resources. In this solution, they applied the dynamic programming algorithm to pick the best m-way join order. However, when a user would like to expand the cost function to the execution time or monetary cost in a multiple-cloud environment, the optimizer could ignore good plans. To address this problem, we propose a MOQP extension for Spark SQL, leveraging a Non-dominated Sorting Genetic Algorithm based on Grid partitioning (NSGA-G). We validate our proposal with preliminary experimental results.

**Keywords:** Non-dominated Sorting Genetic Algorithm · SparkSQL · Multiple Objectives Query Optimization

## 1 Introduction

Minimizing execution time or the size of transferring data is the objective in heterogeneous database systems [9,13,19]. However, cloud computing providing elasticity with a pay-as-you-go model raises an important issue in terms of Multi-Objective Query Processing (MOQP) to find a Query Execution Plan according to users' preferences in terms of time, money, quality, etc. For instance, healthcare data of a given patient may be owned by different hospitals that may use various providers. In the night, the queries could be focused on the monetary aspect more than the execution time. But, during the medical consultation meeting in the daytime, the queries should prioritize the execution time.

Spark SQL[1] and Spark SQL's Catalyst Optimizer [1] provide a solution using Cost-based optimization, which can process data from multiple heterogeneous

[1] https://spark.apache.org/sql/.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15648, pp. 211–224, 2025.
https://doi.org/10.1007/978-3-031-97000-9_13

resources. In this solution, they applied the dynamic programming algorithm in [20] to pick the best m-way join order. In this way, the search space is reduced significantly. But, the cost value is combined with the selectivity and size of tables as a single cost metric, not multiple objectives. Hence, when a user would like to expand the cost function to the execution time or monetary cost in a multiple-cloud environment, the optimizer could ignore good plans.

However, when users consider a trade-off between response time, monetary fees, and other objectives, they should use the Multiple Objectives Query Optimization algorithm to find a good Query Execution Plan. It raises the importance of finding the Pareto-optimal front [11] of QEPs. Besides, generating the Pareto-optimal front can be computationally expensive and is often infeasible [26]. Since generating QEPs may be infeasible due to high complexity, one of the objectives of MOQP in Spark SQL is to find an approximate optimal solution.

In this context, a challenging problem is how to find an approximate optimal solution in MOQP using an efficient Multi-Objective Optimization algorithm in Spark SQL [1].

The solutions in [13,19] provide various tools to manage heterogeneous data in multiple cloud environments. [13] is a functional SQL-like language, capable of querying multiple heterogeneous data stores, but it optimizes queries on minimizing data transfers, not execution time, monetary or other cost metrics. Besides, [19] showed a method to process a query that integrates data in PostgreSQL and MongoDB. They used the syntax and semantics of SQL++, which is a unifying semi-structured data model and query language that is designed to encompass the data model and query language capabilities of NoSQL, New SQL, etc. In their paper, they did not discuss cost optimization and polystore design. Intelligent Resource Scheduler (IReS) [9] resolved MultiObjective Query Processing in heterogeneous systems by a specified policy that combined multiple objectives into a scalar value and used Waikato Environment for Knowledge Analysis (Weka) [10]. They generated all possible query plans and chose the best plan by combining multiple objectives into a scalar value after having all the plans, not using any algorithms to reduce the search space during the building plan process. Hence, their solution has a very large search space.

Furthermore, [17] focused on the way to build cost models on Spark SQL, and Spark SQL's Catalyst Optimizer provides a solution using Cost-based optimization, which can process data from multiple heterogeneous resources. In this approach, they did not focus on the large search space of Query Execution Plans. Recent advancements like UDAO [21] optimize Spark clusters for performance and cost. Nonetheless, these efforts lack a unified MOQP solution for large-scale variable environments.

To the best of our knowledge, existing solutions do not address the problem of MOQP in the large-scale variable environment and optimize Multi-Objective Query Processing efficiently.

On the other hand, Evolutionary Multiobjective Optimization (EMO) approaches used Pareto dominance techniques [11] to solve the high complexity of the Multi-Objective Optimization Problem. Among EMO approaches, Non-

dominated Sorting Genetic Algorithms (NSGAs) [6,7] have lower computational complexity than other EMO approaches [7].

However, [12] stated that the original NSGA-II [7] does not work well on many-objective test problems, such as DTLZ [8], and NSGA-III [6] does not outperform classical EMO algorithms such as NSGA-II in some many-objective test problems. In the previous works [14–16], NSGA-G was proposed to solve this problem in DICOM data management [18] and IReS platform [9]. The algorithm has not been applied to Spark SQL. [4] used NSGA-II to propose a Dynamic Multi-Objective Optimization With jMetal and Spark, while the algorithm may not work well on many-objective test problems [12]. [2] used the multi-objective optimization algorithm based on decomposition (MOEA/D) [25] to focus on the configuration of Spark, and combine a multi-objective optimization approach and a performance prediction model. This algorithm does not completely outperform NSGA-II, NSGA-III, and NSGA-G [15] in many problems, which will be described in Sect. 3.2.

Hence, the other challenging problem is: how to improve the quality of the classical EMO algorithms in Multi-Objective Optimization Problems in Spark SQL.

This paper presented an application of Non-dominated Sorting Genetic Algorithm based on Grid partitioning (NSGA-G) to get a Pareto query execution plan set during the searching plans process. The algorithm is implemented in Spark SQL as an extended function for MOQP.

The remainder of this paper is organized as follows. In Sect. 2, we present the background information and the motivation. Non-dominated Sorting Genetic Algorithm based on Grid partitioning is shown in Sect. 3. NSGA-G algorithm is implemented in Spark SQL and shown in Sect. 4 as an extended function in Spark SQL and Catalyst library. Finally, Sect. 5 concludes this paper and hints at future researchers.

## 2 Preliminaries

First of all, the preliminaries of this paper show the background of techniques we use to propose an application of a Non-dominated Sorting Genetic Algorithm for Multiple Objectives Query Optimization in Spark SQL.

### 2.1 Multiple Linear Regression

A cost function of Multiple Linear Regression (MLR) model in [22] is defined: $c = \beta_0 + \beta_1 x_1 + ... + \beta_L x_L + \epsilon$, similar with cost model above, where $\beta_l$ is the unknown coefficient, $x_l (l = 1, ..., L)$ are the variables, e.g., size of data, computer configuration; $c$ is the cost function value, and $\epsilon$ is the random error with a corresponding distribution which is approximated by a normal distribution $\mathcal{N}(0, \sigma)$. The **fitted equation** is defined:

$$\hat{c} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_L x_L \tag{1}$$

### 2.2 Non-dominated Sorting Genetic Algorithm

Non-dominated Sorting Genetic Algorithm (NSGA) is one of the first Evolutionary Algorithms (EAs) for solving MOQP. Generally, the multi-objective optimization problem is:

$$minimize F(x) = (f_1(x), f_2(x)..., f_m(x)) \tag{2}$$

where $x = (x_1, ..., x_n)^T \in \Omega \subseteq R_n$ is an n-dimensional vector of the decision variables, $\Omega$ is the decision (variable) space and $F$ is the objective vector function that contains $m$ real value functions.

Initially, a parent population $P_0$ is created randomly. Each solution is on a determined rank or nondomination level (any solution in level 1 is not dominated, any solution in level 2 is dominated by one or more solutions in level 1, and so on). In the beginning, the offspring $O_0$ is created with $N$ solutions by the binary tournament selection and mutation operators. Next, a population $C_0 = P_0 \cup O_0$ with the size of $2N$ should be divided into multiple ranks. After that, a parent population of next-generation $P_1$ is selected in $C_0$ from level 1 to level k so that the size of $P_1 = N$, and so on. At the $t - th$ step in the process of generation, population $C_t = P_t \cup O_t$ is generated. $C_t$ is sorted in level $F_1, F_2, ..., F_k$. Now the solutions belonging to the first non-dominated $F_1$ are the better solutions in $C_t$. If the size of $F_1$ is smaller than $N$, all member of $F_1$ is selected to $P_{t+1}$. Thus, solutions in $F_2$ are chosen next, and so on. This procedure is continued until no more levels can be fitted in $P_{t+1}$. The last level $F_l$ cannot be filled in $P_{t+1}$: $\mid F_l \mid > N - \sum_{i=1}^{l-1} \mid F_i \mid$. Figure 1 shows the procedure of NSGAs.

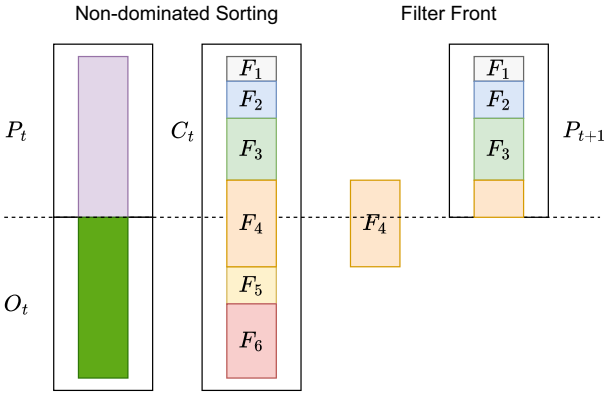


**Fig. 1.** NSGA-II procedure

NSGA-II and NSGA-III have two different approaches to select members in the last level $F_l$. While NSGA-II selected solutions by the crowding distance, NSGA-III used Reference Point approach. Both approaches require expensive computation, will be shown in this paper.

### 2.3 Motivation

Traditional optimizers like Catalyst's [1] reduce search spaces effectively but neglect multi-objective trade-offs. Alternatives like CloudMdsQL [13] and IReS [9] focus on single objectives or scalarized costs, lacking scalability for large QEPs. Recent work [21] highlights the need for dynamic, multi-objective solutions in Spark SQL. They still need to improve the computation of Multiple Objectives Optimization Problems in Spark SQL.

Non-dominated Sorting Genetic Algorithms make it possible to reduce the computational cost of selecting a good solution in the truncating process. Our algorithm experimented with DTLZ problems [8] and compared the quality between NSGA-II, NSGA-III in the Generational Distance (GD) [23], the Inverted Generational Distance (IGD) [3], and the Maximum Pareto Front Error (MPFE) [24]. Furthermore, NSGA-G is proposed to be applied in Spark SQL to address the problem of MOQP in a large-scale variable environment and optimize Multi-Objective Query Processing efficiently.

## 3 Non-dominated Sorting Genetic Algorithm Based on Grid Partitioning

Recently, many efficient Evolutionary Multi-objective Optimization (EMO) approaches have been developed based on Pareto dominance techniques [11]. Among them, NSGA-II and NSGA-III are well-known approaches. While NSGA-II used the crowding distance operator to select the solutions that do not perform well for many-objective problems [12], NSGA-III replaced that operator with a set of well-distributed reference points, which affects the execution time of the algorithm.

In this section, we introduce and experiment NSGA-G [14–16] with DTLZ problems [8] and compare the quality between NSGA-II, NSGA-III in the Generational Distance (GD), the Inverted Generational Distance (IGD), and the Maximum Pareto Front Error (MPFE).

### 3.1 NSGA-G

As shown in Fig. 1, at the $t$-th generation of NSGA algorithms, $P_t$ presents the parent population with $N$ size and $O_t$ is the offspring population with $N$ members created by $P_t$. $C_t = P_t \cup O_t$ is a group which need to be chosen $N$ members for $P_{t+1}$. In the first step, $C_t$ is sorted by a fast and elitist Multiobjective genetic algorithm to reduce computation time [6,7]. NSGA-II and NSGA-III are used to find the first front (dominating other solutions) and the rest are in the next fronts. After that, the algrithms find the second front, and so on. At the final step, the output of the algorithms are the fronts and their ranks.

After sorting, each non-domination level is selected one at a time to construct a new population $P_{t+1}$, starting from $F_1$, until $F_l$ so that the size of $\mid F_l \mid + \sum_{i=1}^{l-1} \mid F_i \mid$ is larger or equal to $N$. At $l$-th level, the algorithm

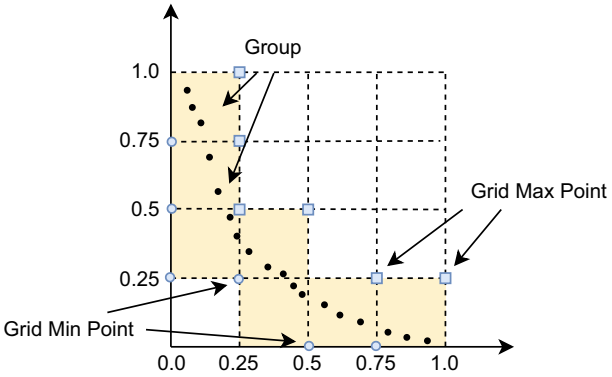


**Fig. 2.** Grid points and Groups

only accepted partially $F_l$. NSGA-II in [7] used crowding-distance-assignment algorithm while an Evolutionary Many-Objective Optimization Algorithm Using Reference-point Based Non-Dominated Sorting Approach, [6] concluded that the identification of neighbors became computationally expensive in a large dimension space and replaced the crowding-distance-assignment algorithm in NSGA-II by the Reference Points on a Hyper-Plane approach, NSGA-III. The work needed to associate each population member with the reference point in making a new parent member in the next iteration. Both algorithms require expensive computation by equations:

$$I[\,i\,]_{\,distance} = I[\,i\,]_{\,distance} + \frac{(I[\,i+1].m\, - I[\,i-1].m\,)}{(f_m^{max} - f_m^{min})}$$

or

$$f_i^n(x) = \frac{f_i^{'}(x)}{a_i - z_i^{min}}$$

Our approach finds the nearest smaller and bigger grid points of each solution. For instance, Fig. 2 shows an example of the two-objective problems. If we have the unit of the grid point is 0.25 and the solution with two objectives values of $[0.55, 0.66]$, the nearest smaller point is $[0.5, 0.5]$ and the nearest bigger point is $[0.75, 0.75]$. In many circumstances, there are many solutions that have the same nearest point, and they will be in a group. The solution has the smallest distance to the nearest smaller point in a group will be added to $P_{t+1}$. In this way, in any loop, we do not need to calculate the crowding-distance values or estimate the smallest distance from solutions to the reference points among all members in $F_4$ in Fig. 1. Algorithm 1 shows our strategy to select $N - \sum_{i=1}^{l-1} F_l$ members in $F_l$.

The functions updateIdealPoint and updateIdealMaxpoint in lines 1 and 2 will determine the new origin coordinates and the maximum objective values of all solutions. After that, they will be normalized to a range of 0–1. All solutions will be in different groups according to the coefficient of the grid. The key to

**Algorithm 1.** Filter front in NSGA-G

**Require:** $F_l, M = N - \sum_{i=1}^{l-1} F_i$
**Ensure:** $N - \sum_{i=1}^{l-1} F_i$ members in $F_l$
1: updateIdealPoint()
2: updateIdealMaxPoint()
3: translateByIdealPoint()
4: normalizeByMinMax()
5: createGroups
6: **while** $\mid F_l \mid > M$ **do**
7: selectRandomGroup()
8: removeMaxSolutionInGroup()
9: **end while**
10: **return** $F_l$

the NSGA-G algorithm is that we select the group randomly, like NSGA-III did, to keep the diversity characteristic and remove the solution among members of that group. It helps avoid comparing and calculating the maximum objectives in all solutions.

### 3.2 Experiment

For fair comparison and evaluation, we have used the same parameters, such as Crossover probability, Mutation probability, Distribution index for crossover, Distribution index for mutation for 1. NSGA-II, 2. NSGA-III, 3. MOEA/D, 4. NSGA-G[2], during their 50 run independent runs in solving two types of problems: DTLZ1 and DTLZ3 test problems with the objectives, $m$, are 4, 5, 6, 7, 8, 9, 10 in MOEA framework[3]. These three algorithms use the same population size $N = 100$ and the maximum evaluation $M = 1000$. To estimate the quality of three algorithms, we use the Generational Distance (GD) [23], the Inverted Generational Distance (IGD) [3], and the Maximum Pareto Front Error (MPFE) [24]. The GD value determines the distance from the evolved solution to the true Pareto Front [23]. IGD could measure both convergence and diversity in a sense. The most significant distance between the individuals in the Pareto Front and the evolved solution is measured and shown in MPFE. In three metrics, the better quality is shown by the lower value. Tables 1, 3 and 5 show that the quality of NSGA-G is better than other algorithms in these experiments. The advantages of NSGA-G are shown in most cases.

In the high computation comparing, Tables 2, 4, and 6 show the average of computation time of NSGA-II, NSGA-III, MOEA/D, and NSGA-G in seconds. The tables showed that NSGA-G had shorter computation times in most cases.

In conclusion, NSGA-G are presented not only the advantages in the diversity, convergence, the most distance between the individuals in Pareto Front and the evolved solution in three experiments, but also in the smaller computation time

[2] https://github.com/dungltr/MOEA.
[3] http://moeaframework.org/.

than the others in the large objective experiments, as shown in Tables 2, 4 and 6.

**Table 1.** Generational Distance

| | m | NSGA-II | MOEA/D | NSGA-III | NSGA-G |
|---|---|---|---|---|---|
| DTLZ1 | 4 | 3.427e+01 | 1.624e+00 | 1.093e+01 | 1.071e+00 |
| DTLZ3 | 4 | 5.081e+01 | 1.563e+01 | 2.718e+01 | 1.133e+01 |
| DTLZ1 | 5 | 2.143e+01 | 4.708e−01 | 3.130e+00 | 1.527e−01 |
| DTLZ3 | 5 | 1.133e+00 | 9.667e−02 | 3.109e−01 | 1.475e−02 |
| DTLZ1 | 6 | 9.309e+01 | 1.918e+00 | 7.097e+00 | 4.370e−01 |
| DTLZ3 | 6 | 1.277e+02 | 1.698e+01 | 2.385e+01 | 5.731e+00 |
| DTLZ1 | 7 | 3.160e+01 | 4.492e−01 | 1.858e+00 | 1.696e−01 |
| DTLZ3 | 7 | 8.994e−01 | 1.881e−02 | 9.341e−02 | 4.874e−03 |
| DTLZ1 | 8 | 1.134e+02 | 1.838e+00 | 1.000e+01 | 4.030e−01 |
| DTLZ3 | 8 | 1.639e+02 | 1.425e+01 | 2.977e+01 | 4.967e+00 |
| DTLZ1 | 9 | 1.656e+01 | 2.153e−01 | 1.701e+00 | 4.856e−02 |
| DTLZ3 | 9 | 7.132e−01 | 8.410e−03 | 1.194e−01 | 3.008e−03 |
| DTLZ1 | 10 | 2.665e+01 | 3.495e−01 | 2.724e+00 | 8.250e−02 |
| DTLZ3 | 10 | 6.030e−01 | 6.901e−03 | 1.041e−01 | 2.103e−03 |

**Table 2.** Average compute time in GD experiment (seconds)

| | m | NSGA-II | MOEA/D | NSGA-III | NSGA-G |
|---|---|---|---|---|---|
| DTLZ1 | 4 | 3.434e+01 | 1.721e+02 | 1.246e+02 | 3.790e+01 |
| DTLZ3 | 4 | 3.536e+01 | 1.678e+02 | 1.279e+02 | 4.036e+01 |
| DTLZ1 | 5 | 3.878e+01 | 1.748e+02 | 1.700e+02 | 4.012e+01 |
| DTLZ3 | 5 | 4.114e+01 | 1.741e+02 | 1.751e+02 | 4.336e+01 |
| DTLZ1 | 6 | 4.378e+01 | 1.868e+02 | 1.330e+02 | 4.290e+01 |
| DTLZ3 | 6 | 4.594e+01 | 1.889e+02 | 1.386e+02 | 4.684e+01 |
| DTLZ1 | 7 | 4.782e+01 | 2.047e+02 | 1.339e+02 | 4.588e+01 |
| DTLZ3 | 7 | 5.128e+01 | 2.052e+02 | 1.403e+02 | 4.958e+01 |
| DTLZ1 | 8 | 5.272e+01 | 2.200e+02 | 1.936e+02 | 4.796e+01 |
| DTLZ3 | 8 | 5.560e+01 | 2.223e+02 | 2.026e+02 | 5.184e+01 |
| DTLZ1 | 9 | 5.620e+01 | 2.338e+02 | 2.563e+02 | 5.028e+01 |
| DTLZ3 | 9 | 5.960e+01 | 2.345e+02 | 2.723e+02 | 5.450e+01 |
| DTLZ1 | 10 | 5.964e+01 | 2.401e+02 | 3.374e+02 | 5.246e+01 |
| DTLZ3 | 10 | 6.364e+01 | 2.449e+02 | 3.704e+02 | 5.676e+01 |

## 4 NSGA-G in Spark SQL

Spark SQL is one of the most technical components of Spark in processing SQL queries. The core of Spark SQL is Catalyst optimizer, which supports both rule-based and cost-based optimization. The cost-based join reordering optimizations pick the best m-way join order with the Dynamic Programming approach [20].

**Table 3.** Inverted Generational Distance

| | m | NSGA-II | MOEA/D | NSGA-III | NSGA-G |
|---|---|---|---|---|---|
| DTLZ1 | 4 | 3.669e+01 | 2.760e+00 | 1.083e+01 | 3.837e+00 |
| DTLZ3 | 4 | 1.221e+02 | 4.157e+01 | 8.407e+01 | 4.397e+01 |
| DTLZ1 | 5 | 3.482e+01 | 6.816e−01 | 3.315e+00 | 6.939e−01 |
| DTLZ3 | 5 | 3.298e+00 | 4.599e−01 | 1.154e+00 | 2.132e−01 |
| DTLZ1 | 6 | 2.424e+02 | 2.703e+00 | 9.366e+00 | 2.782e+00 |
| DTLZ3 | 6 | 5.083e+02 | 4.021e+01 | 8.398e+01 | 3.222e+01 |
| DTLZ1 | 7 | 5.321e+00 | 4.468e−01 | 4.405e−01 | 6.095e−01 |
| DTLZ3 | 7 | 3.126e+00 | 2.087e−01 | 2.864e−01 | 1.716e−01 |
| DTLZ1 | 8 | 3.974e+02 | 2.843e+00 | 1.204e+01 | 2.589e+00 |
| DTLZ3 | 8 | 7.116e+02 | 3.856e+01 | 1.027e+02 | 2.655e+01 |
| DTLZ1 | 9 | 5.406e+00 | 4.198e−01 | 5.694e−01 | 6.325e−01 |
| DTLZ3 | 9 | 2.825e+00 | 2.243e−01 | 3.974e−01 | 1.994e−01 |
| DTLZ1 | 10 | 1.254e+01 | 3.325e−01 | 1.080e+00 | 6.636e−01 |
| DTLZ3 | 10 | 2.525e+00 | 1.891e−01 | 4.312e−01 | 2.109e−01 |

**Table 4.** Average compute time in IGD experiment (seconds)

| | m | NSGA-II | MOEA/D | NSGA-III | NSGA-G |
|---|---|---|---|---|---|
| DTLZ1 | 4 | 3.144e+01 | 1.650e+02 | 1.194e+02 | 3.548e+01 |
| DTLZ3 | 4 | 3.426e+01 | 1.649e+02 | 1.257e+02 | 3.932e+01 |
| DTLZ1 | 5 | 3.820e+01 | 1.727e+02 | 1.690e+02 | 3.978e+01 |
| DTLZ3 | 5 | 4.084e+01 | 1.737e+02 | 1.739e+02 | 4.330e+01 |
| DTLZ1 | 6 | 4.326e+01 | 1.850e+02 | 1.313e+02 | 4.284e+01 |
| DTLZ3 | 6 | 4.638e+01 | 1.885e+02 | 1.386e+02 | 4.642e+01 |
| DTLZ1 | 7 | 4.780e+01 | 2.034e+02 | 1.332e+02 | 4.554e+01 |
| DTLZ3 | 7 | 5.074e+01 | 2.056e+02 | 1.399e+02 | 4.902e+01 |
| DTLZ1 | 8 | 5.242e+01 | 2.192e+02 | 1.912e+02 | 4.790e+01 |
| DTLZ3 | 8 | 5.602e+01 | 2.217e+02 | 2.019e+02 | 5.156e+01 |
| DTLZ1 | 9 | 5.586e+01 | 2.325e+02 | 2.520e+02 | 5.056e+01 |
| DTLZ3 | 9 | 5.910e+01 | 2.334e+02 | 2.698e+02 | 5.402e+01 |
| DTLZ1 | 10 | 5.910e+01 | 2.398e+02 | 3.372e+02 | 5.218e+01 |
| DTLZ3 | 10 | 6.320e+01 | 2.451e+02 | 3.640e+02 | 5.620e+01 |

**Table 5.** Maximum Pareto Front Error

| | m | NSGA-II | MOEA/D | NSGA-III | NSGA-G |
|---|---|---|---|---|---|
| DTLZ1 | 4 | 8.329e+02 | 2.937e+01 | 5.433e+02 | 1.960e+01 |
| DTLZ3 | 4 | 1.042e+03 | 1.237e+02 | 8.645e+02 | 1.896e+02 |
| DTLZ1 | 5 | 6.860e+02 | 1.006e+01 | 2.633e+02 | 4.940e+00 |
| DTLZ3 | 5 | 2.637e+01 | 1.828e+00 | 1.457e+01 | 4.145e−01 |
| DTLZ1 | 6 | 1.561e+03 | 5.778e+01 | 4.258e+02 | 7.048e+00 |
| DTLZ3 | 6 | 1.817e+03 | 1.840e+02 | 6.905e+02 | 7.986e+01 |
| DTLZ1 | 7 | 7.804e+02 | 6.656e+00 | 1.082e+02 | 2.031e+00 |
| DTLZ3 | 7 | 1.470e+01 | 3.712e−01 | 3.610e+00 | 1.396e−01 |
| DTLZ1 | 8 | 1.961e+03 | 4.591e+01 | 5.125e+02 | 6.087e+00 |
| DTLZ3 | 8 | 2.173e+03 | 2.167e+02 | 8.754e+02 | 6.377e+01 |
| DTLZ1 | 9 | 7.183e+02 | 6.447e+00 | 1.884e+02 | 1.449e+00 |
| DTLZ3 | 9 | 1.783e+01 | 1.920e−01 | 6.497e+00 | 1.038e−01 |
| DTLZ1 | 10 | 7.173e+02 | 7.045e+00 | 2.049e+02 | 9.801e−01 |
| DTLZ3 | 10 | 1.401e+01 | 1.986e−01 | 5.728e+00 | 7.336e−02 |

**Table 6.** Average compute time in MPFE experiment (seconds)

| | m | NSGA-II | MOEA/D | NSGA-III | NSGA-G |
|---|---|---|---|---|---|
| DTLZ1 | 4 | 3.152e+01 | 1.647e+02 | 1.211e+02 | 3.608e+01 |
| DTLZ3 | 4 | 3.448e+01 | 1.655e+02 | 1.265e+02 | 3.938e+01 |
| DTLZ1 | 5 | 3.814e+01 | 1.726e+02 | 1.682e+02 | 3.950e+01 |
| DTLZ3 | 5 | 4.072e+01 | 1.745e+02 | 1.747e+02 | 4.340e+01 |
| DTLZ1 | 6 | 4.392e+01 | 1.845e+02 | 1.314e+02 | 4.282e+01 |
| DTLZ3 | 6 | 4.642e+01 | 1.879e+02 | 1.389e+02 | 4.654e+01 |
| DTLZ1 | 7 | 4.868e+01 | 2.032e+02 | 1.340e+02 | 4.536e+01 |
| DTLZ3 | 7 | 5.076e+01 | 2.060e+02 | 1.390e+02 | 4.912e+01 |
| DTLZ1 | 8 | 5.212e+01 | 2.187e+02 | 1.908e+02 | 4.778e+01 |
| DTLZ3 | 8 | 5.518e+01 | 2.206e+02 | 2.036e+02 | 5.140e+01 |
| DTLZ1 | 9 | 5.582e+01 | 2.309e+02 | 2.545e+02 | 5.040e+01 |
| DTLZ3 | 9 | 5.942e+01 | 2.332e+02 | 2.715e+02 | 5.420e+01 |
| DTLZ1 | 10 | 5.938e+01 | 2.388e+02 | 3.398e+02 | 5.194e+01 |
| DTLZ3 | 10 | 6.320e+01 | 2.423e+02 | 3.648e+02 | 5.668e+01 |

When building m-way joins, they only keep the best plan for the same set of m items. For instance, with tables A, B, C, they keep the best plan and the order of joining $\{A, B, C\}$ instead of all plans (A join B) join C, (A join C) join B and (B join C) join A. This strategy significantly reduces the search space, but could ignore some good plans during MOQO. They estimate the cost of a join operator with a single formula:

$$cost = weight * cardinality + (1.0 - weight) * size \tag{3}$$

Our strategy extends the cost-based join reordering with NSGA-G above for MOQO in multiple cloud environments.

We enhance Catalyst by: 1. Constructing logical plans with dynamic programming [20]. 2. Estimating multi-metric costs via MLR [22], augmented by CBO statistics [5]. 3. Pruning with NSGA-G to yield Pareto-optimal QEPs

### 4.1 Query Processing

Figure 3 shows the sequence of our query processing. When a user puts a query with their Weighted Sum Model, the Spark SQL will generate multiple logical plans with multiple objectives, cost-based optimization with the Dynamic Programming approach. After that, the best plan will be chosen by the Weighted Sum Model and put to Spark SQL to generate and optimize the physical plans.

### 4.2 Integrating NSGA-G in Spark SQL

Figure 4 shows the architecture of generating multiple objectives logical plans step in Fig. 3 with NSGA-G. At the first step, after Spark SQL receives a query, Multiple Linear Regression Algorithm [22] puts the parameter into the model of multiple cost metrics estimation. After that, Spark SQL will generate and optimize logical plans with the Dynamic Programming and multiple cost metric

values. To reduce the search space, the NSGA-G will be applied to find a Pareto plan set. This pruning strategy significantly reduces the search space. At the final step, a Pareto plan set will be put out of the Spark SQL module.

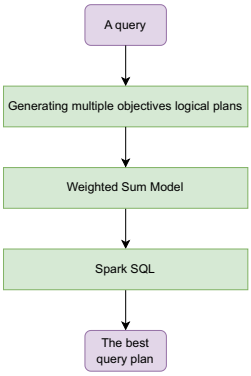


**Fig. 3.** Steps of query processing

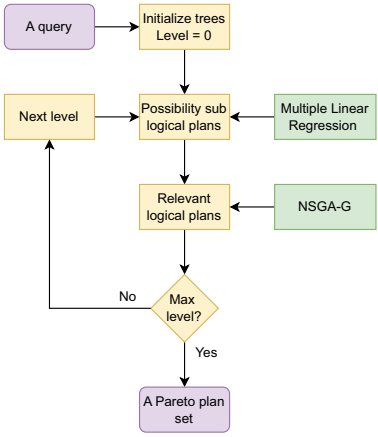


**Fig. 4.** Steps of generating logical plans

### 4.3 Experiment

We implemented our solution in Java, Spark SQL, and Catalyst library. In the optimizer code, [1] used the dynamic programming algorithm in [20] to pick the best $m$-way join order and kept the best plan for the same set of $m$ tables.

Spark's Cost-Based Optimizer estimated the cost of a join operator as Eq. 3. The strategy significantly reduces the search space, but it could ignore some good plans when we consider other metrics, such as the execution time or the monetary cost.

Our experiments keep all the intermediate logical subplans and add the estimated values to all logical plans. The estimated values are determined by Multiple Linear Regression (MLR) with the previous value of each logical plan based on the cardinality. The MLR is used to estimate the parameter $\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_L$, and estimate $\hat{c}$ with the newest value of logical plans. For example, $Q = ((A \text{ join } B) \text{ join } C) \text{ join } D$, our experiment will generate all logical plans for the same set of 4 tables. The cardinality and sizes of each logical plan are the cardinality and size of the intermediate logical subplans of 3 tables for the left side and the cardinality and size of the rest tables for the right side. With the previous value of each logical plan, MLR will estimate the parameter and calculate other cost metrics. After that, NSGA-G will be used to determine a Pareto plan set of logical plans. Finally, the user policy will be used to choose the best plan from Pareto plan set.

Our experiment is implemented with TPC-DS benchmark 1G dataset and a query of joining 5 tables.

## 5 Conclusion

In this paper, we propose a MOQP extension for Spark SQL, combining Multiple Linear Regression Algorithm and NSGA-G to address variance in a large-scale variable environment and to provide accurate estimations for MOQO. We also provide an application of NSGA-G, which has higher performance than the original NSGA-II and NSGA-III in DTLZ 1 and 3 problems. Both algorithms are integrated into Spark SQL module to solve the MOQO for the large-scale variable environment. Future work will validate this proposal with commercial providers and data management systems.

## References

1. Armbrust, M., et al.: Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 1383–1394. ACM (2015). https://doi.org/10.1145/2723372.2742797
2. Cheng, G., Ying, S., Wang, B.: Tuning configuration of Apache Spark on public clouds by combining multi-objective optimization and performance prediction model. J. Syst. Softw. **180** (2021). https://doi.org/10.1016/j.jss.2021.111028
3. Coello, C.A.C., Cortés, N.C.: Solving multiobjective optimization problems using an artificial immune system. Genetic Programm. Evol. Mach. **6**, 163–190 (2005). https://doi.org/10.1007/s10710-005-6164-x
4. Cordero, J.A., et al.: Dynamic multi-objective optimization with jMetal and spark: a case study. In: Pardalos, P.M., Conca, P., Giuffrida, G., Nicosia, G. (eds.) MOD 2016. LNCS, vol. 10122, pp. 106–117. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-51469-7_9
5. Databricks: Cost-based optimizer in Apache Spark SQL. Databricks Documentation (2024). https://docs.databricks.com/spark/latest/spark-sql/cbo.html
6. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014). https://doi.org/10.1109/TEVC.2013.2281535
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
8. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable multi-objective optimization test problems. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), vol. 1, pp. 825–830. IEEE (2002). https://doi.org/10.1109/CEC.2002.1007032
9. Doka, K., Papailiou, N., Tsoumakos, D., Mantas, C., Koziris, N.: IReS: intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD 2015), pp. 1451–1456. ACM (2015). https://doi.org/10.1145/2723372.2735377

10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. Newslett. **11**(1), 10–18 (2009). https://doi.org/10.1145/1656274.1656278
11. Hulgeri, A., Sudarshan, S.: Parametric query optimization for linear and piecewise linear cost functions. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002), pp. 167–178. VLDB Endowment (2002). https://dl.acm.org/doi/10.5555/1287369.1287385
12. Ishibuchi, H., Imada, R., Setoguchi, Y., Nojima, Y.: Performance comparison of NSGA-II and NSGA-III on various many-objective test problems. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 3045–3052. IEEE (2016). https://doi.org/10.1109/CEC.2016.7744174
13. Kolev, B., Bondiombouy, C., Vakluriez, P., Jimenez-Peris, R., Pau, R., Pereira, J.: The CloudMdsQL Multistore System. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD 2016), pp. 2113–2116. ACM (2016). https://doi.org/10.1145/2882903.2899400
14. Le, T.-D., Kantere, V., d'Orazio, L.: An efficient multi-objective genetic algorithm for cloud computing: NSGA-G. In: 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 3883–3888 (2018). https://doi.org/10.1109/BigData.2018.8622148
15. Le, T.-D., Kantere, V., d'Orazio, L.: Dynamic estimation and grid partitioning approach for multi-objective optimization problems in medical cloud federations. In: Hameurlain, A., Tjoa, A.M. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVI. LNCS, vol. 12410, pp. 32–66. Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-662-62386-2_2
16. Le, T.-D., Kantere, V., d'Orazio, L.: Optimizing DICOM data management with NSGA-G. In: Proceedings of the 2019 International Workshop on Data Warehouse Design and OLAP (DOLAP) (2019)
17. Misegiannis, M.G., Kantere, V., d'Orazio, L.: Multi-objective query optimization in Spark SQL. In: Proceedings of the 26th International Database Engineering & Applications Symposium (IDEAS), pp. 70–74 (2022). https://doi.org/10.1145/3548785.3548800
18. Nguyen-Cong, D., d'Orazio, L., Tran, N., Hacid, M.-S.: Storing and querying DICOM data with HYTORMO. In: Wang, F., Yao, L., Luo, G. (eds.) DMAH 2016. LNCS, vol. 10186, pp. 43–61. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57741-8_4
19. Papakonstantinou, Y.: Polystore query rewriting: the challenges of variety. In: EDBT/ICDT Workshops (2016)
20. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (SIGMOD 1979), pp. 23–34. ACM (1979). https://doi.org/10.1145/582095.582099
21. Song, F., et al.: Spark-based cloud data analytics using multi-objective optimization. In: IEEE 37th International Conference on Data Engineering (ICDE), Chania, Greece, 2021, pp. 396–407 (2021). https://doi.org/10.1109/ICDE51399.2021.00041
22. Soong, T.T.: Fundamentals of Probability and Statistics for Engineers. Wiley (2004)
23. Veldhuizen, D.A.V., Lamont, G.B.: Evolutionary computation and convergence to a pareto front. In: Late Breaking Papers at the Genetic Programming 1998 Conference, pp. 221–228 (1998)
24. Veldhuizen, D.A.V.: Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Ph.D. thesis, Air Force Institute of Technology (1999)

25. Zhang, Q., Li, H.: MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. **11**(6), 712–731 (2007). https://doi.org/10.1109/TEVC.2007.892759
26. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evol. Comput. **7**(2), 117–132 (2003). https://doi.org/10.1109/TEVC.2003.810758