Dominance-Based Data Reduction for Package Queries

Vasileios Vittis
University of Massachusetts Amherst
vvittis@cs.umass.edu

Azza Abouzied
New York University Abu Dhabi
azza@nyu.edu

Peter Haas
University of Massachusetts Amherst
phaas@cs.umass.edu

Alexandra Meliou
University of Massachusetts Amherst
ameli@cs.umass.edu

ABSTRACT

Prescriptive analytics workloads often require solving package queries over data that changes continuously. A package query (PQ) returns a multiset of tuples satisfying global constraints and optimizing a given objective, a natural formulation of constrained optimization within a database. When each attribute in the query has a clear "better" direction, e.g., lower cost and higher performance, most candidate tuples are irrelevant: they are strictly worse than others on every dimension and can never appear in an optimal solution. Yet, keeping solutions current as data evolves remains challenging: re-solving from scratch is slow, warm-starting helps only modestly, and solver preprocessing achieves only limited data reduction. We demonstrate SKYPQ, a system that employs a novel, dominance-based data reduction method for such package queries under updates. The key idea is to maintain a *K-skyband index*, a small, correctness-preserving subset of candidates, and apply a *re-solve checker* to skip re-optimization when updates cannot affect the optimal package. The index can be shared by multiple PQs. Via an interactive interface, participants visualize the candidate space collapsing to the K-skyband, solve PQs over the reduced space with exact results, explore what-if scenarios, and observe the system efficiently handling batch updates. Participants experience firsthand how K-skyband reduction provides both speed and correctness for constrained optimization over evolving data.

PVLDB Reference Format:
Vasileios Vittis, Azza Abouzied, Peter Haas, and Alexandra Meliou. Dominance-Based Data Reduction for Package Queries. PVLDB, 19(12): XXX-XXX, 2026.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/umass-dream-lab/skypq.
A live demo is available at https://umass-dream-lab.github.io/skypq/demo.html.

1 INTRODUCTION

Package queries, database queries that select a multiset of tuples optimizing an objective subject to global constraints, capture a broad class of constrained optimization problem directly within the database. They are expensive to solve over large datasets. Yet in many settings, much of the data is *irrelevant*. If we could identify and exclude this irrelevant data upfront, the solver would run on a dramatically smaller input. Consider the following scenario:

Example 1.1 (VM Provisioning). Jordan is a platform engineer provisioning VMs for a data pipeline that requires at least 800 Compute Units (CU). Jordan wants the cheapest bundle meeting this requirement, with practical constraints: at least 8 instances for fault tolerance, at most 12 to respect account quotas, and no more than 2 copies of any single VM type to avoid over-reliance on one configuration. The catalog contains thousands of offers, yet Jordan's package holds only 8–12. Spot prices fluctuate, so Jordan must re-optimize frequently. Yet most options are obviously poor deals: why consider a 20 CU instance at $2/hr when another offers 40 CU at $1/hr? Jordan suspects most of the catalog is irrelevant, but needs a principled way to identify which options matter.

Surprisingly, today's technology cannot help Jordan efficiently. Solving the integer linear program (ILP) over the full catalog takes several seconds, acceptable for a one-time decision but not when the data changes constantly. Warm-starting still considers every VM; the solver's presolve prunes only options dominated by *at least one* other, missing Jordan's limit of 2 copies per type. Greedy heuristics achieve speed but sacrifice optimality; Jordan needs both.

Fortunately, Jordan's instinct that most options in the catalog are irrelevant turns out to be correct. The key is that Jordan's query has *consistent preferences* (lower cost is always better, higher CU is always better), a property we call *COnsistent Preference* (COP). Not all constrained optimization problems have this structure, but many practical package queries do, and it is precisely this structure that determines which options are irrelevant because they are *dominated*, i.e., they are strictly worse than others on every relevant dimension and can never contribute to an optimal solution. The set of non-dominated options is called a *skyline* [2]; this generalizes to a *K-skyband* [5], which accounts for repetition limits like Jordan's 2 copies per VM type. For Jordan's catalog, the K-skyband contains only a fraction of VMs. Crucially, solving over this reduced set is guaranteed to yield the same optimal package as solving over the full catalog.

We demonstrate SKYPQ, an end-to-end system that achieves dominance-based data reduction for COP *package queries* under updates. The key idea is to prune candidates via a K-skyband, excluding tuples that can never appear in any optimal package; SKYPQ maintains this skyband incrementally and applies a re-solve checker that often skips re-optimization entirely, so the same insight that

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

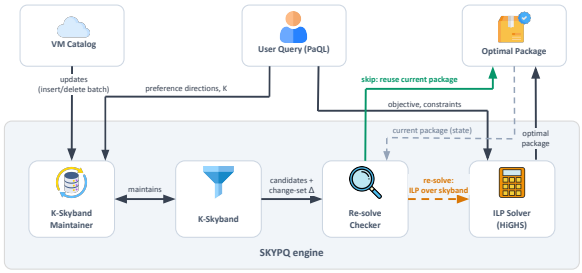

Figure 1: The SKYPQ architecture.

enables reduction also makes maintenance cheap. We proceed to discuss how SKYPQ's approach differs from prior art (Section 2), provide a solution sketch (Section 3), and conclude with a detailed outline of our demonstration (Section 4).

2 CONTRAST WITH PRIOR ART

The problem we address, constrained optimization over evolving data, arises whenever decisions must be made repeatedly as underlying options change.

Solver-based approaches. The most direct strategy is to re-solve the ILP from scratch after every update batch, which is correct but wasteful: the solver considers the full candidate space regardless of whether the update affects the solution. For catalogs with thousands of options, solving takes seconds; acceptable for one-time decisions, but prohibitive when data changes continuously. Warm-starting reuses the previous solution as a starting point, reducing solver iterations, but still considers all tuples and must verify optimality over the full problem; speedups are modest (typically 1.5–2×). LP/ILP presolve [1] includes *dominated column elimination*, mathematically equivalent to computing the 1-skyband over variable coefficients, but has three limitations for our setting: (i) it removes only variables dominated by *at least one* other, whereas package queries with repetition bounds require the K-skyband, since a tuple dominated by $K-1$ others may still appear in optimal packages if its dominators are saturated; (ii) dominance detection is expensive, with solvers capping it at ~1M variable pairs; and (iii) presolve runs from scratch on every solve, whereas our skyband is an index maintained incrementally. Even on a fast machine these reductions run anew on every solve, whereas SKYPQ maintains them as an index and, through the re-solve checker, often skips the solve entirely; the two are complementary rather than competing.

Scalable package query systems. SketchRefine [3] and Progressive Shading [6] scale to large *static* datasets via partitioning and hierarchical representatives, but their structures are costly to maintain under updates, triggering repairs even when the change cannot affect the optimal package, and neither returns an exact solution.

Heuristics. Greedy, local search, and genetic algorithms produce feasible packages quickly but sacrifice optimality, which is unacceptable when decisions have real cost consequences.

The gap. Solver-based approaches treat every update as a fresh problem; heuristics trade correctness for speed; and existing presolve captures only the skyline, not the K-skyband. While K-skyband queries have been studied in the skyline literature, prior work focuses on static computation or incremental maintenance of the 1-skyband (skyline) only [5]. We extend these techniques to maintain the K-skyband incrementally, prove that reduction preserves optimality for package queries, and introduce a re-solve checker that often skips re-optimization entirely. The result is exact optimization at interactive speed.

3 SOLUTION SKETCH

We now provide a solution sketch for SKYPQ. Figure 1 depicts SKYPQ's end-to-end pipeline. SKYPQ registers a package query template, computes the K-skyband for the current data, and maintains it under updates. When the user requests a solution, SKYPQ solves over the skyband; when updates arrive, it maintains the skyband incrementally and applies a re-solve checker to determine whether re-optimization is needed.

Modeling package queries. Following prior work on package queries [3], we model the selection problem as an ILP. A package query selects a multiset of tuples from a relation that optimizes a linear objective subject to aggregate constraints. Each tuple i has a decision variable x_i denoting how many times it appears in the package. Constraints include per-tuple repetition bounds ($0 \le x_i \le \rho$), cardinality bounds ($k_\ell \le \sum x_i \le k_u$), and aggregate constraints on tuple attributes. The Package Query Language (PaQL) provides SQL-like syntax for expressing these queries. Figure 2 shows Jordan's query in PaQL.

COP package queries. Dominance presumes a notion of "better," so we focus on a subclass we call ***COP (COnsistent Preference) PQs***: each constraint and the objective involves a single attribute, and every attribute keeps the same preference direction wherever it appears. In Jordan's query, cost appears only in a MINIMIZE objective (lower is better, direction −1) and CU only in a "≥" constraint (higher is better, direction +1). Consistent directions define dominance: tuple p dominates tuple q if p is at least as good as q on every attribute and strictly better on at least one, and a dominated tuple cannot improve any package. If an attribute has no consistent direction, for example one that is both maximized and minimized, dominance is undefined and the reduction does not apply; SKYPQ then solves the query as a standard ILP over the full relation.

K-skyband reduction. The skyline (the tuples dominated by zero others) is a standard concept in database queries [2]. But for package queries with repetition bounds, the skyline is too aggressive: a tuple dominated by one other might still appear in an optimal package if its dominator is saturated (used ρ times). We generalize to the K-skyband: all tuples dominated by fewer than K others, where $K = \lceil k_u/\rho \rceil$. This captures exactly the tuples that could plausibly contribute to an optimal package. We can prove that optimizing over the K-skyband yields the same optimal value as optimizing over the full relation.

Skyband maintenance. The K-Skyband Maintainer stores the skyband as a Z-order index [5]: tuples are ordered by their Z-address, and each tuple carries its current dominator count as metadata. Because the Z-order places tuples that are close in attribute space close in the index, an update consults only a local neighborhood instead of the whole relation. On insertion, a new tuple is placed by

Jordan's VM provisioning query

```
SELECT  PACKAGE(*) AS P
FROM    vm_offers
REPEAT  EACH P.offer_id AT MOST 2
SUCH THAT
        SUM(P.compute_units) >= 800 AND
        COUNT(P.*)       BETWEEN 8 AND 12
MINIMIZE
        SUM(P.cost_per_hour);
```

Figure 2: Jordan's package query in PaQL. The preference directions define dominance: a VM with higher CU *and* lower cost dominates another.

its Z-address and admitted if fewer than K tuples dominate it; admitting it can raise another tuple's count to K and evict it. Deletion works in the opposite direction: removing a tuple p decrements the count of every tuple that p dominated, so a tuple that was excluded (count at least K) can fall below K and rejoin the skyband, a step we call *promotion*. Each update thus costs a logarithmic lookup by Z-address plus a scan of the affected region, proportional to the number of candidates it touches rather than the size of the relation.[1]

Re-solve checker. After maintenance, the optimal package often does not change, so before invoking the solver we test whether the batch could have affected it. The current package remains optimal when none of its tuples were modified or removed and no inserted or updated tuple enters the skyband where it could improve the objective. Deletions show both cases: removing a tuple outside the package only shrinks the candidate set and leaves the package optimal, even when it promotes other tuples back into the skyband, while removing a tuple from the package calls for a fresh solve. Any such solve runs over the maintained skyband rather than the full relation. Across our experiments, this check skips the solver for the majority of batches.

Efficient package generation. Once the K-skyband is computed, we translate the package query into an ILP and solve it with any standard ILP solver[2]; this is fast because the skyband is far smaller than the full relation.

4 DEMONSTRATION

We demonstrate SKYPQ on a synthetic catalog of 6,500 VM offers to show how participants can observe dominance-based reduction in action and understand why it preserves optimality while dramatically shrinking the search space. We describe the user's interaction through two scenarios (Figure 3), first following Jordan, a platform engineer assembling the cheapest VM bundle that meets a performance target, and then Kulli, a data scientist assembling the highest-performance bundle within a fixed budget.

[1]For portability, the browser demo maintains the skyband incrementally with a linear scan per edit; the Z-order index reduces this scan to a local neighborhood for larger and streaming data.
[2]Our demo uses HiGHS [4], compiled to WebAssembly for browser portability.

The interface is organized into three panels: a central scatterplot displaying the full candidate space, a sidebar with step-by-step controls that guide participants through the workflow, and a dialogue panel that voices the current persona's reasoning, explaining at each step why a strategy works or fails (for example, why a greedy package misses the target) so that participants follow the motivation behind each action, not only its result. The scatterplot shows two attributes, CU and cost, for visual clarity; dominance and the K-skyband are defined over any number of attributes, so higher-dimensional queries are reduced and solved the same way and would be displayed through pairwise two-dimensional projections.

Following Jordan.

Step ① (Query setup): The participant configures the package constraints: at least 800 CU in total, 8–12 VMs, at most 2 copies of any single type, and an objective of minimizing cost. As they adjust these values, the interface previews how each choice shapes the optimization problem, including the derived value of K that drives the reduction.

Step ② (Optimization workflow): The scatterplot initially shows all 6,500 VMs as undifferentiated points. The participant works through a progression of solution strategies, with the visualization updating after each step. Selecting VMs by CU-per-dollar produces a cheap but underpowered package that fails to satisfy the workload target. Selecting by raw CU reaches the target, but at unnecessarily high cost. Solving an ILP over all 6,500 candidates displays the exact optimal package, but takes noticeably longer.

The key step is K-skyband computation. The system applies dominance under the query's preference directions (higher CU and lower cost are preferred), and tuples dominated by at least K others fade to gray. The remaining candidates stay highlighted, and a counter confirms the reduction: fewer than 1% of the original catalog survives. Solving the ILP over this reduced set returns the same optimal package, now marked in green, but much faster. The participant observes the central message: greedy methods are fast but unreliable, full ILP is exact but expensive, and K-skyband reduction preserves exactness while dramatically shrinking the search space.

Step ③ (What-if scenarios): With an optimal package in hand, the participant triggers updates. Removing a VM from the current solution updates dominance, promotes any newly uncovered candidates into the skyband, and re-solves over the revised set. A price change re-solves only if it alters skyband membership or the current optimum, and is otherwise safely skipped.

Step ④ (Playground): Beyond the guided workflow, the participant can interact directly with the candidate space, dragging points to new positions, deleting candidates, or adding new ones anywhere on the scatterplot. Each edit maintains the K-skyband incrementally, updating only the affected dominator counts and promoting points back into the band when a deletion frees them, and then either re-solves over the retained candidates or, when the checker determines the optimum cannot change, skips the solve entirely. The playground reports the per-edit cost live, so participants can weigh incremental maintenance plus the reduced solve against solving the full ILP over all 6,500 points, and watch the checker skip re-optimization when an edit leaves the optimum untouched. This

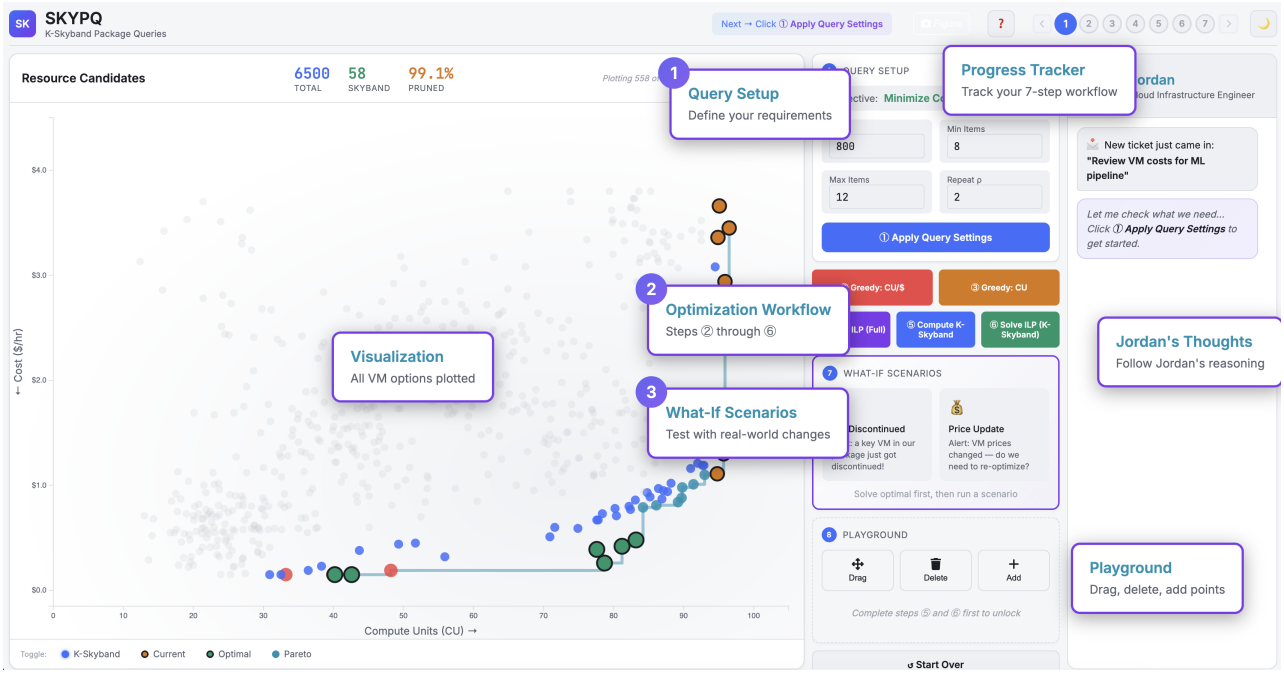

Figure 3: The SKYPQ demo interface guides participants through dominance-based data reduction. ① Query setup: define constraints. ② Optimization workflow: compare greedy heuristics, full ILP, and K-skyband ILP. ③ What-if scenarios: test VM discontinued and price change updates, observe when re-solving is skipped. Also shown: progress tracker, visualization, playground, and Jordan's dialogue.

reinforces that the K-skyband is not a static precomputed index but a structure maintained efficiently under change.

Following Kulli. In our second scenario, the participant switches to Kulli, a data scientist assembling a high-performance VM bundle within a fixed budget. Where Jordan minimizes cost subject to a CU threshold, Kulli maximizes CU subject to a spending limit, a dual formulation over the same attributes. A budget slider lets participants observe how the optimal bundle evolves as constraints tighten. Crucially, Kulli's query reuses Jordan's K-skyband: dominance depends only on preference directions (higher CU, lower cost), not on whether the objective minimizes or maximizes. For catalogs with thousands of options and multiple analysts issuing different queries, the skyband becomes a shared index, computed once and reused across objectives. Participants observe that Kulli's maximization query solves just as fast as Jordan's minimization, both benefiting from the same candidate reduction.

Demonstration engagement. Besides the guided workflow, participants can adjust query parameters and observe how K and the skyband change, hover over points to inspect attributes and dominance relationships, and run additional what-if scenarios with different VMs. A progress tracker and guided tour help first-time users follow the workflow.

Our demonstration showcases how dominance-based reduction makes exact package optimization practical under change. By watching the candidate space contract, observing that the reduced solve returns the same optimum, and triggering updates that the system handles efficiently, participants gain concrete intuition for why most tuples can be safely ignored.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grants 1943971 and 2211918. This work was also supported by NYUAD CITIES, funded by Tamkeen under the Research Institute Award CG001. Generative AI tools were used to assist in implementing the browser-based demonstration; the authors reviewed and verified the resulting code.

REFERENCES

[1] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. 2020. Presolve Reductions in Mixed Integer Programming. *INFORMS J. Comput.* 32, 2 (2020), 473–506. https://doi.org/10.1287/ijoc.2018.0857

[2] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 421–430. https://doi.org/10.1109/ICDE.2001.914855

[3] Matteo Brucato, Juan Felipe Beltran, Azza Abouzied, and Alexandra Meliou. 2016. Scalable Package Queries in Relational Database Systems. *Proc. VLDB Endow.* 9, 7 (2016), 576–587. https://doi.org/10.14778/2904483.2904489

[4] Qi Huangfu and J. A. Julian Hall. 2018. Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10, 1 (2018), 119–142. https://doi.org/10.1007/s12532-017-0130-5

[5] Ken C. K. Lee, Wang-Chien Lee, Baihua Zheng, Huajing Li, and Yuan Tian. 2010. Z-SKY: An Efficient Skyline Query Processing Framework Based on Z-Order. *VLDB J.* 19, 3 (2010), 333–362. https://doi.org/10.1007/s00778-009-0166-x

[6] Anh L. Mai, Pengyu Wang, Azza Abouzied, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. 2024. Scaling Package Queries to a Billion Tuples via Hierarchical Partitioning and Customized Optimization. *Proc. VLDB Endow.* 17, 5 (2024), 1146–1158. https://doi.org/10.14778/3641204.3641222