第一章 绪论

物化视图用作减少针对数据仓库提出的分析查询的响应时间的替代方法。由于无法物化所有视图，并且最佳视图选择是 NP 难问题，因此需要选择合适的视图子集进行物化，以减少分析查询的响应时间。本文使用多目标遗传算法(MOGA)解决了同时最小化两个成本的双目标优化问题。提出的基于 MOGA 的 MVS 算法从多维点阵中选择 Top-K 视图，目的是在上述两个目标之间实现最佳权衡。实现这些选定的 Top-K 视图将减少分析查询的响应时间，从而导致有效和高效的决策制定。

研究问题的背景

如今，有关客户购买行为、营销、销售和产品的数据分析已成为日常业务的重要组成部分。跨国公司利用这些数据分析结果做出关键的业务相关决策，以保持在全球市场上的竞争力。这些公司收集日常交易信息数据，来自遍布全球的多个不同位置，用于分析。
有两种方法可以从这些不同的数据源访问这些数据，即在查询时（按需方法）或通过预先积累数据以供将来查询（提前方法）。后一种方法更方便，因为考虑到数据是在本地累积和存储的，因此可以高效地回答查询。但是，每当源数据库发生更改时，需要频繁更新这些聚合数据。在这种方法中，数据需要存储在中央存储库中，以便进行查询和分析。该中央存储被称为数据仓库。数据仓库存储从多个不同数据源积累的相关数据，用于回答分析问题决策者提出的查询。分析性查询为决策制定而提出的问题庞大而复杂，当针对不断增长的数据仓库提出问题时，这些问题的响应时间很长。减少此查询响应时间的一种方法是具体化数据仓库中的一组适当的视图，可以为决策查询提供答案。与虚拟视图不同，物化视图包含预先计算和聚合的数据及其定义。物化视图比数据仓库小得多，如果选择得当，可以在缩短响应时间的情况下为大多数分析性查询提供答案，从而实现高效决策。MVS 问题与选择这些明智的视图集有关。这个问题可以描述为从所有可能的视图中为给定的数据库模式选择一组适当的视图，以减少查询响应时间，同时将维护和查询处理成本降至最低。视图选择的另一种定义是，针对给定的查询工作负载，在可用存储空间内选择数据库模式上的一组视图。
大多数视图选择算法遵循经验或启发式方法。在经验方法中，视图选择基于过去的查询模式。在基于启发式的方法中，对可能视图的大搜索空间进行修剪，以选择接近最优的视图子集进行具体化。广泛用于选择视图的框架有多维晶格（Multi-Dimensional Lattice, MDL）、和/或视图图（AND/OR view graph）和多视图处理计划（Multi-View Processing Plan, MVPP）。在本文中，视图选择考虑了 MDL 框架。

研究问题的挑战

由于可以物化的可能视图的数量随着维度数量的增加呈指数增长，而且对于更高维度，由于存储空间的限制，不可能物化所有视图。此外，最佳视图选择是一个 NP 难问题。因此，唯一的替代方法是物化一个近似的视图在满足资源约束（即存储空间、视图维护成本等）的同时，适当减少查询响应时间的视图子集。从所有可能的视图中选择这样的视图子集进行物化称为物化视图选择（MVS）。MVS 问题是数据仓库设计中最具挑战性的问题之一。

当前研究工作的不足之处

最基本的视图选择算法，即（Harinarayan et al. 1996）中给出的贪婪算法，以下称为 MVS_{HRUA} ,从 MDL 中选择 Top-K 视图。但是对于高维度来说，MVS_{HRUA} 无法从 MDL 中选择 Top-K 视图。

本文的工作要解决的问题以及方法

由于贪婪算法在高维中不可行，因此选择基于启发式的方法，如 randomized, evolutionary 和 swarm，通过最小化系统的总成本来解决 MVS 问题。本文选择多目标遗传算法（MOGA）解决同时最小化两个成本的双目标优化问题，其中成本分别是物化视图评估的总成本 C_{MV} 与非物化视图评估的总成本 C_{NMV}。

本文的贡献

在本文中，给出了双目标 MVS 问题，提出了一种基于 MOGA 的物化视图选择算法，该算法从 MDL 中选择 Top-K 视图。如果采用物化视图，可以在缩短查询响应时间的情况下为大多数分析查询提供答案，从而实现有效和高效的决策。

章节安排

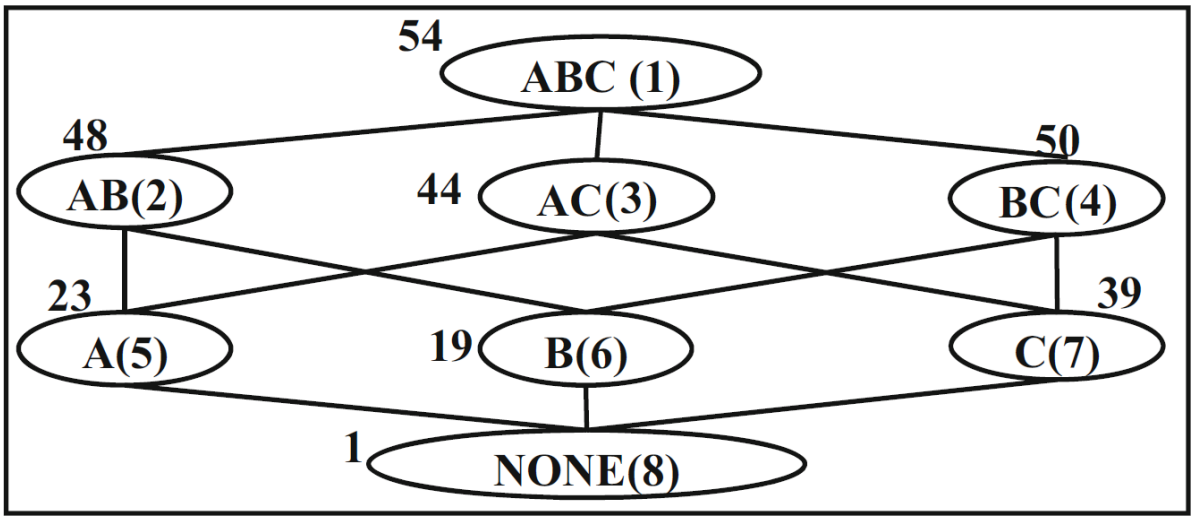
首先介绍本次实验使用的系统/方法框架和用到的技术，接着实现本次实验，最后展示本次实验的结果。

第二章 系统/方法框架

本次实验使用 PySpark 实现。Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark 是 UC Berkeley AMP lab (加州大学伯克利分校的 AMP 实验室)所开源的类 Hadoop MapReduce 的通用并行框架，Spark，拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是——Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。

第三章 Multi-Dimensional Lattice

在本文中，视图选择考虑了 MDL 框架。MDL 描述了查询中涉及的所有可能视图。网格的节点表示一个可能的视图，而根节点（即最顶层视图）表示星形图式中的事实表。MDL 以这样的方式描述视图之间的直接或间接依赖关系，即任何视图上的查询都可以由其任何祖先视图来回答。以下图为例，根视图节点 ABC 表示事实表，晶格中的其余视图节点（AB、AC、BC、A、B、C、None）表示关于维度 A、B 和 C 上可能分组的其余可能视图。视图大小显示在节点旁边。



我们的目的是在这些视图中选择 Top-K 个视图物化，使得

$$Minimize\ C_{MV} = \sum_{v \in MV} Size(v) \tag{1}$$

$$Minimize\ C_{NMV} = \sum_{v \notin MV} SizeSMA(v) \tag{2}$$

其中 MV 表示物化的视图组成的集合，Size(v)为该视图的大小，SizeSMA(v)为该视图代价最小的物化视图祖先的大小。这是因为 C_{MV} 表示查询物化视图的代价，即储存物化视图所需的代价； C_{NMV} 表示查询非物化视图的代价，即查询这类视图的最小祖先物化视图所需的代价。因此根节点一定要物化[3]，因为它没有祖先视图。

第四章 Multi-objective Genetic Algorithm

上面讨论的 MVS 问题需要同时优化两个目标函数，本文采用多目标遗传算法 MOGA 解决 MVS 问题。

4.1 多目标遗传算法

对于多目标问题，目标通常是相互冲突的，这妨碍了每个目标的同时优化。大多数时候，真正的工程问题实际上有多个目标，即最小化成本，最大化性能，最大化可靠性等。这些都是困难但现实的问题。遗传算法是一种流行的元启发式算法，特别适合这类问题。传统的遗传算法通过使用专门的适应度函数和引入促进解多样性的方法来适应多目标问题。

多目标优化有两种通用方法。一种是将单个目标函数组合成单个复合函数，或将除一个目标外的所有目标移动到约束集。在前一种情况下，可用效用理论、加权和法等方法确定单个目标，但问题在于正确选择权重或效用函数来描述决策者的偏好。实际上，即使对于熟悉问题领域的人来说，也很难精确地选择这些权重。使这一缺点更加复杂的是，需要在目标之间进行缩放，权重中的小扰动有时会导致完全不同的解。在后一种情况下，问题是要将目标移动到约束集，必须为每个前一个目标建立约束值。这可能是相当武断的。在这两种情况下，优化方法都会返回一个单一的解，而不是一组可以权衡的解。出于这个原因，决策者通常更喜欢考虑多个目标的一组好的解。

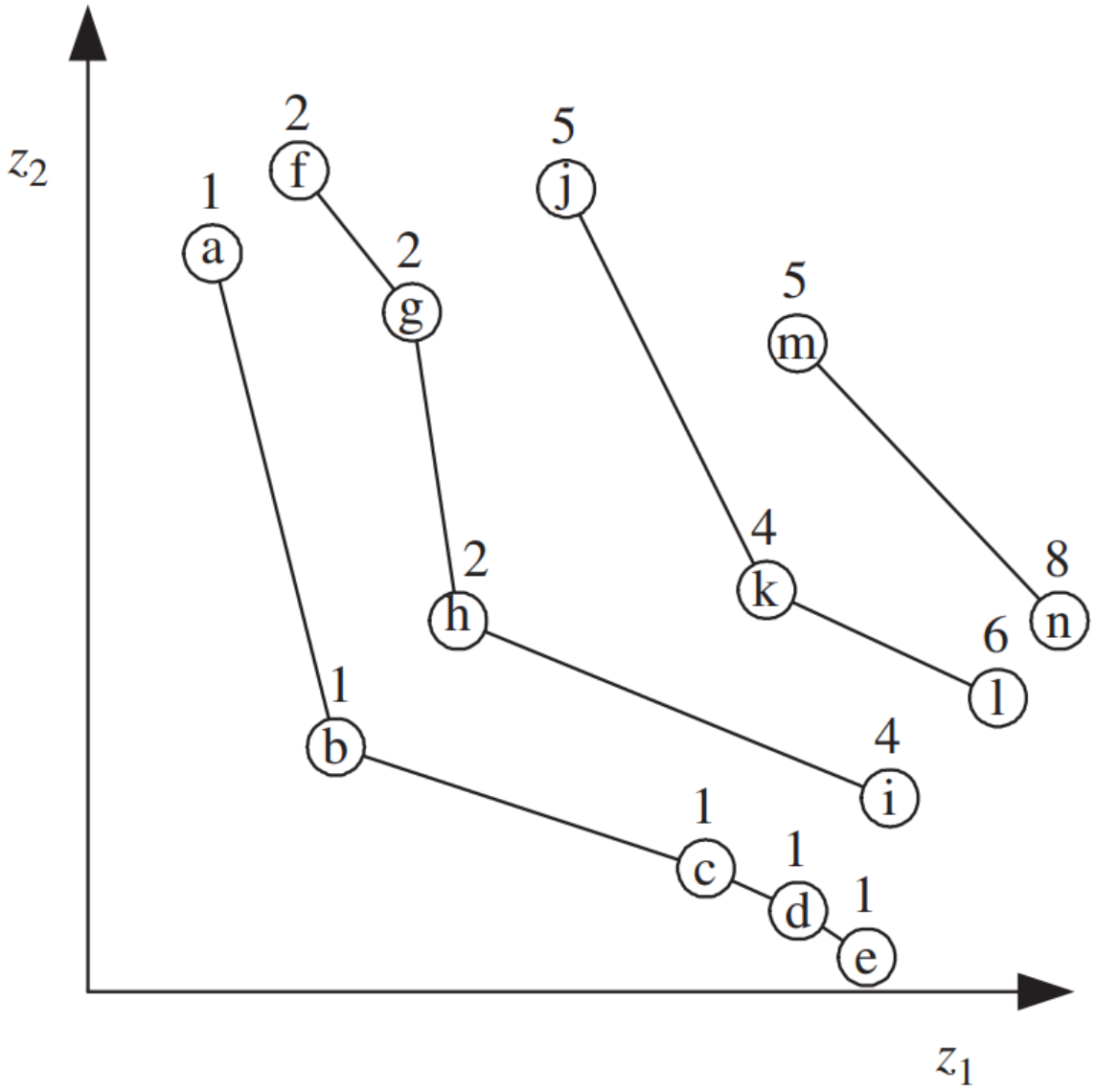
第二种通用方法是确定整个帕累托最优解集（Pareto optimal solution）或代表子集。帕累托最优集是一组互不占优的解。当从一个帕累托解转移到另一个时，在一个目标中总是有一定数量的牺牲，以在另一个目标中实现一定数量的收益。帕累托最优解集通常优于单一解，因为它们在考虑实际问题时是实用的因为决策者的最终解始终是一种权衡。帕累托最优集可以有不同的大小，但帕累托集的大小通常随着目标数量的增加而增加。[2]

4.2 Pareto-ranking approaches

帕累托排序方法明确利用了帕累托优势（Pareto dominance）在评估适应度或分配选择概率给解方面的作用这一概念。根据优势规则对种群进行排序，然后对每个解进行排序根据其在总体中的排名而不是实际的目标函数值分配适应度值。请注意，此处假设所有目标都最小化，因此排名（rank）越低，对应的解越好。MOGA 使用的计算 rank 的算法如下：

$$r_2(\mathbf{x},t) = 1 + nq(\mathbf{x},t) \tag{3}$$

其中 $nq(\mathbf{x},t)$ 表示在 t 代时支配解 $\mathbf{x}$ 的解的数量。该排序方法惩罚位于被帕累托前沿的种群的稠密部分支配目标函数空间区域的解，帕累托前沿（Pareto front）是指帕累托解对应的目标函数值。例如，在下图中，解 i 被解 c、d 和 e 支配。因此，虽然它与仅由一个解支配的解 f、g 和 h 处于同一前沿，但它的 rank = 4。



虽然 rank 方法可直接用于为单个解分配适应值，但通常将其与各种适应值共享技术相结合，利于找到多样且一致的帕累托前沿。

4.3 种群多样性：Fitness sharing, niching

在多目标遗传算法中，保持种群的多样性是获得均匀分布于帕累托前沿的解的一个重要考虑因素。在没有采取预防措施的情况下，群体往往在多目标遗传算法中形成相对较少的聚类。这种现象被称为遗传漂变（genetic drift），MOGA 采用了一些方法来防止遗传漂变。

适应度共享通过人为降低种群稠密区域解的适应度，鼓励在帕累托前沿未经探索的区域进行搜索。通过确定种群稠密区域并对其中的解进行惩罚来实现这一目标，具体如下：

Step 1：计算在 0 到 1 的规范化目标空间中的解 x 与 y 之间的欧式距离

$$dz(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{k=1}^{K}\left(\frac{z_k(\mathbf{x}) - z_k(\mathbf{y})}{z_k^{max} - z_k^{min}}\right)^2} \tag{4}$$

其中 z_k^{max}（z_k^{min}）是当前最大（最小）目标函数值。

Step 2：基于上述距离计算每个解 $\mathbf{x} \in P$ 的 niche count 如下：

$$nc(\mathbf{x},t) = \sum_{\mathbf{y}\in P, r(\mathbf{y},t)=r(\mathbf{x},t)} max\left\{\frac{\sigma_{share} - dz(\mathbf{x},\mathbf{y})}{\sigma_{share}}, 0\right\} \tag{5}$$

其中 σ_{share} 表示 niche size。

Step 3：计算完 niche count 后，每个解的适应度调整如下：

$$f'(\mathbf{x},t)=\frac{f(\mathbf{x},t)}{nc(\mathbf{x},t)} \tag{6}$$

在上述过程中，σ_{share} 定义了目标空间中解的邻域。同一邻域中的解会对彼此的 niche count 产生影响。因此，拥挤邻域中的解将具有更高的 niche count，从而降低选择该解作为父解的概率。因此，niching 限制了解在目标函数空间中某一特定邻域的扩散。

4.4 MOGA

MOGA 是第一个明确使用基于帕累托排序和 niching 技术的多目标遗传算法，以鼓励在保持种群多样性的同时搜索真正的帕累托前沿。MOGA 的流程如下所示:

Step 1：以随机初始种群 P_0 开始，将 t 置为 0。

Step 2：若满足循环终止条件，返回 P_t 。

Step 3：按以下步骤评估适应度：

Step 3.1：用式(3)计算每个解 $\mathbf{x}\in P_t$ 的 $rank(\mathbf{x},t)$

Step 3.2：基于解的 rank 用下式计算每个解的适应值：

$$f(\mathbf{x},t)=N-\sum_{k=1}^{r(\mathbf{x},t)-1}n_k-.5\times(n_{r(\mathbf{x},t)}-1)$$

其中 n_k表示 rank=k 的解的个数。

Step 3.3：用式(5)计算每个解 $\mathbf{x}\in P_t$ 的 niche count $nc(\mathbf{x},t)$

Step 3.4：用式(6)计算每个解 $\mathbf{x}\in P_t$ 的共享适应值 $f'(\mathbf{x},t)$

Step 3.5：用下式规范化适应值：

$$f''(\mathbf{x},t)=\frac{f'(\mathbf{x},t)n_{r(\mathbf{x},t)}}{\sum_{\mathbf{y}\in P_t,r(\mathbf{y},t)=r(\mathbf{x},t)}f'(\mathbf{x},t)}f(\mathbf{x},t)$$

Step 4：利用基于 f'' 的随机选择方法选择交配池中的亲本。在交配池中应用交叉和变异，直到产生子代种群 Q_t 的大小为 N。令 $P_{t+1}=Q_t$

Step 5：令 $t=t+1$ ，回到 Step 2。

第五章 实验

基于Ubuntu 20.04，pyspark版本为spark-3.2.0-bin-hadoop3.2-scala2.13，数据集由python脚本生成。

实验设计

INPUT:
带有各个视图大小的视图格 L、种群的 Top-K 视图 PTKV、交叉概率 P_c 、变异概率 P_m 、选择视图数 K、最大生成子代数 G_M。

$$Minimize\ \ C_{MV}=\sum_{v\in MV}Size(v)$$

$$Minimize\ C_{NMV} = \sum_{v \notin MV} SizeSMA(v)$$

其中 MV 表示物化的视图组成的集合，NMV 表示不物化的视图组成的集合，Size(v)为该视图的大小，SizeSMA(v)为该视图代价最小的物化视图祖先的大小。

OUTPUT:

Top-K 视图集合 TKV。

METHOD:

Step 1: 随机生成有 N 个 Top-K 视图向量初始种群 PTKV，每个 Top-K 视图向量 TKV 的大小为 K。初始化 G=1。

REPEAT

Step 2: 选择一个 σTKV_{share} 的值，对 i=1...N 的所有 rank RTKV 初始化 $\mu(RTKV_i) = 0$。令 Top-K 视图向量 TKV 的计数器 i=1。

Step 3: 计算支配 $DTKV_i$ 个 Top-K 视图向量 TKV_i 的 Top-K 视图向量数量。用下式计算 TKV_i 的 rank $RTKV_i$：

$$RTKV_i = 1 + DTKV_i$$

将表示 rank 为 $RTKV_i$ 的计数器加一，即 $\mu(RTKV_i) = \mu(RTKV_i) + 1$。

Step 4: 如果 i< N，i 加一并回到 Step 2。否则，跳到 Step 5。

Step 5: 确定最大 rank $RTKV^*$ 且其 $\mu(RTKV_i) > 0$。按 rank RTKV 排序，用下式计算每个 Top-K 视图向量 TKV_i 的平均适应度 $FTKV_i$：

$$FTKV_i = N - \sum_{k=1}^{RTKV_i - 1} \mu(k) - 0.5(\mu(RTKV_i) - 1)$$

Step 6: 对每个 rank 为 RTKV 的 Top-K 视图向量 TKV_i，用下式计算 niche count $NCTKV_i$：

$$NCTKV_i = \sum_{j=1}^{\mu(RTKV_i)} Sh(dTKV_{ij})$$

其中 $dTKV_{ij}$ 是任意两个 rank 相同的 Top-K 视图向量 TKV_i 和 TKV_j 之间的规范化距离，共享函数值 $Sh(dTKV_i j)$ 通过下式定义：

$$Sh(dTKV) = \begin{cases} 1 - \left(\dfrac{dTKV}{\sigma TKV_{share}}\right)^{\alpha}, dTKV \le \sigma TKV_{share}; \\ 0, otherwise \end{cases}$$

其中 σTKV_{share} 是考虑共享效应的 Top-K 视图向量间的最小距离。

Step 7: 对每个 rank 为 $RTKV$ 的 Top-K 试图向量 TKV_i，计算共享适应度如下：

$$FTKV'_j = FTKV_j / NCTKV_j$$

然后规范化共享适应度如下：

$$SFTKV'_i = \frac{FTKV_i \times \mu(RTKV_i)}{\sum_{k=1}^{\mu(RTKV)} FTKV'_k} \times FTKV'_i$$

Step 8: 应用基于共享适应度随机选择 Top-K 视图向量到交配种群 MTKV 中。

Step 9: 以概率 P_c 应用单点交叉，以概率 P_m 随机变异，生成新的 Top-K 视图向量种群 PTKV'。

UNTIL $G \leq G_M$

RETURN TKV=Top-K 视图向量。

实验结果

初始随机种群如下：

```
0
(0, (8, 13, 11, 12, 10, 14, 7, 15), (648, 359))
(1, (2, 7, 11, 12, 4, 3, 15, 13), (496, 483))
(2, (4, 9, 15, 5, 12, 8, 6, 10), (503, 467))
(3, (5, 7, 10, 3, 13, 2, 14, 11), (479, 523))
(4, (7, 11, 15, 10, 6, 9, 8, 5), (512, 448))
(5, (7, 8, 11, 13, 9, 10, 5, 4), (484, 507))
(6, (12, 3, 7, 9, 13, 14, 8, 5), (522, 440))
(7, (4, 8, 14, 3, 13, 9, 5, 2), (436, 561))
(8, (7, 3, 4, 15, 2, 9, 5, 13), (435, 556))
(9, (11, 9, 3, 14, 6, 15, 5, 13), (560, 455))
```

50轮迭代后种群如下：

```
49
(0, (7, 16, 8, 4, 6, 5, 9, 13), (398, 591))
(1, (7, 16, 15, 12, 6, 13, 2, 11), (496, 481))
(2, (7, 16, 12, 4, 3, 5, 14, 13), (448, 556))
(3, (7, 16, 14, 12, 6, 5, 2, 13), (453, 508))
(4, (7, 16, 12, 4, 6, 5, 14, 13), (468, 518))
(5, (7, 16, 12, 4, 6, 5, 14, 13), (468, 518))
(6, (7, 16, 14, 12, 6, 5, 2, 13), (453, 508))
(7, (7, 16, 15, 12, 6, 13, 2, 11), (496, 481))
(8, (7, 16, 15, 12, 6, 13, 2, 11), (496, 481))
(9, (7, 16, 8, 4, 6, 5, 9, 13), (398, 591))
```

用户可以根据最终结果以及对两个目标的权衡决定物化哪些视图。

第六章 相关工作

1. 本次实验还有可以改进的地方，例如将单点交叉改为其他交叉方法；或者编码方式上进一步研究，以求获取更好的效率。
2. 可以考虑将物化视图与学习索引相结合，使用上述查询物化视图和非物化视图的代价的加权和作为损失函数，训练模型。同时为了提高模型的准确率，使用层级模型。

第七章 结论

在本文中，给出了双目标MVS问题，其中两个目标是查询物化视图和非物化视图的代价。为此， 提出了一种基于 MOGA 的物化视图选择算法MVS_{MOGA}，该算法从MDL中选择Top-K视图。如果物化MVS_{MOGA}选择的Top-K视图，可以在缩短查询响应时间的情况下为大多数分析查询提供答案，从而实现有效和高效的决策。

参考文献

[1] Prakash, J., & Vijay Kumar, T. V. (2020). Multi-objective materialized view selection using MOGA. International Journal of System Assurance Engineering and Management. doi:10.1007/s13198-020-00947-2

[2] Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety, 91(9), 992–1007. doi:10.1016/j.ress.2005.11.018

[3] Harinarayan, V., Rajaraman, A., & Ullman, J. D. (1996). Implementing data cubes efficiently. ACM SIGMOD Record, 25(2), 205–216. doi:10.1145/235968.233333