On the Complexity of the View-Selection Problem

Howard Karloff [*] Milena Mihail [†]

Abstract

A commonly used and powerful technique for improving query response time over very large databases is to precompute ("materialize") frequently asked queries ("views"). The problem is to select an appropriate set of views, given a limited amount of resources. Harinarayan, Rajaraman and Ullman formalized this technique by proposing a framework in which queries are modeled by a weighted partial order, and selecting a set of views whose materialization minimizes the average query response time is equivalent to selecting a subset of nodes of the partial order that minimizes a suitably defined cost function. Because this problem is NP-Hard, the focus is on approximability and heuristics. Harinarayan, Rajaraman and Ullman proposed a greedy heuristic together with a "benefit" criterion to measure its performance; this heuristic and performance measure are used in several subsequent papers which generalize their work.

We prove the following lower bounds: **(a)** The greedy heuristic of Harinarayan, Rajaraman and Ullman has query response time at least $n/12$ times optimal for infinitely many n. (Compare this to the fact that no algorithm, regardless of how naive it is, ever has query response time exceeding n times optimal.) **(b)** If P$\neq$NP, then for every $\epsilon > 0$, every polynomial-time approximation algorithm for the view-selection problem will output solutions with query response time at least $n^{1-\epsilon}$ times optimal, for infinitely many n, even for partial orders with bounded degrees and bounded depth. **(c)** A similar result applies even if we generously allow the algorithm to materialize αk views, α a constant, and compare its performance to the optimal achievable when k views are chosen.

Our results prove (if P$\neq$NP) that the view-selection problem is essentially inapproximable for general partial orders (the "benefit" performance measure of Harinarayan, Rajaraman and Ullman provides no competitiveness guarantee against the optimal solution). Hence studies of the Harinarayan, Rajaraman and Ullman framework and its generalizations should focus on special cases of practical significance, such as hypercubes, and on experimental comparison of heuristics.

[*] College of Computing, Georgia Institute of Technology, howard@cc.gatech.edu. Research supported in part by NSF grant CCR-9732746.

[†] College of Computing and School of Industrial and Systems Engineering, Georgia Institute of Technology, mihail@cc.gatech.edu. Work done in part while the author was at Bellcore.

1 Introduction

In decision support as well as data mining applications, users query very large databases, called data warehouses, to carry out their analyses. Such analyses are becoming increasingly important in maintaining business competitiveness, and therefore optimizing query response times is a critical issue. A commonly used and powerful technique for improving query response times in very large decision support databases is to precompute frequently asked queries. Harinarayan, Rajaraman and Ullman [6] formalized this technique by modeling query response time and dependencies among queries with a weighted partial order, and stating the problem of minimizing query response time in explicit combinatorial terms. The VIEW SELECTION problem is precisely the optimization problem that arises in the above context; natural generalizations were subsequently introduced in [3, 4, 5, 9]. In this paper we study the complexity of the VIEW SELECTION problem, thus providing some guidance on promising and not so promising strategies to attack it.

In particular, since the VIEW SELECTION problem is NP-Complete, even in its simplest form, we focus on efficient heuristics. Indeed, [6] proposes such a heuristic which greedily improves a certain "benefit" of precomputing queries; this heuristic carries over to all generalizations of their work [3, 4, 5, 9]. Of course, the quality of the solutions produced by a heuristic for an NP-Hard problem must be evaluated either by a formal argument or by experimental evidence. Toward the former end, [6] introduces a "benefit" criterion to measure performance. On the other hand, the standard criterion by which theoretical computer science measures performance is the "performance ratio," which compares the solutions produced by the heuristic to the optimal solution. The performance ratio measures competitiveness in a pragmatic way, while the benefit criterion of [6] can be inconclusive.

Can we get an efficient approximation algorithm with good performance? Our inapproximability result (Theorem 1) suggests that, when the criterion is worst case performance ratio, the answer is negative: no polynomial time algorithm can achieve worst case performance ratio better than $n^{1-\epsilon}$, if P≠NP. Clearly, all inapproximability results for VIEW SELECTION apply to all generalizations of VIEW SELECTION, such as those of [3, 4, 5, 9]. We are thus left with the options of using worst case performance ratios for evaluating heuristics on special instances only, or using a different evaluation method, such as experiments. For example, hupercubes form a class of instances with special structure for which there is no evidence against the existence of an efficient heuristic with provably good performance ratio; hypercubes are also of practical significance. However, our inapproximability results (Theorem 1 and Sections 3.1 and 3.2) further suggest that such an algorithm has to take rather "detailed" advantage of the special structure of the hypercube, e.g., taking advantage of depth and degree bounds alone will not suffice.

The rest of the paper is organized as follows. The VIEW SELECTION problem of Harinarayan, Rajaraman and Ullman is formally defined in Section 1.1. We follow the line of [6] which explains the practical context in which the problem arises, thus also exemplifying the importance of hypercubes. In Section 1.2 we review the basic theoretical performance measure of heuristics for NP-Hard problems. In Section 1.3 we compare the standard theoretical performance measure to the "benefit" criterion of [6] and review our inapproximability results. Section 2 contains the proof that no good approximation algorithm exists (if P≠NP) for VIEW SELECTION. Section 3 provides some hard-to-approximate special cases and shows that allowing the algorithm to materialize a constant factor more views than the optimal algorithm (i.e., relaxing slightly the performance measure) doesn't help. We analyze the algorithm of [6] in Section 4. In Section 5 we mention important special cases of VIEW SELECTION that remain open and outline heuristics worthy of further experimental evaluation.

1.1 Partial Order Models, Data Cubes, and the View-Selection Problem

Consider the TPC-D benchmark instance which exemplifies the special case of partial orders that are hypercubes, or "data cubes" (Example 1.1 in [6]): The database consists of roughly $6 \cdot 10^6$ rows of raw data of the form psc, where p is the name of a part, s is the name of a supplier, c is the name of a customer and $S(p, s, c)$ is the number of sales to customer c of part p which the company bought from supplier s. See the cube in Figure 1a. Now suppose that we want to determine the total number of sales of part p. By scanning the raw data the response time for this query would be proportional to $6 \cdot 10^6$. However, one can precompute or "materialize" the view associated with parts only by computing and storing, for each part p, the total number of sales of p, which is $S_p = \sum_{s,c} S(p, s, c)$. If this view has $0.2 \cdot 10^6$ rows, then, once the view is materialized, the query response time would be proportional to $0.2 \cdot 10^6$. In general, one can answer a query by scanning any materialized view which is sufficiently detailed to contain the desired information. For example, the total number of sales of part p can be also inferred by scanning the view $S_{ps} = \sum_c S(p, s, c)$ associated with parts and suppliers, and computing $S_p = \sum_s S_{ps}$ in time proportional to $0.8 \cdot 10^6$. Of course, one would refer to the smallest materialized view which contains the information. Since materializing views requires resources such as preprocessing time, storage space, etc., the problem is to select an appropriate set of views that minimizes average query response time.

In general, the Harinarayan et al. framework assumes that dependencies among n queries are modeled by a partial order on n nodes, with a unique largest node representing the raw data. We henceforth refer to nodes of the partial order as *views*, and we call the largest node the *root*. In the special case of hypercubes, if there are N attributes associated with the data then there are $n = 2^N$ views altogether, with $u \preceq v$ if and only if the set of attributes associated with u is a subset of the set of attributes associated with v. Other cases of partial orders arise, for instance, as product graphs. Figure 1b is such a product graph (Example 2.1 in [6]).

Furthermore, the Harinarayan et al. framework models query response time for a precomputed view u with a nonnegative cost function $rows(u)$. When some subset R of the views is precomputed, or *materialized*, then any query u can be answered by scanning any materialized view $v \in R$ that is an ancestor of u in the partial order. Thus we define the cost $cost_R(u)$ of a view u with respect to R as the minimum value of $rows(v)$ over all ancestors v of u that are materialized:

$$cost_R(u) = \min_{v \in R:\, u \preceq v} rows(v).$$

Finally, assuming that all queries are equally likely, the cost of the implementation R is the average cost over all views:

$$\frac{1}{n} \sum_u cost_R(u).$$

and we shall use the notation $totalcost_R = \sum_u cost_R(u)$.

Formally, VIEW SELECTION is this problem: Given a positive integer k and a rooted partial order in which each node u has a nonnegative weight *rows*(u), find an R which minimizes $totalcost_R = \sum_u cost_R(u)$ among those sets R which contain the root and have size $k+1$. (Since one can always get the answer from laboriously scanning the raw data, we assume that the root, which represents the raw data, is always materialized.) For general partial orders, Harinarayan et al. observe [6] that the view-selection problem is NP-Hard via a reduction from SET COVER.

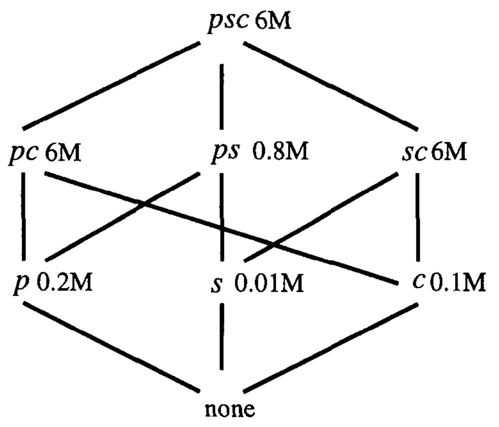

Figure 1a. The TPC-D benchmark example is a hypercube.

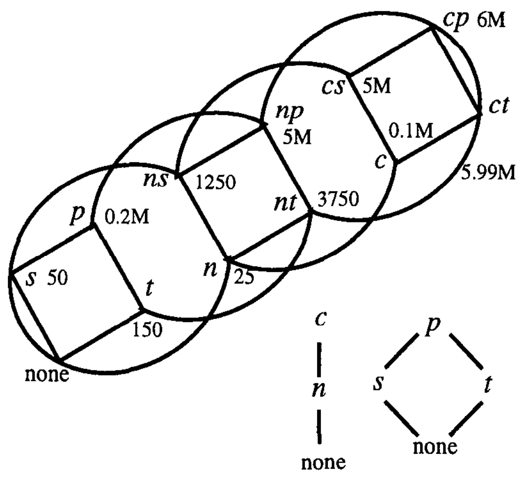

Figure 1b. More general partial orders arise, for instance, as product graphs. Above is the product of a hierarchy of customer attributes (c =customer, n =nationality) with a hierarchy of part attributes (p =part, s =size, t =type).

1.2 Performance of Approximation Algorithms

The *performance ratio* of an approximation algorithm is the worst case ratio of the cost of the solution it achieves, relative to that of the optimal, perhaps parameterized by the size of the problem. For our problem, the most trivial algorithm is to materialize nothing except *root*, which must be materialized. Defining $M := rows(root)$, the trivial situation $R = \{root\}$ yields $cost_R(u) \leq M$ for all u, and hence $totalcost_R \leq n \cdot M$. Even if we were allowed to materialize *every* node, we would still have $cost_R(root) = M$, and hence $totalcost_R \geq M$ for *any* R. So since the total cost of the optimal R is at least M, and the trivial R achieves total cost at most $n \cdot M$, the performance ratio of every algorithm is at most n.

1.3 Our Results

Now we have enough background to describe our results. First and foremost, we prove that if P$\neq$NP, then for every $\epsilon > 0$, for every polynomial-time approximation algorithm A for VIEW SELECTION, the performance ratio of algorithm A is at least $n^{1-\epsilon}$ for infinitely many values of n. (Since the performance ratio cannot exceed n, this is almost as bad as it can get.) In other words, assuming P$\neq$NP, there is no good approximation algorithm for this problem. Second, we prove that in some important special cases, the problem remains hard to approximate. Third, we show that even allowing the algorithm to materialize a constant factor more views doesn't help. Fourth, we study the approximation algorithm proposed by Harinarayan, Rajaraman and Ullman in [6] and show (without any assumption on P vs. NP) that its performance ratio is at least $n/12$ for infinitely many n.

Let us contrast these results with those of [6]. Instead of minimizing average cost or $totalcost_R$, their algorithm *maximizes* the complementary objective function. Precisely, instead of seeking R of size at most k to minimize the response time $totalcost_R = \sum_u cost_R(u)$, they seek an R to maximize the improvement of the response time over the trivial solution with $R = \{root\}$. The value of the improvement is $\sum_u (M - cost_R(u)) = nM - \sum_u cost_R(u)$. For this complementary problem, they achieve a performance ratio of at least $(e-1)/e > 0.632$ via an analysis reminiscent of that of set coverage [7, page 136]: the cost their algorithm achieves is at least 63% of the optimal value of $\sum_u (M - cost_R(u))$. Of course, an optimal solution to the complementary maximization problem immediately yields an optimal solution to the original minimization problem. However, achieving a constant performance ratio for the complementary problem yields *no performance guarantee whatsoever* for the original problem. For example, if the average response time of the trivial implementation is T and the average response time of the optimal solution is $0.01T$, for an improvement of $0.99T$, then a solution whose improvement is $0.8T$, or a respectable $0.8/0.99 > 80\%$ of the optimal improvement $0.99T$, has response time $0.20T$, which is a poor 20 times optimal. Naturally, the relevant resource in this context is response time, which is time spent, rather than improvement in response time over a trivial implementation. Therefore, it is indeed important to look at the response time minimization problem rather than the complementary maximization problem.

For response time minimization, our results prove (if P$\neq$NP) that the VIEW SELECTION problem is essentially inapproximable for general partial orders. Therefore, future theoretical studies should focus on special cases, such as hypercubes or product graphs, for which the problem might be easier. There is no known lower bound or even hardness result for hypercubes, and there is no known algorithm with nontrivial performance guarantee either. For applications, it is also important to extend the studies of Harinarayan et al. [6], who implemented their greedy heuristic, and to compare the performance of further heuristics in practice.

2 A Lower Bound for Every Polynomial-Time Algorithm

In this section we prove that no good polynomial-time algorithm exists for the minimization problem. We do so via a reduction from EXACT COVER BY 3-SETS (X3C): Given a positive integer a which is a multiple of 3, and a set $\{S_1, S_2, ..., S_m\}$ of m distinct 3-element subsets of the universe $U := \{v_1, v_2, v_3, ..., v_a\}$, and such that no element appears in more than three sets, is there a set $I \subseteq \{1, 2, 3, ..., a\}$, $|I| = a/3$, such that $\cup_{i \in I} S_i = U$? In other words, is there an exact cover of the universe by a collection of the given 3-sets? Garey and Johnson [2, page 221] state that EXACT COVER BY 3-SETS is NP-Complete.

Now define RESTRICTED EXACT COVER BY 3-SETS (RX3C) to be this problem: Given a universe $U = \{v_1, v_2, ..., v_a\}$ with $3|a$ and distinct 3-sets $S_1, S_2, ..., S_a \subseteq U$, such that for each $v \in U$, there are at most four sets S_i containing v, is there a subset $I \subseteq \{1, 2, ..., a\}$, $|I| = a/3$, such that $\cup_{i \in I} S_i = U$? (Notice that now the number of sets matches the size of the universe.)

Because it is not hard to reduce EXACT COVER BY 3-SETS to RESTRICTED EXACT COVER BY 3-SETS (though we omit the proof), we infer that RESTRICTED EXACT COVER BY 3-SETS is NP-Hard.

Theorem 1 *If for some $\epsilon > 0$ there is a polynomial-time approximation algorithm A_ϵ for* VIEW SELECTION *with performance ratio at most $n^{1-\epsilon}$ on all n-node, rooted, weighted dags when n is sufficiently large, then P=NP.*

Proof. Choose any $\epsilon > 0$ such that algorithm A_ϵ exists. We now show that if A_ϵ's performance ratio is less than $n^{1-\epsilon}$ for all sufficiently large n, then A can solve RX3C in polynomial time.

Given an instance of RX3C, construct a rooted dag, a nodes $S_1, S_2, ..., S_a$ corresponding to the a 3-sets, a nodes $v_1, v_2, ..., v_a$ corresponding to the elements of the universe, and exactly ab additional nodes, b to be determined later. The arcs of the partial order are as follows. There is an arc $(S_i, root)$ for $i = 1, 2, ..., a$ and there is an arc (v_i, S_j) if element v_i is in set S_j. The ab additional nodes are partitioned arbitrarily into a sets, called *groups*, each of size b. There is an arc from each of the b nodes in group i to node v_i, $i = 1, 2, ..., a$. These are all the arcs. Every node is at distance at most three from the root. Set $rows(root) := 1$, and $rows(v) := 0$ for all other v. (One can massage the proof, if desired, so that $rows(v) > 0$ for all v, by making $rows(root)$ suitably large.) Set $k := a/3$. See Figure 2.

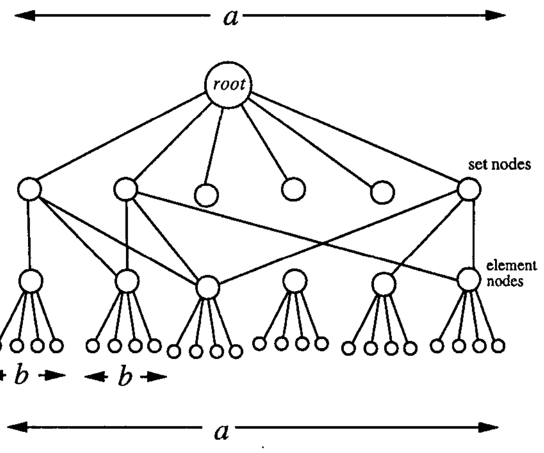

Figure 2. Picture to illustrate the proof of Theorem 1.

Define OPT to be the minimum value of $totalcost_R$ over all subsets R, containing $root$, of size at most $k + 1$. We need this lemma:

Lemma 2 *If the RX3C instance is a yes instance, then $OPT \leq (1 + 2a/3)$. Otherwise, $OPT \geq b$.*

Proof. If the RX3C instance is a *yes* instance, then there is an $I \subseteq \{1, 2, ..., a\}$ of size $a/3$ such that $\cup_{i \in I} S_i = U$. Define $R := \{root\} \cup \{S_i | i \in I\}$. Clearly $|R| = k + 1$. Each node v other than $root$ in R has $rows(v) = 0$. The set nodes $v \notin R$ have $cost_R(v) = 1$. The key point is that since I defines a cover of U, every nonroot, nonset node v has $cost_R(v) = 0$. Since there are $2a/3$ set nodes of cost 1, and every other node (except $root$) has cost 0, $totalcost_R = 1 + 2a/3$.

Now suppose the RX3C instance is a *no* instance. Denote the set $\{S_1, S_2, ..., S_a\}$ by $\mathcal{S}$. Let R^* denote a subset achieving minimum cost. It is not hard to see that without loss of generality, $R^* - \{root\} \subseteq \mathcal{S}$. The reason is that instead of choosing a nonroot, nonset node v, it is better to choose its parent. Thus $R' := R^* \cap \mathcal{S}$ has size at most $a/3$. Now, because the RX3C instance is a *no* instance, there is an element node u such that none of u's b children has an ancestor in $R^* - \{root\}$. It follows that $totalcost_{R^*} \geq b$. ■

The performance ratio of algorithm A_ϵ is given to be at most $n^{1-\epsilon}$ for all sufficiently large n. Given $\epsilon > 0$, choose a positive integer c large enough that $\epsilon \geq 3/(c + 2)$. Set $b := a^c$. The total number of nodes in the partial order above is $n = 1 + 2a + a^{c+1}$. The reduction above is polynomial time, since n is polynomial in a.

Now the ratio between the values of OPT above on *no* and *yes* instances is at least

$$\frac{b}{1+2a/3} = \frac{a^c}{1+2a/3} = \frac{3a^c}{3+2a} \geq \frac{3a^c}{3a} = a^{c-1}$$

$$> n^{\frac{c-1}{c+2}} = n^{1-\frac{3}{c+2}} \geq n^{1-\epsilon}.$$

Now we apply a standard argument. If n is large enough, and if the RX3C instance is *yes*, then algorithm A_ϵ produces a solution of cost at most $(1+2a/3)n^{1-\epsilon} < b$. Otherwise (if n is large enough), it clearly produces a solution of cost at least b. So one infers the solution to the RX3C instance: it's *yes* if and only if the cost of A_ϵ's solution is less than b. Thus A_ϵ solves RX3C in polynomial time for instances in which a is large enough, and hence can be modified to solve all instances of RX3C in polynomial time. ■

3 Extensions

In this section, we strengthen somewhat the hardness result of Theorem 1. Define the *depth* of a rooted dag to be the length of the longest leaf→root path.

3.1 Bounded Depth

Notice that in the dags constructed in the proof of Theorem 1, the depth is 3. Hence the negative result of Theorem 1 applies even to the special case of rooted dags of depth at most 3. Specifically, if P≠NP, then for every $\epsilon > 0$, no polynomial-time algorithm which finds feasible solutions in dags *of depth at most 3* has performance ratio at most $n^{1-\epsilon}$ for all sufficiently large n.

3.2 Bounded Degree and Logarithmic Depth Simultaneously

The proof can easily be modified so as to construct dags in which every indegree and every outdegree are at most 4, and simultaneously the depth is $O(\log n)$. To do so, start by replacing the group of b nodes which hangs off from each of the a element nodes by a binary tree with b nodes and depth $\lfloor \lg b \rfloor$. As before, each of these nodes v has $rows(v) = 0$. Then, between the root and the a set nodes, insert at most a dummy nodes, so that the structure, from the root down to the a set nodes, is a binary tree with a leaves (the set nodes) and depth $\lceil \lg a \rceil$. Each of the new dummy nodes v has $rows(v) = 1$. As before, $k = 1 + a/3$. At this point, the depth is $O(\log n)$, where the hidden constant depends on ϵ (since b and hence n depend on ϵ). Every node has indegree at most 3. The only nodes of outdegree possibly exceeding 4 are the a element nodes. However, because every element is in at most four sets, each element node has outdegree at most 4. Without loss of generality, an algorithm will choose none of the dummy nodes between the root and the set nodes, since each of these nodes v has $rows(v) = 1$, as does the root, and so nothing would be gained by choosing any of them. As before, it follows that the algorithm will choose the root and exactly $a/3$ of the set nodes. Hence the new reduction is correct. By the same reasoning as above, if P≠NP, then for every $\epsilon > 0$, there is a $\gamma > 0$ such that no polynomial-time algorithm which finds feasible solutions in dags of maximum indegree and outdegree at most 4 and simultaneously of depth at most $\gamma \lg n$ has performance ratio at most $n^{1-\epsilon}$ for all sufficiently large n. (One can improve the dependence of γ on ϵ by increasing the degree bound.)

That the problem remains hard to approximate even in this case is particularly interesting in light of the fact (see Section 5) that the problem might be solvable exactly in polynomial time when the dag represents a hypercube partial order. An N-node hypercube dag has maximum indegree and outdegree $\lg N$ and simultaneously depth $\lg N$. Though the problem is hard to approximate (if P≠NP) in digraphs of maximum indegree, maximum outdegree, and depth simultaneously $O(\log N)$, we can say nothing if the arcs are exactly those of the hypercube: the problem may be solvable exactly in polynomial time.

3.3 Generosity Doesn't Help

What if we give the algorithm a "break" by allowing it to materialize more than k, say, αk, nonroot views, where α is a fixed positive integer, and we compare the total cost it incurs to the best possible total cost when only k nonroot views may be materialized? Maybe one can do much better. Unfortunately, that is not the case. We can show that if such a polytime algorithm exists, then every language in NP can be solved very quickly—in time $n^{O(\log \log n)}$—and it is widely believed that NP-Complete languages, in particular, cannot be solved so quickly. Specifically:

Theorem 3 *Suppose that for some $\alpha, \epsilon > 0$, there is a polynomial-time algorithm $A_{\alpha,\epsilon}$ that takes an arbitrary positive integer k and an arbitrary n-node, weighted, rooted dag as input, and finds a subset of at most αk nonroot nodes to materialize, such that the total cost of a query with those at-most-$1 + \alpha k$ nodes materialized is, when n is sufficiently large, at most $n^{1-\epsilon}$ times the minimum possible total cost when a set of $1+k$ nodes is materialized. Then every language in NP can be decided in time $n^{O(\log \log n)}$.*

To prove this theorem, we need to define SET COVER: Given a universe $\{v_1, v_2, ..., v_a\}$ and m arbitrary subsets $S_1, S_2, ..., S_m$ of U, find a smallest subset $I \subseteq \{1, 2, ..., m\}$ such that $\cup_{i \in I} S_i = U$. (That is, find a smallest collection of sets whose union is the universe.) Clearly SET COVER is a generalization of EXACT COVER BY 3-SETS.

SET COVER is well-known to be NP-Complete [2]. We need a result which follows from a theorem of Feige [1], which is itself an improvement of a theorem of Lund and Yannakakis [8]:

Theorem 4 *[1] If for any α there is a polynomial-time α-approximation algorithm for* SET COVER, *then every language in NP can be decided by an algorithm running in time $n^{O(\log \log n)}$.*

Now Theorem 3 follows trivially from Theorem 4 and Lemma 5:

Lemma 5 *Suppose that for some $\alpha, \epsilon > 0$, there is a polynomial-time algorithm $A_{\alpha,\epsilon}$ that takes an arbitrary positive integer k and an arbitrary n-node, weighted, rooted dag as input, and finds a subset of at most αk nonroot nodes to materialize, such that the total cost of a query with those at-most-$1 + \alpha k$ nodes materialized is, when n is sufficiently large, at most $n^{1-\epsilon}$ times the minimum possible total cost when a set of $1 + k$ nodes is materialized. Then there is a polynomial-time α-approximation algorithm for* SET COVER.

Proof. Choose suitable $\alpha, \epsilon > 0$ and $A_{\alpha,\epsilon}$. We may assume n is sufficiently large.

We use the obvious generalization of the reduction in the proof of Theorem 1 to a reduction from SET COVER to VIEW SELECTION, instead of from RESTRICTED EXACT COVER BY 3-SETS to VIEW SELECTION. The reduction is almost exactly the same. Given a collection of m subsets of universe $\{v_1, v_2, ..., v_a\}$, we have m set nodes, and a element nodes as before, with element node v adjacent to set node S if and only if $v \in S$. Each element node is a parent of b new nodes, where now

$$b := [2(m+1)(1+m+a)]^{\lceil 1/\epsilon \rceil}.$$

As before, $rows(root) = 1$ and every nonroot node v has $rows(v) = 0$. The total number n of nodes is $1 + m + a + ab$. Since ϵ is constant, the reduction is polynomial time.

Let k be any positive integer. Consider any subset R, containing the root, of at most $1 + k$ nodes in the dag. It is easy to see (by "sliding a solution upward") that without loss of generality, $R - \{root\}$ consists only of set nodes.

Now suppose that the (unknown) optimal solution to the SET COVER instance consists of r sets. Set $k = r$ and choose R containing the root so that $R - \{root\}$ is a smallest collection of sets whose union is U. Now $totalcost_R \leq m + 1$, since every nonroot node v which is not a set node has $cost_R(v) = 0$.

Now let R' be any set consisting of the root and any other at-most-αr nodes in the dag. We may assume that R' consists only of set nodes. Now there are two cases. Either those αr set nodes "cover" all a element nodes or they do not. In the latter case, $totalcost_{R'} \geq b$. A simple calculation proves that $b > n^{1-\epsilon}(m+1)$; indeed, that's how b was chosen. Since $totalcost_R \leq m + 1$, $A_{\alpha,\epsilon}$ must find a set R', $|R'| \leq 1 + \alpha r$, with $totalcost_{R'} \leq n^{1-\epsilon}(m+1) < b$. This means that the second case above cannot occur for this R': $R' - \{root\}$ must be a set of at most αr set nodes that covers all a element nodes. Hence $R' - \{root\}$ is a feasible solution to the SET COVER instance of size at most α times optimal. This is almost an α-approximation algorithm for SET COVER.

The only gap in the reasoning is that r is unknown. Assuming $A_{\alpha,\epsilon}$ exists, define $B_{\alpha,\epsilon}$ by running $A_{\alpha,\epsilon}$ for $k = 1, 2, ..., m$ (and converting the nonroot nodes of the output, if necessary, to set nodes), its output now unspecified. When k is the optimal solution r to the SET COVER instance, we get a set R' such that $R' - \{root\}$ is a collection of at most αr sets the union of which is U. Hence we can specify that $B_{\alpha,\epsilon}$ output the smallest collection of set nodes it found, over $k = 1, 2, ..., m$, among those collections which cover all element nodes. Though r will be unknown, the smallest collection will have size at most αr. $B_{\alpha,\epsilon}$ is a polynomial algorithm and thus $B_{\alpha,\epsilon}$ is a (polynomial-time) α-approximation algorithm for SET COVER. ■

4 Analysis of the Harinarayan et al. Algorithm

Iterative, natural and greedy, the algorithm of [6] is this: Start with $R = \{root\}$, and then for k iterations, choose a node $w \notin R$ such that $totalcost_{R\cup\{w\}}$ is minimized, and then add w to R. In other words, in each iteration, choose the one additional node the realization of which would most decrease the total cost, and realize it. (The algorithm is described differently in [6].)

Now we study the performance ratio.

Theorem 6 *For any $n \geq 20$ which is a multiple of 4, the performance ratio of the greedy algorithm is at least $n/12$.*

Proof. Consider the partial order described in Figure 3, in which each of the four paths of unnamed vertices has $N = (n-4)/4$ nodes v, each with $rows(v) = 1$, a simple modification of Figure 9 in [6]. The root node has $rows(root) := M := 36N$; $rows(a) = rows(c) = 2$, $rows(b) = 1$, and $k = 2$.

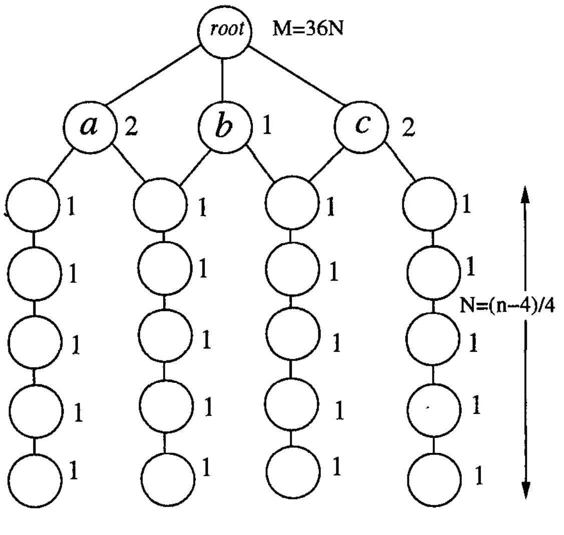

Figure 3. Picture to illustrate the proof of Theorem 6.

What will the greedy algorithm do? Initially $R = \{root\}$. In the first iteration, among all the possible nodes one could add to R, the only viable contenders are a, b, c. Since each has $2N$ proper descendants each with $rows = 1$, and b is cheapest, the algorithm chooses to add b to R. Now the key point is that whatever output set R results after the second iteration, all the N nodes v in at least one path have $cost_R(v) = M$. It follows that $totalcost_R \geq NM$.

However, the optimal solution is to take $R = R^* = \{root, a, c\}$. Now $cost_{R^*}(root) = cost_{R^*}(b) = M$, and every other node v has $cost_{R^*}(v) = 2$, so that $totalcost_{R^*} = 2M + 2(4N + 2)$.

It follows that the performance ratio is at least

$$\frac{NM}{2M + 2(4N+2)} = \frac{36N^2}{80N+4} \geq \frac{N+1}{3} = \frac{n}{12},$$

because $n \geq 20$. ■

5 Special Cases and Heuristics

The results of the previous sections, in particular Theorems 1 and 3, are negative. There is essentially no hope in looking for an approximation algorithm for the view-selection problem for general partial orders. However, none of these results appear to carry over to special cases such as hypercubes or product graphs. The significance of hypercubes and product graphs is explicitly mentioned in [6]: they model the two most commonly used operations in data warehouse applications, namely "drilling down" to see data in finer detail, and "rolling up" to see data in less detail. There is no known lower bound or even hardness result for graphs with such special structure. For example, VIEW SELECTION in hypercubes could be in P. Also, there is no known approximation algorithm with nontrivial performance guarantee on the hypercube.

From a practical point of view, again in view of the negative general results, one should experiment in practice with heuristics beyond the greedy algorithm of [6] and compare their performance. For example, one could start with all views materialized and repeatedly remove a view so that the increase in total cost is minimized, until only $k + 1$ views remain. Alternatively, one could start by materializing $k + 1$ arbitrary views (including the root) and then make local improvements by swapping in one view and swapping out another, either provided that the total cost decreases with each swap, or in an annealing fashion.

Acknowledgments

We thank Munir Cochinwala for pointing us to [6], and we thank Leo Mark and Ed Omiecinski for helpful discussions.

References

[1] U. Feige, "A Threshold of ln n for Approximating Set Cover," *Proceedings of the 28th ACM Symposium on the Theory of Computing*, 1996, Philadelphia, 314-318.

[2] M. Garey and D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman and Company, New York, 1979.

[3] H. Gupta, "Selection of Views to Materialize in a Data Warehouse," *Proceedings of the International Conference on Database Theory*, Delphi, Greece, January 1997.

[4] H. Gupta, V. Harinarayan, A. Rajaraman, J. Ullman, "Index Selection for OLAP," *Proceedings of the International Conference on Data Engineering*, Burmingham, U.K., April 1997.

[5] "Selection of Views to Materialize Under a Maintenance Cost Constraint," *Proceedings of the International Conference on Database Theory*, Jerusalem, Israel, January 1999.

[6] V. Harinarayan, A. Rajaraman, and J. Ullman, "Implementing Data Cubes Efficiently," *Proceedings SIGMOD* 1996, 205-216.

[7] D. Hochbaum, editor, "Approximation Algorithms for NP-Hard Problems," PWS Publishing Company, 1995.

[8] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," *Journal of the Association for Computing Machinery* 41 (1994), 960-981.

[9] J. Ullman, "Efficient Implementation of Data Cubes Via Materialized Views," a survey of the field for the 1996 *KDD Conference*.