Relational Data Markets

544 Class Project – Fall 2007

Ben Birnbaum
birnbaum@cs.washington.edu

Alex Jaffe
ajaffe@cs.washington.edu

1. INTRODUCTION

It is hard to overestimate the value of information as a resource. In the last several years, the number and variety of structured information sources has soared, and yet, centralized access to these resources remains limited. The integration of these (possibly heterogeneous) data sources is a major topic of database research. In this paper, we propose a paradigm in which the sharing of information is incentivized by monetary gain. Such systems have been proposed before, notably in Mariposa [10]. However, these systems typically place value on the resources used to access the data and not on the data itself. We propose Relational Data Markets (RDMs), distributed database systems in which providers can set arbitrary costs for their data, and clients can collect data from multiple overlapping providers of their choice, in order to fill their needs at minimal cost. We consider the theoretical models underlying such choices by clients, leaving the design of a full-scale system for future work.

To make the setting concrete, consider the following scenario. A large retail corporation, Huge-Mart, wishes to generate customized e-mails to their mailing list subscribers. These e-mails should feature products that the customer is likely to be interested in; they should contain discounts that are tailored to the customer's spending patterns and income; and they should be customized in tone in order to appeal to the customer's demographic. Huge-Mart is likely to have some information about its customers already, perhaps acquired from its customer reward program. However, this might not be enough for the directed marketing needed to maximize profits. Therefore, Huge-Mart might turn to outside market research firms that aggregate data about customer demographics, habits, and opinions.

Instead of relying solely on market research firms, Huge-Mart could instead learn about its customers using an RDM. It would decide on a set of features that will be used to generate its e-mail, such as age, income, credit score, and marital status. It would also have a list of customer names and a value to acquiring this information about each of these customers based on information already known. The set of attributes defines a table schema, which would then act as a query through the RDM to find data providers who offer some subset of these columns, for some subset of the row ids (names) that Huge-Mart is interested in.

In response its query, Huge-Mart would be given a listing of providers, along with the rows and columns they have available and the cost they are willing to sell a row for. These providers would run the gamut from other corporations selling their collected data, to social networking sites, to wrappers on top of GPS units and cell phones. No single provider would have all of the rows or columns, but in aggregate, Huge-Mart might be able to purchase them all. It would like to do so in such a way that maximizes the value of the rows completely purchased while minimizing the total cost paid to the various providers. Having decided on a set of rows it would like to purchase from each provider, Huge-Mart authorizes the purchases, and is then able to query the specified data as if it were in a standard distributed database.

Applications of RDMs extend beyond marketing. For example, consider data collection for large-scale scientific research, such as the NSF NEPTUNE project, which consists of an enormous network of sea-floor data labs used for oceanographic data collection and routing. This network is being built to monitor the ocean's processes on a large scale with overall goals such as better understanding global warming. We propose that research projects like NEPTUNE could consist of a large number of distributed data collection sources, rather than one central project. If this were the case, then it could more efficiently use data already being collected by private parties, such as temperature measurements from fishing companies, ocean-floor maps from the military, or salinity measurements by independent scientists. Of course not of all of these parties would be willing to share their information for free, and this is where RDMs would play a role.

Motivated by these scenarios, we examine several versions of an optimization problem that might arise in an automated system for aggregating data in an RDM. In this paper, we study the computational hardness of these problems, and develop algorithms for those problems that are tractable. We go on to design heuristic algorithms for some of our intractable problems, and run implementations of these algorithms on data generated according to two different models for random RDMs. Finally, we study the performance of these algorithms statistically, both in terms of optimality, and running time.

2. RELATED WORK

We know of no pre-existing database research in which the value and cost of data itself, rather than the resources to access it, are considered. Therefore, much of this section surveys the fundamental techniques in the literature that would be essential to the construction of a a functioning RDM. We also discuss other database systems in which monetization is considered, though usually for the purposes of distributing computational rather than information resources.

The problem of providing the user with a single, consistent view of data that in reality comes from multiple sources is known as data integration [7]. In commercial settings, this is sometimes known as Enterprise Information Integration [4]. One approach to data integration is data warehousing, the practice of collecting data from many sources, remapping its schemas for consistency, and storing it in a single database, often with automated updates. This practice has dropped in popularity, in comparison to more versatile systems in which data remains on its source machines, but is nonetheless accessed consistently by the user in real-time, as in [11]. This model is more appropriate for our proposal, since storing the data of multiple providers in a single repository would require a central arbiter whom the providers trust not to share or even examine their data, adding an extra layer of complexity.

Many papers address the problem of distributed query optimization specifically. Here the focus is less on how to design a large-scale system. Instead authors focus on the algorithmic question of how to plan queries, with the knowledge that the tables are spread across multiple systems over a network. Such optimization techniques should be built into a system implementing our proposal, since at the least, traditional distributed queries must be run in order to identify the set of relevant tuples for our algorithms to consider buying.

The standard approach to distributed query optimization considers network transfer to be a bottleneck in the speed of query processing [1]. In contrast, more recent work [2] has addressed how query optimization should be performed over high-speed networks, such as broadband internet connections, where a more fine-grained tradeoff between local computation and network transfer speed must be considered.

Huebsch et al. [5] propose a distributed query optimization technique that is well-suited for a very large number of databases, not just a large amount of data over a few databases. The authors of [9] propose a query planning process that is optimized for the specificities of web-based APIs. An RDM could benefit from the encapsulation and modularity of such a model, especially since many of the data sources may be based off of pre-existing web-services.

The system bearing perhaps the most resemblance to an RDM is the economic bidding system of the distributed database Mariposa [10]. Mariposa, like our proposal, is meant to support distributed databases in which not all of the parties involved have mutual goals. They propose an economic model for a) incentivizing the distribution of data from those who have it to those who want it, and b) determining which of those who want it actually may access it. Note that a) is is a goal of our model as well, but b) is not relevant to RDMs. This is because Mariposa operates under a limited-resources assumption, and charges clients based on the amount of resources it expects to expend on their query, rather than an externally defined value of the data itself. This assumption leads to a database being able to service only a fraction of its clients, and it hence uses an auction system to determine who is given the privilege. Of course, computing resources are always limited, but we are more interested in heterogenous data utilities, which can be more easily studied by assuming unbounded computational resources.

Mariposa's protocols can be prohibitively time-consuming, due to the high cost of performing so many auctions in real-time. Fortunately, our model is simpler in that providers set fixed prices for their data. This is a reasonable limitation, because RDMs are intended to support a large number of data providers, less so than a large number of data customers. Note that an RDM would be interesting even with only a single customer of the data.

Jain and Kanna [6] take a perspective similar to Mariposa's, considering the cost of data in terms of the processing time to retrieve it. Rather than designing a system for such data services, the authors perform a game-theoretic analysis of various pricing schemes for data. This kind of analysis would be interesting to perform on our models, as price-setting is essentially the dual of the optimization problems we consider in this paper. However, the analysis of [6] is not directly relevant to us, since they assume a uniform utility for data, and are interested primarily in the pricing of a single provider's data.

For the purposes of our analysis, we have assumed that all providers share a consistent schema, and have correct data. However, in a real system, it is likely that the providers would in fact have very different formats and structures for the data. Integrating such databases is a difficult problem, but luckily it has been studied extensively, and the introduction of pricing should not change the efficacy of pre-existing solutions. Cohera [11], a substantial content integration system built on top of Mariposa, was designed specifically for a task such as ours. It is intended for the e-business setting, in which many heterogeneous databases with *different* owners must be made to function together. This multiple-owners problem is one of the key issues in designing an RDM. It seems that many integration and query processing needs of an RDM would be sufficiently handled by Cohera, with monetary exchange, query, and optimization systems built on top of their system.

3. MODEL

We model the data aggregation problem in an RDM as follows. A user is interested in purchasing data from a canonical table D, with m rows and n columns. The user can purchase data from this table from a set of ℓ *providers*, each of which sells some some subset of rows and columns of D. In particular, for $1 \leq k \leq \ell$, provider k has a table D_k, which consists of a subset $R_k \subseteq [m]$ of the rows and a subset $C_k \subseteq [n]$ of the columns. Each provider k also has a cost p_k for each row.

The first variation of this problem that we have studied is

called the FullRowMaxUtility model. In this model the user has a budget B and must purchase rows from providers without spending more than B. In other words, the user must select a subset of rows $S_k \subseteq R_k$ from each provider k such that $\sum_k |S_k| p_k \leq B$. The user has a utility u_i for each row i in D, and her goal is to maximize the sum of the utilities of the rows that she has purchased completely. More precisely, given that S_k is the subset of rows purchased from provider k, we can define a predicate $FULL(i)$ to be true if and only if $\bigcup_{k:i \in S_k} C_k = [n]$. The utility of the user is $\sum_{i:FULL(i)} p_i$. Such a model is motivated by scenarios in which a user seeking data would only find a use for the data when collecting feature information of products one is considering purchasing.

The second variation is called the FullRowMinCost model. This is the corresponding minimization to FullRowMaxUtility. Hence the formulation and optimization function are identical, but the problem is to achieve (or beat) a fixed utility U, while minimizing the budget expended to do so. There are subtle differences in framing the problem in this way: it is perhaps slightly less natural for a real-world scenario, but matches up more closely with the algorithmic mainstream, and moreover may permit an approximation algorithm, unlike the maximization version.

The final variation we have studied is called the PartialRowMaxUtility model. In this model, we return to maximizing the utility achieved with a fixed budget B. However, we now assign a value u_{ij} to each row-column pair. The total utility the user gets is the sum of the u_{ij}'s for each row-column pair purchased.

4. THEORETICAL RESULTS

In this section, we prove some theoretical results on the models introduced in the last section. Our first result states that not only is the FullRowMaxUtility model NP-hard, but it is also hard to approximate to within any factor. The proof of this result, which is given in the appendix, is based on a reduction from SetCover.

Theorem 1. *It is NP-hard to approximate* FullRowMaxUtility *to any factor f.*

Another hardness result holds for FullRowMinCost. Although this is not as strong as the result for FullRowMaxUtility, it does imply that no efficient algorithm exists that guarantees a worst-case performance within a constant factor of optimal for this problem. The proof, again given in the appendix, is based on an approximation-preserving reduction from SetCover.

Theorem 2. *It is NP-hard to approximate* FullRowMinCost *to within a factor of $O(\log n)$.*

On a positive note, we show with the following theorem that PartialRowMaxUtility seems to be a more tractable problem.

Theorem 3. *If each provider charges the same cost p for a row, then there is a greedy algorithm for* PartialRowMaxUtility *that achieves an approximation ratio of $1 - 1/e \approx 0.63$.*

To prove this this theorem, we use a well-known result on maximizing monotone submodular functions, which are defined as follows.

Definition 1. *Let $f : 2^X \rightarrow \mathcal{R}$ be a real-valued function defined on subsets of some finite ground set X. The function f is* monotone *if for all $S \subseteq T \subseteq X$, we have $f(S) \leq f(T)$. The function f is* submodular *if for all $S \subseteq T \subseteq X$ and for all $x \in X$ such that $x \notin T$, we have $f(S \cup \{x\}) - f(S) \geq f(T \cup \{x\}) - f(T)$.*

Intuitively, a submodular function is one that satisfies the law of diminishing returns: the marginal value of an item decreases as the set gets larger. Given a submodular function f and an integer $r \leq |X|$, a natural optimization problem is to find the r elements $S \subseteq X$ that maximize $f(S)$. We call this the MaxSubmodular problem. Nemhauser, Wolsey, and Fisher prove the following well-known theorem.

Theorem 4 ([8]). *There is an algorithm for* MaxSubmodular *that achieves an approximation ratio of $1 - 1/e$.*

In fact, the algorithm of Theorem 4 is the natural greedy algorithm: iteratively construct a set S, each time adding the element x that maximizes the marginal utility $f(S \cup \{x\}) - f(S)$. The proof of Theorem 3, given in the appendix, gives a reduction from this restricted version of PartialRowMaxUtility to MaxSubmodular. Hence, this implies that the algorithm that achieves the approximation ratio of Theorem 3 is the greedy algorithm that purchases row-column pairs to maximize the marginal utility at each step.

If the cost for each provider is different, but the number of providers is a constant, we show in the following theorem that there is a polynomial-time algorithm for PartialRowMaxUtility.

Theorem 5. *If ℓ is $O(1)$, then there is a polynomial time algorithm for* PartialRowMaxUtility.

The algorithm that solves this version of PartialRowMaxUtility is a dynamic programming algorithm similar in spirt to the classic pseudo-polynomial time algorithm for Knapsack. The proof of Theorem 5, again given in the appendix, first reduces PartialRowMaxUtility to a problem we call the KChoiceKnapsack problem and then gives a polynomial-time algorithm for KChoiceKnapsack when the number of costs is constant.

5. HEURISTIC ALGORITHMS

In this section, we investigate several heuristics for our models. We have shown that there are no polynomial-time algorithms with reasonable approximation rations for the FullRowMaxUtility and FullRowMinCost models. Therefore, it is important to find algorithms that perform well in

practice. Furthermore, even though PARTIALROWMAXUTILITY seems to be more tractable, it is still interesting to investigate the performance of algorithms for this problem on typical input cases.

5.1 Algorithms

A natural first heuristic to consider is one that greedily makes purchases to maximize the new number of columns obtained over cost. (We define a purchase to be an ordered pair (i, k), where i is a row and k is a provider.) The hope is that this algorithm will be likely to complete a large number of rows for a small price, thereby achieving a large utility. We call this algorithm 1-STAGEGREEDY, which is defined more precisely as follows. While there is still budget left, make the purchase (i, k) that maximizes the ratio $\Delta c_{ik}/p_k$, where Δc_{ik} is the number of new columns in row i obtained by the algorithm from the purchase (i, k), and p_k is the price per row charged by provider k. (Note that the value of $\Delta c_{i,k}$ depends on the columns already owned by the algorithm.)

Despite first impressions, choosing the purchase that gets the most new columns is not necessarily an effective way to complete rows. It is not hard to imagine a scenario in which there are a large number of rows that need just one more purchase to be completed, but 1-STAGEGREEDY instead wastes its money on uncompleted rows in which a large number of new columns can be bought. Motivated by this concern, we present the following algorithm, 2-STAGEGREEDY.

2-STAGEGREEDY uses a sub-procedure called GREEDYCOMPLETEROW, which takes a row i and a budget B as parameters, and attempts to find a set of purchases that complete row i for a cost no greater than B. To complete row i, this sub-procedure uses the same greedy heuristic as 1-STAGEGREEDY: iteratively choose the purchase (i, k) that is under the current budget and that maximizes $\Delta c_{ik}/p_k$.

2-STAGEGREEDY works as follows. While there is still budget $b > 0$ left, run GREEDYCOMPLETEROW$(i,\ b)$ for each uncompleted row i, to obtain a candidate set of purchases that will complete row i. For each row i that GREEDYCOMPLETEROW can complete, let r_i be the cost of the set of purchases returned by GREEDYCOMPLETEROW. Now choose the row i that maximizes u_i/r_i (where u_i is the utility of row i). Make the purchases given by GREEDYCOMPLETEROW, and repeat.

Note that although we motivated 1-STAGEGREEDY and 2-STAGEGREEDY for the FULLROWMAXUTILITY model, the versions of these algorithms that ignore budget and iterate until the utility goal is met are equally well-motivated for the FULLROWMINCOST model. (We refer to both versions of these algorithms by the same name and use context to disambiguate.)

For the PARTIALROWMAXUTILITY model, we have already shown in Theorem 3 that the simple greedy heuristic that chooses the purchase with the largest marginal utility is always within 63% of optimal when the price for each provider is the same. This algorithm has a natural extension when the price of a purchase varies from provider to provider, which we call GREEDY: choose the purchase that maximizes the ratio of the marginal utility to cost. Although we do not have any theoretical results for GREEDY in the general PARTIALROWMAXUTILITY model, it is still possible to test its performance experimentally.

5.2 Experiments

To test the performance of these heuristic algorithms, we implemented them in Python and ran experiments. Since our motivation for studying these problems was based on systems that have not yet been implemented, we do not have real-world data on which to test our algorithms. Instead we tested the algorithms on synthetic data, generated by the following two models:

- UNIFORM takes five parameters: p_r, the row probability; p_c, the column probability; ℓ, the number of providers; m, the number of rows; and n, the number of columns. The model is generated as follows. First, the canonical $m \times n$ table is created. Next, utilities are assigned to rows (in the FULLROWMAXUTILITY and FULLROWMINCOST models) or row-column pairs (in the PARTIALROWMAXUTILITY model). Utilities are chosen independently and uniformly at random between 0 and 1. The cost-per-row for each provider is chosen similarly. Finally, each provider is assigned row i independently with probability p_r, and column j with probability p_c.

- POWERLAW takes four parameters: α, the parameter for the power law; ℓ, the number of providers; m, the number of rows; and n, the number of columns. This model is generated in the same way as UNIFORM, except that for each provider, the row and column probabilities are selected independently at random from the following power law distribution f defined on $[0, 1]$:

$$f(\alpha; x) = \frac{1}{\alpha(x + c_\alpha)^2} \ ,$$

  where $c_\alpha = \frac{1}{2}(-1 + \sqrt{1 + 4/\alpha})$ is defined so that f normalizes on $[0, 1]$. This model is motivated by the fact that Zipf-like power laws seem to be found often in real data. This ensures that a few providers will have many more rows and columns than average.

Testing the optimality of these algorithms is a challenge since even computing the value of the optimal solution is intractable for all three models. Therefore, we define several natural random algorithms as a baseline, and compare our heuristics to these random algorithms. For the full-row models, the random algorithms we implemented are defined as follows.

- 1-STAGERANDOM– While there is budget left (or the utility goal is not met), make a purchase uniformly at random from all feasible purchases that improve our optimum.

- 2-STAGERANDOM– While there is budget left (or the utility goal is not met), pick a random row, and randomly pick purchases for that row until the row is completed, buying them if a feasible solution is found.

Our motivation for 2-StageRandom is that it provides a more fair benchmark for 2-StageGreedy than 1-StageRandom, since 1-StageRandom does not to give any preference for completing rows.

We also use 1-StageRandom as a benchmark against which we can compare the Greedy algorithm in the PartialRowMaxUtility model.

Effect of Budget and Utility Goal

Our first experiment was designed to measure the solution quality of our algorithms as a function of the budget (for the maximization problems), or the utility goal (for the minimization problem) for both the Uniform and PowerLaw models. For the Uniform model, we set $p_c = p_r$ and chose values for p_r ℓ, m, and n that seemed to provide instances that were neither trivial nor impossible to solve. We used the same values of ℓ, m, and n for the PowerLaw model, and chose α such that the mean of $f(\alpha)$ was equal to p_r (so the row and column densities would have the same expected values.) For the maximization problems, we varied the budget from 6 to 400, and for the minimization problems, we varied the utility goal from 0.5 to 40. The data from these experiments, as well as the exact value of our parameters, are shown in Figure 1.

Figure 1(a) shows the solution quality (measured by utility) of 1-StageRandom, 1-StageGreedy, 2-StageRandom, and 2-StageGreedy, on the FullRowMaxUtility problem generated from the Uniform model. Note that 2-StageGreedy performs at least as well as the other algorithms for all budgets. For most budget values, the relative ordering of the other algorithms, from best to worst, is 1-StageGreedy, 2-StageRandom, and 1-StageRandom. Also, note that the utilities of all the algorithms converge to the same value as the budget gets large. This is because once the budget is large enough, all of the algorithms have enough money to obtain the maximum utility possible from a problem.

There are two somewhat puzzling features of this graph. First, for small budget values, 2-StageRandom, a seemingly mindless algorithm, outperforms the more sensible heuristic 1-StageGreedy. This might be explained by the fact that for small budgets, only a small number of rows can be afforded. Since 1-StageGreedy does not give priority to completing rows, it might spend all of its money on purchases that do not complete any row. This is in contrast to larger budgets, when the early investment by 1-StageGreedy on extra value-purchases can help complete more rows later.

The other puzzling feature of Figure 1(a) is the spike in profit the two greedy algorithms at budget 100. We have not yet found a satisfying explanation for this phenomenon. It is not likely to be a statistical error however, since it occurs in the PowerLaw model as well. (Figure 1(b)). One possibility is that there is a critical period in which the budget is large enough to make new expensive purchases that are a mistake but not large enough to overcome those mistakes. Whatever the cause, this phenomenon seems interesting and requires more careful investigation.

Figure 1(c) shows the solution quality (measured by cost) of 1-StageRandom, 1-StageGreedy, 2-StageRandom, and 2-StageGreedy on the FullRowMinCost problem generated from the Uniform model. Again, 2-StageGreedy performs the best, since it spends the least money of the four. The money spent by the algorithms flattens as the utility goal gets high. This is because in this range, the utility goals are impossible to attain, and each algorithm ends up making purchases until it cannot make any more. Thus, the money spent stops increasing past a given utility goal.

For small utility goals, 1-StageRandom is better than both 1-StageGreedy and 2-StageRandom. We believe that the reason that it is better than 1-StageGreedy is that for small utility goals, just one row needs to be completed. As seen earlier, 1-StageGreedy is slow to complete rows, and so performs poorly at a small scale. A reason why it might be better than 2-StageRandom is that since 2-StageRandom chooses a random row and then keeps making purchases until the row is complete, it can get stuck on a very costly row. By not committing itself to a single row at a time 1-StageRandom, hedges its bets better.

Figure 1(e) shows the solution quality (measured by utility) of Greedy and 1-StageRandom on the PartialRowMaxUtility problem from the Uniform model. Note that Greedy performs significantly better than 1-StageRandom, for small budget values, and the the two algorithms converge towards the same solution quality for large budgets. Again, this is because there is a point where the budgets become large enough that any algorithm can obtain all possible utility from the problem.

Figures 1(b), 1(d), and 1(f) show the performance of our algorithms for all three problem models under the PowerLaw model. The behavior of the algorithms is qualitatively the same for the PowerLaw model as for the Uniform model, which suggests that our conclusions about the algorithms are not artifacts of the particular random model we use.

Effect of Number of Providers

Our second experiment was designed to measure the solution quality of our algorithms as a function of the number of providers available. We tested our algorithms on both the Uniform and PowerLaw models, although we only show the results of the Uniform model here, since like the last experiment, the algorithms behaved similarly on both models. This time we fixed values of p_r, p_c, m, n, and B that seemed to provide interesting instances. We varied ℓ, the number of providers, from 2 to 500. The results of this experiment, as well as the exact value of our parameters, are shown in Figure 2. We note that again 2-StageGreedy significantly outperforms the other algorithms, but we omit a detailed discussion of these results because of space constraints.

Running Time of the Algorithms

Our third experiment was designed to measure the running time of our algorithms on both the Uniform and PowerLaw models. For comparison, all of the algorithms have asymptotic running time at most $O(m^2 \cdot (l+n)^2)$. We fixed values for p_r, p_c, α, n, and ℓ, and measured the average time for the algorithms to complete as a function of m, the

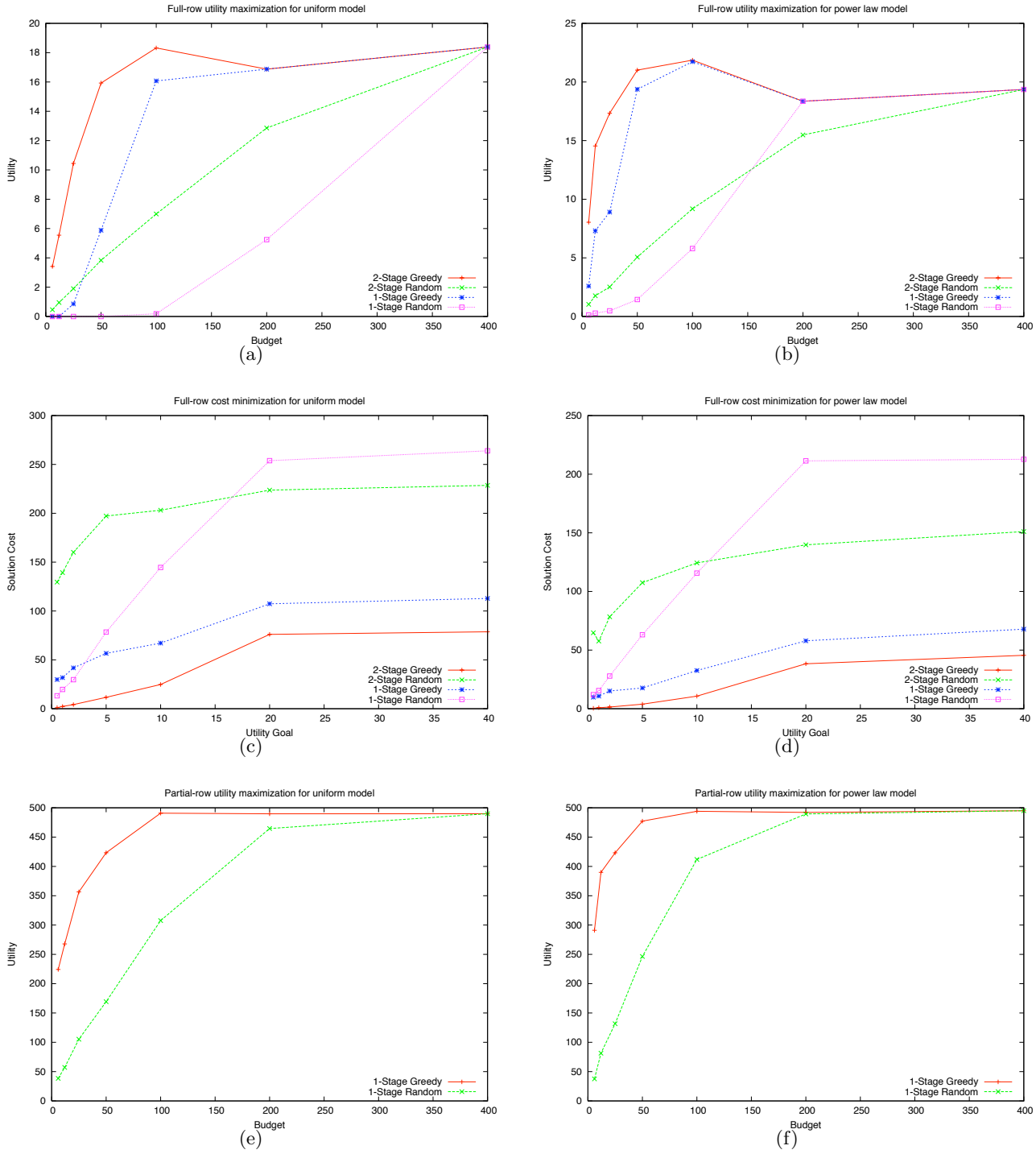

Figure 1: The solution quality of the algorithms as a function of budget (for the maximization problems) and utility goal (for the minimization problems). For the uniform model, the parameters used were $p_r = p_c = 0.2$, $m = 50$, $n = 20$, and $\ell = 100$. For the power-law model, the parameters used were $\alpha = 5.89$, $m = 50$, $n = 20$, and $\ell = 100$. Each graph shows the average results over ten randomly generated instances. Each deterministic algorithm was run once per instance, and each randomized algorithm was run twice per instance.

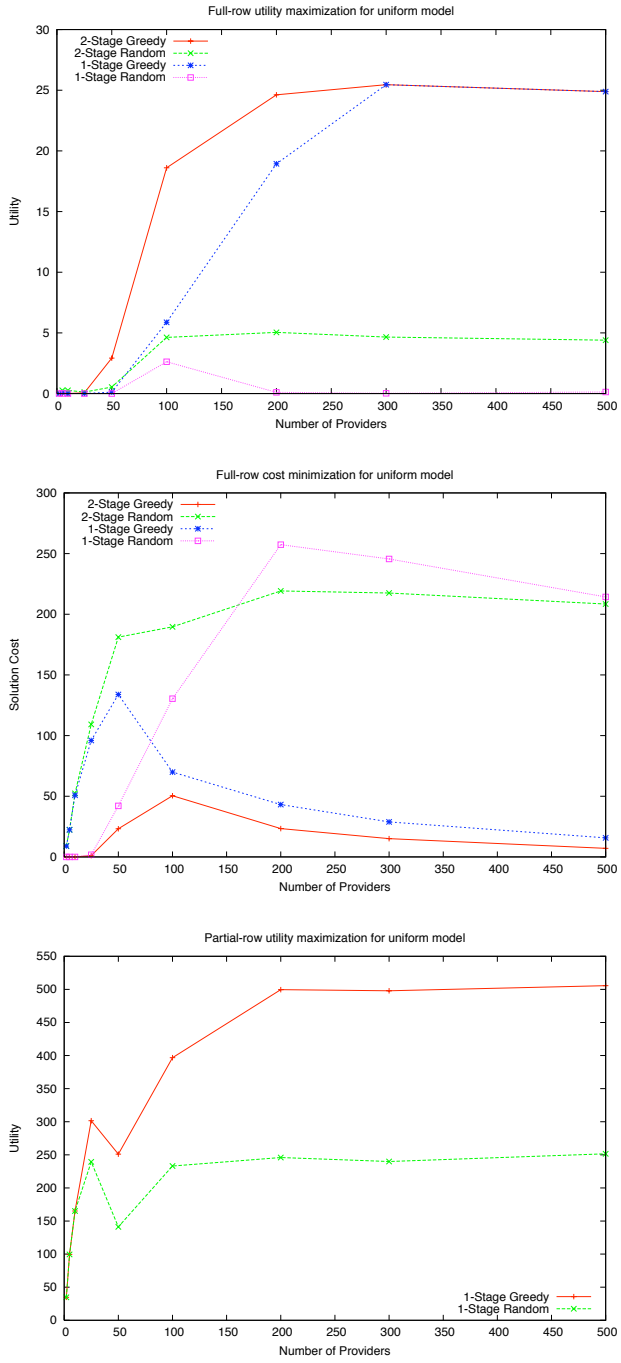

Figure 2: The solution quality of our algorithms as a function of the number of the number of providers for the uniform model. The parameters used were $p_r = p_c = 0.2$, $m = 50$, $n = 20$, $B = 75$ (for the maximization problems), and $U = 15$ (for the minimization problems). Each graph shows the average results over ten randomly generated instances (except for $\ell = 500$, which was run on less instances because of time constraints). Each deterministic algorithm was run once per instance, and each randomized algorithm was run twice per instance.

number of rows.To ensure that we were isolating the effect of m on the running time we set B large enough in the maximization problems to ensure that every row and column could be purchased, and set U large enough in the minimization problem to ensure that every row and column *would* be purchased. Figure 3 shows the result of this experiment on the FullRowMaxUtility model (as well as the specific parameter values). We also tested the algorithms on the FullRowMinCost and PartialRowMaxUtility models, but we omit the figures because they are very similar to the results for FullRowMaxUtility.

Note that for all values of m, the algorithms from fastest to slowest are 2-StageRandom, 2-StageGreedy, 1-StageRandom, and 1-StageGreedy. Although it may seem surprising that the two-stage algorithms are faster than the one-stage algorithms, the reason for this is actually fairly simple. In the one-stage algorithms, the next purchase to make is chosen from a large number of purchases (extending over many rows). This means that a large data structure containing the feasible purchases must be maintained throughout the algorithm. In the two-stage algorithms, on the other hand, the next purchase to make is always chosen from the current row being completed. This more local decision can be made much more efficiently.

The main conclusion that we can draw from this experiment is that there is little advantage to the one-stage algorithms, since 2-StageGreedy is more efficient, and produces higher quality solutions except for small budgets or few providers. Furthermore, greedy algorithms achieve much higher quality solutions than random algorithms, but are significantly slower. This presents a reasonable tradeoff, between 2-StageGreedy and 2-StageRandom .

6. CONCLUSION AND FUTURE WORK

In this report we have motivated the need for Relational Data Markets. We proposed a formal model for several optimization problems that are inherent in running queries in such systems. We showed that these problems were all NP-hard, and for the full-row models, they are even hard to approximate to within a constant factor. For the PartialRowMaxUtility version of the problem, we gave an approximation algorithm and a polynomial-time dynamic programming algorithm for two natural special cases. Beyond these theoretical results, we proposed a heuristic for the full-row models and ran experiments to show both that it produces relatively high-quality solutions and has a reasonable running time.

There are several interesting directions for future work. Regarding our theoretical results, it would be interesting to find an approximation algorithm for either the FullRowMinCost problem or the general case of the PartialRowMaxUtility problem. For our heuristic results, it would be useful to have a heuristic algorithm that has a performance as good as 2-StageGreedy but that scales better for large values of m.

A simplification that we have made is that the utility of a row is known before the data in the column is known. In practice, it seems more realistic that the utility of a row is only known after the actual data inside the column is known.

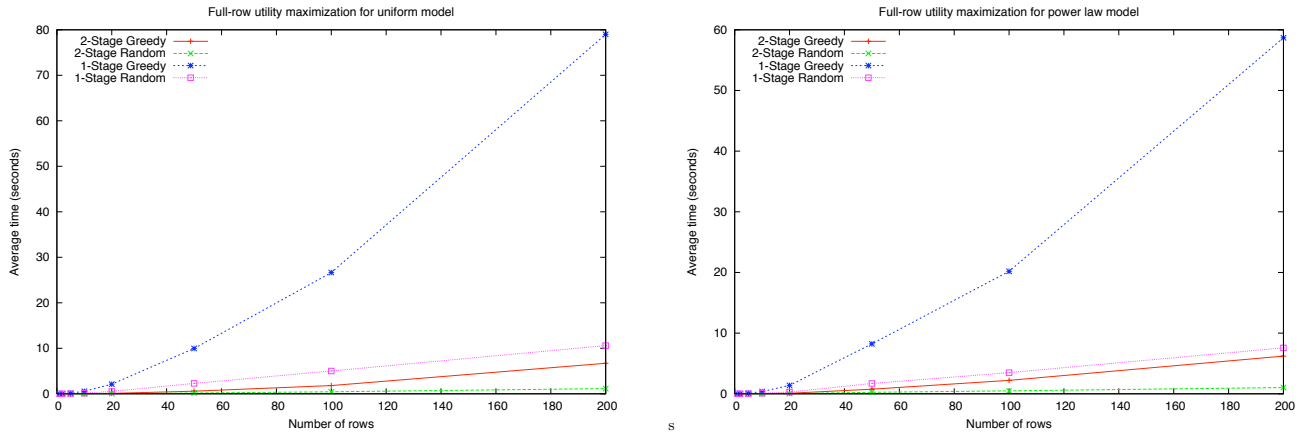

Figure 3: The running time of the algorithms, as a function of m, the number of rows. The parameters used were $p_r = p_c = 0.2$, $\alpha = 5.89$, $n = 20$, $\ell = 20$, and $B = 75$. The graphs show the average results over 10 instances. Each deterministic algorithm was run once per instance, and each randomized algorithm was run twice per instance.

We believe that there is a potential for interesting theoretical models for this. An algorithm in this model would have to pay to learn about the utility for a row, and would therefore have to find a balance between exploring for new valuable rows and completing rows it already knows are valuable.

Another interesting theoretical problem would be to examine RDMs from the point of view of the provider. We have assumed the prices are fixed, but in future work we would like to examine the game-theoretic decisions faced by a provider in choosing a pricing scheme for its data.

Of course, one of the most interesting directions for future work would be to actually implement an RDM. There are a number of practical concerns that would have to be addressed. For example, we have assumed that schemas are consistent between the different providers, which seems unlikely to be the case in practice. We have also assumed that every provider has gives trustworthy data, which is another assumption that would have to be removed in a real system.

Although there are a number of challenges facing an implementor of an RDM, we believe that by providing incentives to heterogeneous parties to provide useful data, RDMs have the potential to help create more efficient data aggregation in domains as diverse as direct marketing and oceanography.

7. REFERENCES

[1] P. M. G. Apers, A. R. Hevner, and S. B. Yao. Optimization algorithms for distributed queries. *IEEE Trans. Softw. Eng.*, 9(1):57–68, 1983.

[2] S. Banerjee, V. O. K. Li, and C. Wang. Distributed database systems in high-speed wide-area networks. *Selected Areas in Communications, IEEE Journal on*, 11(4):617–630, 1993.

[3] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[4] A. Y. Halevy, N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 778–787, New York, NY, USA, 2005. ACM Press.

[5] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier, Sept. 2003.

[6] S. Jain and P. K. Kannan. Pricing of information products on online servers: Issues, models, and analysis. *Management Science*, 48(9):1123–1142, 2002.

[7] M. Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[8] G. Nehmauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[9] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over web services. In *VLDB '06*, 2006.

[10] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB Journal: Very Large Data Bases*, 5(1):48–63, 1996.

[11] M. Stonebraker and J. M. Hellerstein. Content integration for e-business. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 552–560, New York, NY, USA, 2001. ACM.

APPENDIX

PROOF OF THEOREM 1. Suppose we had an algorithm that could find a solution to FULLROWMAXUTILITY in polynomial time that was guaranteed to be within $1/f$ of the optimal solution. We will show how such an algorithm can be used to solve the SETCOVER problem exactly, which is NP-hard. The intuition behind the proof is that the completion of each row is in essence a set cover problem, so maximizing the number of completed rows is tantamount to maximizing the number of set cover instances solved. Yet not even a single set cover instance can be solved in polynomial time.

Consider an instance (S, U, k) of the set cover problem, where U is the universe of elements which must be covered, S is the set of subsets of U, and the goal is to cover U with k elements of S. We reduce this instance of set cover to FULLROWMAXUTILITY as follows. The canonical table D will contain only a single row, and $n = |U|$ columns, one for each element of the set cover universe. Each set in S will be represented by a single provider, selling the subset of columns corresponding the elements in S. Finally, the utility of the single row will be 1, the cost of each provider's subset of the columns is 1, and the user's budget is k.

It is easy to see that in the above formulation, a solution to the set cover instance exists if and only if there is there is a solution to the FULLROWMAXUTILITY problem achieving utility 1. If such a solution exists, then since the providers have uniform cost, the subsets of k providers cover the entire row, and hence the corresponding k sets in S cover U. On the other side, if a solution to the set cover instance exists, then the corresponding k providers will cover the entire row, achieving utility 1 with budget $B = k$.

Finally, we need only to see that any approximation algorithm for FULLROWMAXUTILITY in fact has equivalent behavior to an exact algorithm, on input generated from this reduction. If a utility 1 solution exists, any f-approximation algorithm will find it, because it is guaranteed to find a solution having utility $\geq \frac{opt}{f} = \frac{1}{f}$. Yet no fractional solutions exist to these instances, so such a solution must itself have utility ≥ 1. On the other hand, if no utility 1 solution exists, the maximum utility achievable is 0, so the approximation algorithm will neccessarily produce a utility 0 solution. Hence, an f approximation algorithm for FULLROWMAXUTILITY for $f > 0$, is equivalent to an exact algorithm for the purposes of the reduction from Set Cover. Our reduction clearly operates in polynomial time, so an f-approximation algorithm for cannot exist unless Set Cover is in P, that is, unless P = NP. □

PROOF OF THEOREM 2. We prove this result again by reduction from set cover. We will use the same mapping as to FULLROWMAXUTILITY but because we will show that the reduction to the minimization version is *approximation preserving*. This means that any approximation algorithm for FULLROWMINCOST will imply an approximation algorithm with the same approximation factor for Set Cover. Since Set Cover isknown to be $\log n$-inapproximable [3], this implies that FULLROWMINCOST is $\log n$-inapproximable as well.

Consider the reduction from set cover to FULLROWMAXUTILITY described above, with the modification that the budget is left to vary, and the utility that must be achieved is fixed to 1. We show that for each solution to the resulting FULLROWMINCOST instance, using budget b, there exists a corresponding solution to the set cover instance with cost b, and vice versa. Note that we are now considering the minimization version of Set Cover, rather than simply the decision problem. A solution to the Set Cover instance having cost b is a set of b sets in S, that cover U. Each of these sets s_i corresponds to a provider for the FULLROWMINCOST problem, selling the subset of columns corresponding to the elements of s_i. Hence, the union of these s_i cover the entire row, implying a b-budget solution (since each provider has unit cost). On the other side, a b-budget solution to the FULLROWMINCOST instance is a set of b providers whose columns cover the entire row. Hence, the corresponding sets in the set cover instance cover all of U, and there is a b-cost solution to the set cover instance. Since there is a bijection between solutions of the set cover instance and the FULLROWMINCOST instance, and the corresponding solutions have the same cost, the optimum values of the instances are the same. Any f-approximation algorithm for FULLROWMINCOST will achieve a cost of $f \bullet opt$, which is easily converted to a solution of cost $f \bullet opt$ for the set cover instance. Hence, the reduction is approximation preserving.

□

PROOF OF THEOREM 3. The proof is by reduction to MAX-SUBMODULAR. Define a *purchase* to be a row-provider pair (i, k), which indicates that row i was bought from provider k. Let f be a function from sets of purchases to real numbers that indicates the utility of that set of purchases. More precisely, for a set of purchases S, let

$$P_S = \{(i, j) \ : \ \exists k \text{ such that } (i, k) \in S \text{ and } j \in C_k\}$$

be the set of row column pairs contained in that purchase, and let $f(S) = \sum_{(i,j)\in P_S} u_{ij}$.

The function f is clearly monotone. We also claim that it is submodular. To see this, consider two sets of purchases $S \subseteq T$, and a purchase $(i, k) \notin T$. Let $S' = S \cup \{(i, k)\}$ and $T' = T \cup \{(i, k)\}$. Then

$$f(S') - f(S) \quad = \quad \sum_{(i',j)\in(P_{\{(i,k)\}} \setminus P_S)} u_{i'j}$$

and

$$f(T') - f(T) \quad = \quad \sum_{(i',j)\in(P_{\{(i,k)\}} \setminus P_T)} u_{i'j} \ .$$

Since $S \subseteq T$, we have $P_S \subseteq P_T$, and hence $P_{\{(i,k)\}} \setminus P_S \supseteq P_{\{(i,k)\}} \setminus P_T$. Therefore, $f(S') - f(S) \geq f(T') - f(T)$, which proves that f is submodular.

Given that the price of each row is p, let $\alpha = \lfloor B/p \rfloor$. Without loss of generality, we can assume that the optimal solution makes α purchases from providers, since f is monotone. Hence, the PARTIALROWMAXUTILITY problem is the problem of selecting a set of α purchases that maximizes the utility f. Since f is monotone submodular, by Theorem 3, the greedy algorithm for this problem achieves an approximation of $1 - 1/e$. □

PROOF OF THEOREM 5. In order to prove this theorem, we reduce it to a problem we call KCHOICEKNAPSACK. An instance of KCHOICEKNAPSACK consists of n items that may be purchased, each with k variants. Each variant of each item has an arbitrary utility for acquiring it, but each items must be purchased in only one or zero of its k variants. The costs are more constrained: there are only k possible costs total, one for each variant, and the ith variant of each item has the same costs as the ith variant of each other. The goal, then, is to maximize the total utility of the items purchased, while remaining under a budget B. Formally, an instance is composed of an utility matrix $U \in \mathcal{R}^{n \times k}$, a cost vector $C \in \mathcal{R}^k$, and a budget $B \in \mathcal{R}$.

The straightforward reduction from PARTIALROWMAXUTILITY to KCHOICEKNAPSACK relies on the observation that the choices we make for the purchasing columns for a given row are independent of those for another row, except in so far as we must choose how much we wish to spend on a given row based on the necessary expenditure for the other rows. This is where the k-choice component comes in. If there are ℓ providers, then for a given row there are $\leq 2^\ell$ ways to buy entries from that row: we can purchase the columns for that row offered by any subset of the providers; (note that multiple subsets of the providers may share both the same total cost and utility for that row). Furthermore, the cost of a given subset of providers is the same for every row. Thus, letting $k = 2^\ell$, we can reduce the problem to an instance of KCHOICEKNAPSACK where each item corresponds to a row, and each variant corresponds to a subset of the providers we can purchase it from.

KCHOICEKNAPSACK remains a hard problem in general, but if we constrain the number of choices k to a constant, corresponding to a constant number of providers in PARTIALROWMAXUTILITY, we show that the problem has a polynomial time algorithm.

We describe a dynamic programming-based algorithm for KCHOICEKNAPSACK which computes the optimal solution in time $Poly(k, n, m)$. Since ℓ is a constant, and the reduction from PARTIALROWMAXUTILITYto KCHOICEKNAPSACK sets $k = 2^\ell$, the algorithm runs in $Poly(n, m)$.

At each level of the dynamic programming, we will compute the optimal solution using some restricted piece of our budget, after making only the first i choices. Specifically, we define the function $q(b, i)$ to be the maximum utility that can be achieved, by expending cost b on the first i items. The update procedure is defined as follows.

$$q(b, i) = max\{q(b, i-1), max_k\{(u_{i,k} + q(b - c_i, i-1)\}\}$$

, where $u_{i,k}$ is the utility of the kth variant of the ith item, and c_i is the cost of the ith item.

The intuitive explanation for the above update formula is that for each item, we may choose not to make any choice, thus our cost and utility choosing among the first i items remains that of choosing among the first $i-1$. Otherwise, we can make one of the k choices, and earn the utility of that choice, along with the maximum utility achievable by spending the remaining budget on the first i items.

To find our final result, we must compute $q(B, m)$. In order to do this, we compute each value $q(b, i)$ for integers $b \leq B$ and $i \leq m$, computing the function for smaller parameters first, and using these for larger values. Note that the dependency of computing the values for a given pair of parameters depends only parameter pairs in which i is decreased, so there is a valid order in which to evaluate the function differing parameters.

We now need only show that the number of parameter pairs on which we must evaluate q is polynomial in m and n; since the update procedure takes polynomial time (assuming that the dependent values have already been computed), this will imply that the full algorithm runs in polynomial time. B may be super-polynomial in m and n. Fortunately, it is easy to see that we need only compute $q(b, i)$ for values of b that are potential costs of choosing variants of the items. There are at most k different costs to consider; since there are m items, we need only consider values $b = B - \sum_{1 \leq j \leq k}(a_j \cdot c_i)$ for all sets of k integers a_j s.t. $\sum_{1 \leq j \leq k} a_j \leq m$. Here a_j indicates the number of items for which variant j is selected. There are clearly fewer than m^k such sets of a_j, since each a_j can take one of at most m different values. Hence, only m^k values of b must be considered. Furthermore, there are at most m values for the second parameter, hence the total number of pairs of parameters for which q must be computed is m^{k+1}, which, because ℓ is constant, is polynomial in m. □