Efficient Information Gathering on the Internet*
(*extended abstract*)

O. Etzioni † S. Hanks ‡ T. Jiang§ R. M. Karp ¶ O. Madani ‖ O. Waarts**

Abstract

The Internet offers unprecedented access to information. At present most of this information is free, but information providers are likely to start charging for their services in the near future. With that in mind this paper introduces the following information access problem*: given a collection of n information sources, each of which has a known time delay, dollar cost and probability of providing the needed information, find an optimal schedule for querying the information sources.*

We study several variants of the problem which differ in the definition of an optimal schedule. We first consider a cost model *in which the problem is to minimize the expected total cost (monetary and time) of the schedule, subject to the requirement that the schedule may terminate only when the query has been answered or all sources have been queried unsuccessfully. We develop an approximation algorithm for this problem and for an extension of the problem in which more than a single item of information is being sought. We then develop approximation algorithms for a* reward model *in which a constant reward is earned if the information is successfully provided, and we seek the schedule with the maximum expected difference between the reward and a measure of cost. The monetary and time costs may either appear in the cost measure or be constrained not to exceed a fixed upper bound; these options give rise to four different variants of the reward model.*

1. Introduction

The Internet is rapidly becoming the foundation of an information economy. Valuable information sources include on-line travel agents, nationwide Yellow Pages, job listing services, on-line malls, and many more. Currently, most of this information is available free of charge, and as a result parallel search tools such as MetaCrawler [12] and BargainFinder [7] respond to requests by querying numerous information sources simultaneously to maximize the information provided and minimize delay. However, information providers are likely to start charging for their services in the near future [9]. Billing protocols to support an "information marketplace" have been announced by large players such as Visa and Microsoft [11] and by researchers [14].

Once billing mechanisms are in place, Internet users will have to balance speedy access to information against the cost of obtaining that information. Clearly, the speediest information gathering plan would be to query every potential information source simultaneously, but that plan may well be prohibitively expensive. The most frugal alternative—querying the information sources sequentially—may prove to be prohibitively slow. This observation suggests the following *information access problem*: given a collection of n information sources, each of which has a known time delay, dollar cost and probability of providing the needed information, find an optimal schedule for querying the information sources.

This paper presents several optimization models for the information access problem that vary according to the objective function. In all cases there are n information sources. The ith information source is described by three numbers: its *execution time* t_i (also referred to as its *time cost*), its *dollar cost* d_i, and its *success probability* p_i. The *failure probability* of source i is $1 - p_i$, which we denote by q_i. A source is said to succeed if it provides the answer to the query. The event that a given source succeeds is assumed to be independent of the success or failure of the other sources.

A schedule can be represented as a partial function from the set of sources to the nonnegative reals. A source is in the domain of this function if and only if there is a possibility of executing it. The function value associated with source i is denoted s_i; source i will be initiated at time s_i unless some query succeeds at or before time s_i. Execution of the schedule is terminated either when some source returns a correct answer or when all sources in the domain of the function have completed their execution. Since each source in a schedule succeeds probabilistically, a schedule generates a probability distribution over *outcomes*, where each outcome is one possible way that the schedule's sources might respond to the query. We use $D(\mathcal{O})$ and $T(\mathcal{O})$ to denote respectively the total dollar cost and time cost of outcome $\mathcal{O}$. Within this framework we study two forms for the objective function, which we call the *reward* and *cost* models.

In the cost model, a schedule assigns a start time s_i to every source. The *completion time* of source i is $s_i + t_i$. Source j *precedes* source i if its completion time is less than or equal to s_i. In an execution of the schedule, source i is queried if and only if no source that precedes it has succeeded. It follows that the schedule terminates only when the question has been answered or all sources have been queried. Thus the probability that source i is queried is the product of the failure probabilities of all the sources that precede it. The time cost of a schedule is a random variable which is equal to the earliest completion time of a source that succeeds,

*Research supported in part by Office of Naval Research grant 92-J-1946, ARPA / Rome Labs grant F30602-95-1-0024, a gift from Rockwell International Palo Alto Research, National Science Foundation grant IRI-9357772, Natural Science and Engineering Research Council of Canada Research grant OGP0046613, and Canadian Genome Analysis and Technology Grant GO-12278.

†U. of Washington. etzioni@cs.washington.edu

‡U. of Washington. hanks@cs.washington.edu

§On research leave from Dept. of Comp. Sci., McMaster University, Hamilton, Ont L8S 4K1, Canada. jiang@maccs.mcmaster.ca

¶U. of Washington. karp@cs.washington.edu

‖U. of Washington. madani@cs.washington.edu

**U. of California at Berkeley. Work supported in part by NSF postdoctoral fellowship. waarts@cs.berkeley.edu

and the dollar cost of a schedule is a random variable which is equal to the sum of the dollar costs of the sources that are queried. The *overall cost* of a schedule is the sum of its time cost and its dollar cost. The unit of time can be chosen appropriately so that the sum obtained is a *weighted* sum of dollar cost and time cost. We seek a schedule of minimum expected overall cost. In this model a schedule will always include *all* the sources, and the problem is to determine the order in which they are queried and which should be run simultaneously.

In this model, we also study a more general version of this problem in which the objective is to retrieve $m > 1$ items of information. We assume that the ith source has dollar cost d_i, time cost t_i and, for $j = 1, 2, \cdots, m$, probability p_i of successfully providing the jth item of information. In this case we require that p_i is bounded away from 1 by a constant.

In the reward model a schedule may not include all sources and a schedule may terminate even though the question has not been answered and not all sources have been queried. We assume a constant known reward R which is collected just in case some source returns a correct answer. Let $S(\mathcal{O}) = 1$ if some source in $\mathcal{O}$ successfully answers the query, and $S(\mathcal{O}) = 0$ if none does. The *value* of an outcome $\mathcal{O}$ is $R \cdot S(\mathcal{O})$ less some function of $D(\mathcal{O})$ and $T(\mathcal{O})$. The *expected value* of a schedule $\mathcal{P}$, denoted $V(\mathcal{P})$, is simply the expectation of the value taken over all the schedule's possible outcomes. Our objective is to find a schedule that maximizes $V(\mathcal{P})$.

We consider four variants of the objective function corresponding to cases where $D(\mathcal{O})$ and $T(\mathcal{O})$ are linear in dollar cost (time cost) or are threshold constraints on the amount of money (time) the schedule can consume. In the threshold cases the problem is to find the schedule with the maximum expected value subject to the constraint that the schedule never violates the threshold constraint(s). For example, the **TL** model (see Figure 1) represents the case where there is a dollar cost threshold but the objective function is linear in the schedule's duration. In this case the problem is to find the schedule that maximizes expected reward less expected duration subject to the constraint that the total dollar cost of executing the schedule does not exceed the threshold.

Observe that the cost and reward models are conceptually distinct in that the reward model must address both the question of *which* sources to consult and *when* to consult them whereas the cost model addresses only the latter.

Figure 1 summarizes the problems we address within the reward and cost models. We will hereafter refer to the problems by their acronyms. The four reward-model problems are **LL** for linear in dollar cost and time cost, **LT** for linear in dollar cost and threshold in time cost, **TL** for threshold in dollar cost and linear in time cost, and **TT** for threshold in dollar cost and time. The cost-model problem is **CO** (cost only). With suitable scaling of the dollar and time costs, the objective functions for the models assume the forms given in Figure 1.

1.1 Our Results

We will first summarize the results for the cost model. We develop an algorithm that runs in time $O(n^2)$ and constructs a schedule which achieves the approximation ratio $2 \times 4 \times 4$. Each of these factors is the result of a different transformation in the construction of our algorithm, as described later in the paper. The manner in which we construct our algorithm is a key idea in this part of the paper.

Next, for the cost model, we consider the general case of the information access problem in which there are m items of information being sought, and a query to a given source asks for all the items. In contrast with the case of a single item of information, in the general case the optimal schedule may need to be adaptive; *i.e.*, the decisions of the scheduling algorithm may depend on which items of information have already been gathered. Despite this complication, we give an algorithm which runs in polynomial time and gives a schedule whose expected overall cost is within a constant factor of the optimal expected overall cost. It is somewhat surprising that the approximation ratio is independent of m as well as n.

Turning now to the reward model, we show that finding an optimal schedule is NP-hard in each of the four cases. A fully polynomial time approximation scheme (FPTAS) is obtained for the model **TT**, using an extension of the well-known rounding technique for Knapsack [6]. The FPTAS also works for the model **TL** under a weak assumption: that every source is "profitable" individually according to the **TL** objective function, *i.e.* for every source i, $p_i - t_i \geq 0$.

The approximation algorithms for the case **LT**—where the objective function is linear in total cost subject to a time threshold—are perhaps the most interesting among the reward-model problems. For this case we make the simplifying assumption that the duration parameters t_i are the same. This assumption is powerful because it allows us to consider scheduling sources in simultaneous "batches": all sources will be scheduled at $t = 0, d, 2d, \ldots$, where d is the common duration. Although not fully general, this is a reasonable model of the current and probable future state of information access on the Internet.

We will first present an $O(n^2)$ time approximation algorithm with ratio $\frac{1}{2}$ for optimal single-batch schedules, then extend it to a polynomial time approximation scheme (PTAS). For any constant $r > 1$, the PTAS runs in time $O(n^{r+1})$ to achieve an approximation ratio $\frac{r-1}{r+1}$. The algorithms are simple and are similar to the ones in [10] for Knapsack, but the analyses are different and are more sophisticated. We then design an approximation algorithm with ratio $\frac{1}{6}$ for optimal k-batch schedules, running in time $O(kn^2)$. The algorithm is based on the ratio-$\frac{1}{2}$ algorithm for single-batch schedules, but it also involves some new ideas.

Due to lack of space, most proofs are omitted or only sketched. The proofs for the cost model appear in [2] and those for the reward model appear in [1].

1.2 Related Work

Scheduling problems have been studied in many contexts including scheduling on parallel machines, processor allocation, etc. (see [8] for a survey). Our Internet-inspired query scheduling problem has a unique flavor, however, due to the need to balance the competing time and cost constraints on schedules with unbounded parallelism. In addition, in our problem, once an answer is obtained, no other queries need be made.

If we constrain the schedules to be sequential, then an optimal solution can be found in polynomial time (see subsection 3.2 for the **LT** case). Similar problems have been addressed in [4, 13] and elsewhere. The difference in this

Objective fn	linear in time	time threshold
linear in cost (w/reward)	**LL**: min $\mathbf{E}[S(\mathcal{O}) - D(\mathcal{O}) - T(\mathcal{O})]$	**LT**: max $\mathbf{E}[S(\mathcal{O}) - D(\mathcal{O})]$ s.t. $\forall \mathcal{O}\ \ T(\mathcal{O}) \leq \tau$
cost threshold (w/reward)	**TL**: max $\mathbf{E}[S(\mathcal{O}) - T(\mathcal{O})]$ s.t. $\forall \mathcal{O}\ \ D(\mathcal{O}) \leq \varsigma$	**TT**: max $\mathbf{E}[S(\mathcal{O})]$ s.t. $\forall \mathcal{O}\ \ D(\mathcal{O}) \leq \varsigma$ and $T(\mathcal{O}) \leq \tau$
cost linear (no reward)	**CO**: min $\mathbf{E}[D(\mathcal{O}) + T(\mathcal{O})]$	

Figure 1: The five objective functions. $\mathcal{O}$ denotes a possible outcome of the schedule $\mathcal{P}$ to be found.

paper is the ability to query any number of sources in parallel. [3, 5] study scheduling tasks with unlimited parallelism with some similarity to the **LL** and **CO** models, but the positive results in [3, 5] are limited to an exponential-time dynamic programming algorithm and some heuristics.

2 The Cost Model

2.1 Batch Schedules for a Single Item of Information

Issuing the query to information source i is referred to as *performing job* i. We define a mathematical notion called a *fraction of a job*, or equivalently, a *fractional job*, as follows: an α-fraction of job i, where $0 \leq \alpha \leq 1$, has dollar cost $\alpha \cdot d_i$, time cost t_i, and probability of failure q_i^α. Thus the dollar cost is assessed in proportion to the fraction α, the full time cost is charged regardless of the fraction α, and the failure probability is chosen so that, if a job is broken into fractional jobs with fractions summing to 1, then the product of the failure probabilities of the fractional jobs is equal to the failure probability of the entire job. Note that each job is also a fractional job, since it is a 1-fraction of itself. An α-fraction of a job, where $\alpha \notin \{0, 1\}$, is also called a *strictly fractional job*. Our strategy is to first construct a schedule in which any given job may be split into fractional jobs with fractions summing to 1, and then to convert this fractional schedule into one without strictly fractional jobs.

A *batch schedule* is one in which the sources are partitioned into an ordered sequence of subsets called *batches*. The first batch is started at time 0 (*i.e.* all sources in the first batch are queried at time 0), and, in general, batch $i+1$ is started upon the completion of the last job in batch i, provided that no job in the first i batches has succeeded. Batch schedules are not fully general, since they do not allow two jobs to overlap unless they start at the same time, but we show that the restriction to batch schedules costs only a small constant factor in the expected overall cost.

case putting a job in a batch increases the probability of execution

A *fractional batch schedule* is constructed by breaking some of the jobs into strictly fractional jobs with fractions summing to 1, and then constructing a batch schedule using the resulting set of jobs.

Given a (fractional) batch schedule R, denote its ith batch by R_i. The costs and failure probabilities of the batches of R are defined in a natural way as follows. The *dollar cost of the ith batch*, denoted by $D(R_i)$, is defined as the sum of the dollar costs of the jobs (and fractions of jobs) contained in it. The *time cost of the ith batch*, denoted by $T(R_i)$, is defined as the maximum time cost among the jobs and fractional jobs it contains. (Note that the actual time spent executing a batch may be somewhat smaller than its defined time cost since an answer may be obtained before all the jobs in the batch have been completed; however, the above definition suffices for our purposes.) The *overall cost of the ith batch*, denoted by $OC(R_i)$, is the sum of its dollar cost and its time cost. The *failure probability of the ith batch*, denoted by $Q(R_i)$, is the product of the failure probabilities of all jobs (including the strictly fractional ones) contained in the batch. Its success probability is denoted by $P(R_i) = 1 - Q(R_i)$. We define $Q(R_0) = 1$ and $C(R_0) = T(R_0) = 0$. For example, if the ith batch contains jobs $i_1, \ldots, i_k$ and an α-fraction of job i_{k+1}, then $D(R_i) = \alpha d_{i_{k+1}} + \sum_{j=1}^{k} d_{i_j}$; $T(R_i) = \max_{1 \leq j \leq k+1}\{t_{i_j}\}$; $OC(R_i) = D(R_i) + T(R_i)$; and $Q(R_i) = 1 - P(R_i) = q_{i_{k+1}}^{\alpha} \cdot \prod_{1 \leq j \leq k} q_{i_j}$.

The expected overall cost of a batch schedule R is the sum of its expected dollar and time costs.

We refer to jobs whose probability of success is greater than $1/2$ as *heavy jobs* and to all other jobs as *light jobs*. A batch that consists only of fractions of light jobs (recall that whole jobs are a special case of fractional jobs) is called a *light batch*, and a batch that consists of a single whole heavy job is called a *heavy batch*. Note that in general a batch may be neither light nor heavy.

Finally, we call a fractional batch schedule *balanced* if each of its batches is either light or heavy, each of its light batches except the last light batch has failure probability exactly $1/2$, and the last light batch has failure probability greater than or equal to $1/2$.

2.2 The Greedy Schedule

Our schedule is a batch schedule. Its batches are constructed in three steps. In the first step we put aside the heavy jobs and construct a balanced fractional batch schedule from the light jobs. In this schedule the last batch has failure probability greater than or equal to $1/2$, and each of the other batches has failure probability $1/2$. We call this schedule the *light fractional greedy schedule* and denote it by LFG. In the second step, we construct a balanced schedule such that each of its batches is either a batch of LFG or a single heavy job. We call this schedule the *balanced greedy schedule* and denote it by BG. In the third step we convert BG into a non-fractional batch schedule by combining the fractions of each strictly fractional job in BG and placing the resulting whole job in an appropriate batch. This schedule is called the *greedy schedule* and is denoted by G. The greedy schedule is our final schedule, and our main result in the single query case is that the expected overall cost of the greedy schedule is within a constant factor of the optimal expected overall cost.

The Light Fractional Greedy Schedule The light fractional greedy schedule uses only the original light jobs. Some of these jobs may be broken into fractional light jobs with fractions summing to 1. The batches are constructed successively, starting with batch 1. We now describe the construction of batch i. Let α_{ik} be the fraction of the kth light job occurring in batch i. Then the α_{ik} are nonnegative and, for each k, $\sum_i \alpha_{ik} = 1$.

In general, given batches $1, 2, \cdots, i-1$, batch i is constructed to be of minimum overall cost, such that:

1 for each k, $\alpha_{ik} \leq 1 - \sum_{j=1}^{i-1} \alpha_{jk}$;

2 $\prod_k q_i^{\alpha_{ik}}$, the failure probability of batch i, is equal to $1/2$;

3 Batch i contains at most one job k such that $\alpha_{ik} > 0$ and $\sum_{j=1}^{i} \alpha_{jk} < 1$. Such a job is said to be *partially completed in batch* i.

It turns out that, among the minimum-cost choices of batch i satisfying the first two conditions, there is one that also satisfies the third.

An exception to the second condition occurs when the fractional jobs remaining are not sufficient to yield a failure probability as small as $1/2$. In that case, all the remaining fractional jobs are placed in a single final batch.

In subsection 2.4 we show how the above batches can be selected efficiently.

The Balanced Greedy Schedule Each batch of LFG occurs as a batch in BG. In addition, each original heavy job occurs by itself as a batch in BG. Subject to this requirement, BG is constructed to be of minimum expected overall cost. This is achieved by sorting the two types of batches (batches of LFG and batches consisting of a single heavy job) in increasing order of the ratio OC/P, where OC is the overall cost of the batch and P is its success probability, and executing the batches in that order, halting as soon as some fractional job is successful.

The Greedy Schedule We start with the balanced greedy schedule BG and combine strictly fractional jobs appearing in it, in order to obtain batches that do not contain strictly fractional jobs. The combining is done as follows. Let k be a job that occurs fractionally in more than one batch of BG. Let α_{ik} be the fraction of job k appearing in batch i of BG; note that, if batch i is heavy, then $\alpha_{ik} = 0$. Let P_i be the probability that batch i of BG is executed. Let $f_k = \sum_{i=1}^{\infty} \alpha_{ik} P_i$. Thus, f_k is the expected fraction of job i that is executed in a run of BG. Job k is moved to batch i, where i is the least index satisfying $P_i < 2 f_k$. This move is motivated by the wish to approximately preserve the expected overall cost of the schedule.

2.3 Analysis of the Greedy Schedule

The analysis proceeds in three steps. The first step shows that the expected overall cost of the balanced greedy schedule is at most twice the expected overall cost of any balanced schedule. The second step shows that the expected overall cost of the greedy schedule is at most four times the expected overall cost of the balanced greedy. The third step shows that there is a balanced schedule whose expected overall cost is at most four times the expected overall cost of the optimal schedule. Combining these results, we find that the expected overall cost of the greedy schedule is at most $2 \times 4 \times 4$ times the expected overall cost of an optimal schedule.

2.3.1 Balanced Greedy is Almost Optimal among Balanced Schedules

The main result of this subsubsection is the following theorem.

Theorem 2.1 *The expected overall cost of the balanced greedy schedule is at most twice the expected overall cost of any other balanced schedule.*

Let Schedule A be an arbitrary balanced schedule. Let A_i denote the ith batch of schedule A. We construct from A a new schedule ALG whose ith batch is denoted ALG_i. ALG is constructed from A by replacing the light batches of A with the corresponding batches of LFG while leaving the heavy batches of A unchanged. Thus, if A_i is heavy, then $ALG_i = A_i$; otherwise, if A_i is light, and it is the jth light batch in A (*i.e.* is preceded in A by $j-1$ light batches), then ALG_i is the jth batch of LFG.

The following lemma states the key observation of this subsubsection:

Lemma 2.2 *For each* $i = 1, \ldots, \infty$, $OC(ALG_i) \leq \sum_{j=1}^{i} D(A_j) + \max_{j=1,\ldots,i} T(A_j)$

Sketch of Proof: For the heavy batches there is nothing to prove, since they are not changed in passing from A to ALG. For the light batches, we argue as follows. For each r, since the first r light batches of A each has failure probability $1/2$, and the first $r-1$ batches of LFG each has failure probability $1/2$, it must be possible to construct an rth light batch, say batch B, from fractional jobs contained in the first r light batches of A but not in the first $r-1$ batches of LFG. The overall cost of such a batch B would not exceed $\sum_{j=1}^{i} D(A_j) + \max_{j=1,\ldots,i} T(A_j)$.

On the other hand, by construction of LFG, the rth light batch of LFG is the light batch of minimum overall cost which has failure probability $1/2$ and can be constructed from the fractional parts of jobs remaining after the first $r-1$ batches of LFG have been constructed (the last batch of LFG is exceptional, as its failure probability may be greater than $1/2$. This complication is easily handled.) Thus the overall cost of the rth batch of LFG is less than or equal to the overall cost of batch B, which, as stated above, is $\leq \sum_{j=1}^{i} D(A_j) + \max_{j=1,\ldots,i} T(A_j)$. ■

Lemma 2.3 *The expected overall cost of schedule ALG is at most twice the expected overall cost of A.*

Proof. Let W_i be the probability that ALG executes its ith batch. Then for $i > 1$, $W_i \leq W_{i-1}/2$, since each batch of ALG except the last has success probability at least $1/2$. Lemma 2.2 thus implies:

$$\begin{aligned} OC(ALG) &= \sum_{j=1}^{\infty} W_j \cdot OC(ALG_j) \\ &\leq \sum_{j=1}^{\infty} W_j \cdot \left(\sum_{i=1}^{j} D(A_i) + \max_{i=1,\ldots,j} T(A_i) \right) \end{aligned}$$

$$\leq \sum_{j=0}^{\infty} 2^{-j} \cdot \left(\sum_{i=1}^{\infty} (D(A_i) + T(A_i)) \cdot W_i \right)$$

$$\leq 2 \sum_{i=1}^{\infty} OC(A_i) \cdot W_i \ .$$

On the other hand, note that it follows from the construction of schedule ALG that the probability of executing A_i in A is exactly the same as the probability of executing ALG_i in ALG. Thus, $OC(A) = \sum_{i=1}^{\infty} OC(A_i) \cdot W_i$, and the claim follows.

Next we show:

Lemma 2.4 *The expected overall cost of the balanced greedy schedule BG is not greater than the expected overall cost of ALG.*

Sketch of Proof: Observe that BG can be obtained from ALG by reordering the batches of ALG in increasing order of their ratios OC/P, where OC is the expected overall cost of the batch and P is its success probability. An easy interchange argument shows that this reordering does not increase the expected overall cost of the schedule. ■

Lemmas 2.3 and 2.4 immediately imply the above Theorem 2.1.

2.3.2 Comparing the Greedy Schedule with the Balanced Greedy Schedule

In this subsubsection we show that the expected overall cost of the greedy schedule is at most four times the expected overall cost of the balanced greedy schedule.

Let BG_i denote the ith batch of BG, and let G_i denote the ith batch of G.

Lemma 2.5 *The probability of executing batch G_i in G is at most twice the probability of executing batch BG_i in BG.*

Sketch of Proof: Recall that α_{ik} denotes the fraction of light job k executed in batch BG_i, P_i denotes the execution probability of BG_i, and $f_k = \sum_{i=1}^{\infty} \alpha_{ik} P_i$ denotes the expected fraction of light job k executed during an execution of BG. Schedule G assigns light job k to a batch G_j, where j is the least index satisfying $P_j < 2f_k$. The assignment of heavy jobs to batches does not change in passing from BG to G; *i.e.*, a heavy job occurring in BG_i is assigned to G_i.

Recall that job k is said to be partially completed in BG_i if $\alpha_{ik} > 0$ and $\sum_{j=1}^{i} \alpha_{jk} < 1$. Any increase in the probability of executing batch G_i in G over the probability of executing batch BG_i in BG is accounted for by the movement of some light job that is partially completed in some light batch BG_j of BG, where $j < i$, but is assigned to some batch G_r of G, where $r \geq i$.

It is enough to consider $i > 1$. By the construction of BG each such batch BG_j mentioned above contains at most one partially completed job. Moreover, if a partially completed job k in BG_j is moved to batch G_r, where $r \geq i$, then $P_{i-1} \geq 2f_k$. Since $\alpha_{jk} P_j \leq f_k$ it follows that $\alpha_{jk} \leq \frac{P_{i-1}}{2P_j}$. Thus, if $j = i-1$, then $\alpha_{jk} = \alpha_{i-1,k} \leq 1/2$; otherwise, suppose there are t light batches in BG with indices greater than or equal to j but less than $i-1$. Then since each light batch has failure probability $1/2$, $P_{i-1} \leq 2^{-t} P_j$, from which it follows that $\alpha_{jk} \leq 2^{-t-1}$.

The probability that the α_{jk}-fraction of job k in BG_j fails is $(1-p_k)^{\alpha_{jk}} \geq 2^{-\alpha_{jk}}$. The inequality follows from the fact that $p_k \leq 1/2$, since job k is a light job. Hence the movement of job k from light batch BG_j to batch G_r, where $r \geq i$, increases the ratio between the execution probability of G_i and the execution probability of BG_i by at most the factor $2^{\alpha_{jk}}$. It follows that the ratio between the execution probability of G_i and the execution probability of BG_i is at most $\prod_{t=1}^{i-1} 2^{2^{-t}} \leq 2$. ■

Theorem 2.6 *The expected overall cost of G is at most four times the expected overall cost of BG.*

Proof. We first compare the expected dollar cost of G with the expected dollar cost of BG. A heavy job that occurs in BG_i also occurs in G_i. By Lemma 2.5, its probability of execution in G is at most twice its probability of execution in BG, and hence its contribution to the expected dollar cost of G is at most twice its contribution to the expected dollar cost of BG. A light job that is executed with probability f_k in BG is assigned to a batch G_r such that $P_r < 2f_k$, where P_r is the execution probability of batch BG_r in BG. It follows from Lemma 2.5 that the execution probability of this job in G is at most $2P_r$, which is at most $4f_k$. Hence the contribution of this job to the expected dollar cost of G is at most four times its contribution to the expected dollar cost of BG.

Next we show that the expected time cost of G is at most four times the expected time cost of BG. Let $T(BG_i)$ denote the time cost of batch BG_i, and let $T(G_i)$ denote the time cost of batch G_i. Let $P(G_i)$ denote the execution probability of batch G_i and let P_i denote the execution probability of batch BG_i. Then the expected time cost of BG is $\sum_i P_i T(BG_i)$ and the expected time cost of G is $\sum_i P(G_i) T(G_i)$.

Any increase in $T(G_i)$ over $T(BG_i)$ can be accounted for by the movement of some light job that is partially completed in some light batch BG_j of BG, where $j < i$, and is assigned to G_i. Thus,

$$\begin{aligned} \sum_{i=1}^{\infty} P(G_i) \cdot T(G_i) &\leq \sum_{i=1}^{\infty} P(G_i) \cdot \sum_{j=1}^{i} T(BG_j) \leq \\ \sum_{j=1}^{\infty} T(BG_j) \cdot \sum_{i=j}^{\infty} P(G_i) &\leq \sum_{j=1}^{\infty} T(BG_j) \cdot P(G_j) \cdot \sum_{g=0}^{\infty} 2^{-g} \\ &\leq 2 \cdot 2 \cdot \sum_{j=1}^{\infty} T(BG_j) \cdot P_j \ . \end{aligned}$$

The third inequality follows from the fact that for $i > 1$, $P_i \leq P_{i-1}/2$ (since each batch of BG, except possibly the last one, has success probability at least $1/2$), and the last inequality follows since Lemma 2.5 tells us that $P(G_i) \leq 2P_i$.

2.3.3 Existence of a Low Cost Balanced Schedule

Let `Opt` denote the (unknown) optimal schedule for the given set of jobs. Starting with `Opt`, we construct a balanced schedule called `Bopt` whose expected overall cost is at most four times the expected overall cost of `Opt`.

We describe the construction of the first batch of `Bopt`. For any time T, the probability that `Opt` has an execution time greater than T is equal to the product of the failure probabilities of the jobs terminating by time T. Let T_1 be the least T for which this probability is less than $1/2$. If the set of jobs terminating by time T_1 contains a heavy job, then the first batch of Bopt consists of the earliest heavy job to terminate in Opt. If all the jobs terminating by time T_1 are light, then the first batch of Bopt is constructed as follows. Let the light jobs terminating by time T_1 be arranged in increasing order of their termination times, and let the failure probability of the rth light job in this ordering be q_r. Then there exists an index s and a fraction α such that $\prod_{r=1}^{s-1} q_r \times q_s^{\alpha} = 1/2$. Then the first batch of Bopt is a light batch consisting of the first $s-1$ jobs in the ordering plus an α-fraction of the sth job.

For a general i, the ith batch of Bopt is constructed similarly. First, a reduced schedule $\mathtt{Opt}^i$ is constructed from Opt by deleting the jobs or fractional jobs occurring in the first $i-1$ batches of Bopt. If a total fraction $\beta < 1$ of some job k is executed in the first $i-1$ batches of Bopt (*i.e.* $\beta = \sum_{j=1}^{i-1} \alpha_{ik}$), then job k is replaced in the reduced schedule by a $(1-\beta)$-fraction of job k having the same start time and completion time as k. The ith batch of Bopt is then constructed by applying to this reduced schedule $\mathtt{Opt}^i$ the same construction that was applied to Opt to obtain the first batch of Bopt.

Lemma 2.7 *The expected dollar cost of* Bopt *is at most twice the expected dollar cost of* Opt.

Lemma 2.8 *The expected time cost of* Bopt *is at most four times the expected time cost of* Opt.

Lemmas 2.7 and 2.8 immediately imply:

Theorem 2.9 *The expected overall cost of* Bopt *is at most four times the expected overall cost of* Opt.

2.4 Efficient Construction of the Light Fractional Greedy Schedule

The batches of LFG are constructed as follows.

Let $T_1, \ldots, T_g$ be the distinct time costs among the given light jobs.

Sort the light jobs in increasing order of $d_i / -\ln q_i$. We refer to this list as the *efficiency list*. For each T_h, where $1 \leq h \leq g$, we define the T_h *efficiency list*, as the sublist of the efficiency list that contains all jobs whose time costs do not exceed T_h.

The batches of LFG are constructed successively. We describe the construction of a generic batch i. For each light job k, set β_k equal to $1 - \sum_{j=1}^{i-1} \alpha_{jk}$. Thus β_k is the fraction of job k that is not assigned to the first $i-1$ batches.

If the product over all light jobs k of $q_k^{\beta_k}$ is greater than or equal to $1/2$ then assign all the remaining fractional jobs to batch i and halt; batch i is the final batch of the schedule.

Otherwise, for each T_h, where $1 \leq h \leq g$, do the following:

Compute $\prod q_k^{\beta_k}$ where the product extends over all the fractional jobs on the T_h efficiency list. If this product is less than or equal to $1/2$ then construct a batch called the T_h *candidate batch* as follows. Consider the jobs on the T_h efficiency list in order. When job k is encountered, assign a fraction β_k of job k to the T_h candidate batch, unless doing so would reduce the failure probability of the batch to a value less than or equal to $1/2$. In that case, assign an α fraction of job k to the batch, where α is chosen to make the failure probability of the batch exactly $1/2$, and terminate the construction of the batch.

After performing the above procedure for each T_h, compute the overall cost of each T_h candidate batch, and set the ith batch equal to a T_h candidate batch of minimum overall cost.

Lemma 2.10 *For each $i = 1, \ldots, \infty$, for each T_h, the set of fractional jobs selected for the ith batch in the above fashion has the minimum dollar cost among all possible batches of failure probability $1/2$ that can be constructed subject to the constraint that each fractional job selected has time cost less than or equal to T_h, and, for each k, at most a β_k-fraction of job k is used.*

Corollary 2.11 *If batch i in the above construction has a failure probability equal to $1/2$ then it has the minimum overall cost among all possible batches of failure probability $1/2$ that can be constructed subject to the constraint that, for each k, at most a β_k-fraction of job k is used.*

Theorem 2.12 *Using appropriate data structures for maintaining the T_h efficiency lists, the batches of Schedule LFG can be constructed in time $O(n\ max(g, \log n))$.*

We note that each batch of Schedule LFG contains at most one partially completed job.

2.4.1 Gathering Many Items of Information

We consider the task of obtaining answers to m questions, where m may be greater than 1. Job i consists of issuing a request to information source i for the answers to all m questions. The information source may provide any subset of the answers. The schedule terminates as soon as all questions have been answered or all jobs have been completed. The paper up to now deals with the case $m = 1$.

Job i has dollar cost d_i, time cost t_i and probability p_i of succeeding in answering question j. For technical reasons we require that each p_i is less than $1/2$ (actually, the constant $1/2$ can be replaced by any constant less than 1, at the expense of an increase in the constant approximation ratio). We assume that the events "Job i succeeds in answering question j" are independent.

We have constructed a polynomial-time schedule MG (M stands for many and G stands for greedy) whose expected overall cost is within a constant factor of the expected overall cost of an optimal schedule. The construction is similar to the one given for $m = 1$, proceeding through the construction of a light fractional greedy schedule $MLFG$. Because of our assumption that $p_i < 1/2$ for all i, there are no heavy jobs, and thus, unlike the case $m = 1$, we can pass directly from $MLFG$ to MG without the intermediate step of interleaving the batches of $MLFG$ with batches consisting of single heavy jobs. The construction of $MLFG$ requires the following further changes:

- Independently for each question j, an α-fraction of job i has probability q_i^{α} of failing to answer question j;

- The failure probability of a set of jobs or fractional jobs is defined as the probability that it fails to answer all m questions;

- For each j, the failure probability of the set of jobs or fractional jobs in the first j batches of $MLFG$ is 2^{-j};

The chief difficulty in showing that schedule MG achieves a constant-factor approximation arises from the fact that, in the case $m > 1$, an optimal schedule may be adaptive; *i.e.*, it may not follow a fixed timetable. Instead, its choice of jobs to schedule at any time may depend on the number of questions that have already been answered. Consequently, the analysis of the case $m = 1$ cannot be extended straightforwardly to the case $m > 1$. We overcome this difficulty by showing that there is an oblivious schedule (*i.e.*, one that follows a fixed timetable) for the case $m > 1$ whose expected overall cost is within a constant factor of the expected overall cost of an optimal adaptive schedule. Starting with this oblivious schedule, the rest of our analysis for the single-question case (*i.e.* subsections 2.3.1, 2.3.2, 2.3.3 and 2.4) will apply with minor adjustments.

2.4.2 Existence of an Almost Optimal Oblivious Batch Schedule

Denote the (unknown) optimal schedule for the case $m > 1$ by `Mopt` (M stands for many). `Mopt` can be described as a rooted tree in which each internal node represents a conditional branch based on the number of questions successfully answered by a certain time, and each edge represents a sequence of actions, each of which is the initiation of a given job at a given time. It is required that the schedule always *reaches completion*; *i.e.*, it either answers all m questions or executes all n jobs. The probability that a given root-leaf path is followed is called its *execution probability.*

The sequence of actions along each root-leaf path of `Mopt` constitutes an oblivious schedule. We refer to each such schedule as an *oblivious path* of `Mopt`. Such an oblivious path need not always reach completion, as certain runs of `Mopt` will not satisfy the conditional tests along the path. The probability that an oblivious path of `Mopt` reaches completion will be called its *completion probability.*

The following lemma is the key observation to our construction of an oblivious schedule from `Mopt`.

Lemma 2.13 *Let S be a subset of the set of all root-leaf paths occurring in* `Mopt`*. Then there is an oblivious schedule derived from one of the paths in S whose completion probability is greater than or equal to the sum of the execution probabilities of the paths in S.*

Sketch of Proof: Pick an internal node in S for which all children are leaves, *i.e.* no child of this internal node is an internal node. For each of the paths emanating from this node compute the probability that all jobs on the path will fail. Replace the node and the paths emanating from it by the path of lowest failure probability among these. Repeat until all internal nodes are removed. ■

Using the lemma, we construct an oblivious schedule from `Mopt` as follows. Let the oblivious schedules corresponding to root-leaf paths of `Mopt` be arranged in increasing order of their overall execution costs. Let x be any number in the interval (0,1). Let S_x be the smallest initial segment of the ordering of oblivious schedules to have total execution probability at least x. Let A_x be any oblivious schedule within S_x that has completion probability at least x; by Lemma 2.13 such an schedule must exist. The batches of `Mopt` are constructed successively. Batch i consists of the jobs in the oblivious schedule $A_{1-2^{-i}}$, minus any jobs that occur in previous batches. We refer to the resulting oblivious schedule by `Omopt` (the O stands for oblivious).

Theorem 2.14 *The expected overall cost of the oblivious schedule* `Omopt` *is at most 4 times the expected overall cost of* `Mopt`*.*

Proof. By construction of `Omopt`, the probability that all first i batches in `Omopt` fail is at most 2^{-i}. The expected cost of `Omopt` is thus at most $\sum_{i=1}^{\infty} OC(\texttt{Omopt}_i)2^{-i+1}$.

On the other hand, by construction of `Omopt`, in at least 2^{-i} runs of `Mopt`, the overall cost of `Mopt` is $\geq OC(\texttt{Omopt}_i)$. Define $OC(\texttt{Omopt}_0) = 0$. Then, the expected cost of `Mopt` is at least:

$$\sum_{i=1}^{\infty} 2^{-i}(OC(\texttt{Omopt}_i) - OC(\texttt{Omopt}_{i-1})) \geq \sum_{i=1}^{\infty} 2^{-i-1} OC(\texttt{Omopt}_i) \,,$$

and the claim follows.

3. The Reward Models

3.1 The Complexity of Computing Optimal Schedules

We prove that computing an optimal schedule in any of the reward models is NP-hard, by reductions from the Partition Problem. The only subtlety is that the constructions require exponentiation.

Theorem 3.1 *Finding an optimal schedule in any of the variations of the reward model is NP-hard.*

3.2 The LT Model

The following simple facts and definition will be useful throughout our discussion of the **LT** model. The first lemma shows the subadditivity of the objective function for batched schedules.

Lemma 3.2 *Let* OPT_0 *be an optimal k-batch schedule. For any partition of* OPT_0 *into two subschedules* OPT_1 *and* OPT_2*, where the sources in* OPT_1 *and* OPT_2 *are scheduled in the same batches as they are in* OPT_0, $V(\mathrm{OPT}_0) \leq V(\mathrm{OPT}_1) + V(\mathrm{OPT}_2)$.

Lemma 3.3 *Suppose that $\mathcal{P}$ is any k-batch schedule, i is an index between 1 and k, and j is a source not appearing in $\mathcal{P}$. Let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ denote the subschedules consisting of the first $i-1$ batches, the i-th batch, and the last $k-i$ batches of $\mathcal{P}$, respectively. Also denote the expected cost and collective success probability of the sources in schedule $\mathcal{P}_l$ as D_l and P_l, $l = 1, 2, 3$. Then adding source j to the i-th batch of schedule $\mathcal{P}$ increases its expected value by: $V(\mathcal{P} \cup \{j\}) - V(\mathcal{P}) = (1 - P_1)p_j((1 - P_2)(1 - P_3 + D_3) - d_j/p_j)$*

It follows from the lemma that, without loss of generality, we can assume $p_i \geq d_i$ for all i, since a source violating this condition should never be used.

We say that a source i is *profitable in a set S* if $i \in S$ and excluding the source from the set S would not increase the expected value of S. From the above lemma, this is the case if $\prod_{j \in S, j \neq i}(1-p_j) \geq d_i/p_i$. A set S of sources is *irreducible* if every element of S is profitable in S. Clearly, if S is irreducible, then $V(S_1) \leq V(S)$ for any subset $S_1 \subseteq S$. Every optimal single-batch schedule is irreducible.

We will use the following lemma in our discussion of k-batch schedules.

Lemma 3.4 *For any set of sources, an optimal serial schedule (including all sources in the set) sorts the sources in the nondecreasing order of their cost to success probability ratios.*

3.2.1 Single-Batch schedules

In this subsection we consider schedules that send out all their queries in a single batch, *i.e.* all queries are performed in parallel at time $t = 0$. We present an algorithm that approximates the optimal single-batch schedule with ratio $1/2$, then develop a PTAS. Recall that a single-batch schedule $\mathcal{P}$ is just a set of sources, and our goal is to maximize $V(\mathcal{P}) = (1 - \prod_{i \in \mathcal{P}}(1-p_i)) - \sum_{i \in \mathcal{P}} d_i$.

A Ratio $\frac{1}{2}$ Approximation Algorithm Our algorithm, Pick-a-Star, is somewhat similar to the greedy approximation algorithm for Knapsack given in [10], though the analysis of its performance is more complex. Pick-a-Star sorts the sources in ascending order of the ratio d_i/p_i. It then goes over each source i, picks it and then picks the rest from the sorted list (with i removed) until it reaches a source j such that the stopping criterion $\prod_{k=i \text{ or } k<j}(1-p_k) \leq d_j/p_j$ is satisfied. Lemma 3.3 explains the choice of the criterion. Thus a schedule is generated for each source i, and Pick-a-Star keeps track of the schedule with the highest expected value over the iterations. Clearly the running time is $O(n^2)$.

Now we analyze the performance of Pick-a-Star and show that it results in an expected value that is at least half of the optimum. Let APPR be the schedule obtained by Pick-a-Star and OPT an optimal single-batch schedule. Without loss of generality, we may assume $|\text{APPR}| > 1$. Moreover, we will consider henceforth the iteration where the first source picked by Pick-a-Star is the "most profitable" source in OPT, *i.e.* some source i with the maximum $V(\{i\})$ over all sources in OPT.

Define $S_0 = \text{APPR} \cap \text{OPT}$, $S_1 = \text{APPR} - S_0$, and $S_2 = \text{OPT} - S_0$. For each $i = 0, 1, 2$, let D_i and P_i be the collective cost and success probability of the sources in S_i. Observe that

$$\forall i \in S_1 \forall j \in S_2, d_i/p_i \leq d_j/p_j \tag{1}$$

Let us first consider the (easier) case in which $S_2 = \emptyset$. Let *last* be the last source picked by Pick-a-Star. Observe that $S_1 \subseteq \{1, \ldots, last\}$. Since the collective failure probability of $\text{APPR} - \{last\}$ is greater than $d_{last}/p_{last} \geq \cdots \geq d_1/p_1$, every element of S_1 is profitable in the set $\text{APPR} - \{last\}$. By Lemma 3.3, $V(\text{APPR} - \{last\}) \geq V(\text{OPT} - \{last\})$. We also know that $V(\text{APPR}) \geq V(\text{APPR} - \{last\})$ by Lemma 3.3. Since $V(\text{APPR}) \geq V(\{last\})$ and $V(\text{OPT}) \leq V(\text{OPT} - \{last\}) + V(\{last\})$ by Lemma 3.2,

$$2V(\text{APPR}) \geq V(\text{OPT} - \{last\}) + V(\{last\}) \geq V(\text{OPT}).$$

Now suppose that $S_2 \neq \emptyset$. Since OPT is irreducible, $S_1 \neq \emptyset$. We can assume that the collective failure probability of APPR is at least the ratio d_{last}/p_{last}, because otherwise we could modify APPR by decreasing p_{last} while keeping d_{last}/p_{last} constant until the collective failure probability of APPR becomes equal to d_{last}/p_{last}. This is possible since the collective failure probability of $\text{APPR} - \{last\}$ is greater than d_{last}/p_{last}. By Lemma 3.3, such modification could only worsen the expected value of APPR. Note that we may assume that source *last* is not in OPT, since otherwise we can replicate *last* and perform the modification on the replicated source. The replicated source cannot be in OPT, since OPT does not contain other sources of S_1 with lower ratios. Note also that this modification does not affect the first source picked by Pick-a-Star since $|\text{APPR}| > 1$. Let $m = |S_2|$ and $l = |S_1|$. Let

$$\alpha_1 = \max_{i \in S_1} \frac{d_i}{p_i(1-P_0)} \leq \frac{d_{last}}{p_{last}(1-P_0)}$$
$$\alpha_2 = \min_{i \in S_2} \frac{d_i}{p_i(1-P_0)}$$

By relation 1, clearly $\alpha_1 \leq \alpha_2$. The next lemma relating α_1, α_2 to P_1, P_2 is a key to our analysis.

Lemma 3.5 *(i)* $\alpha_1 \leq 1 - P_1 \leq \alpha_2$ *and (ii)* $1 - P_2 > \alpha_2^{m/(m-1)}$.

Now we want to find a lower bound for the ratio

$$\frac{V(\text{APPR})}{V(\text{OPT})} = \frac{P_0 - D_0 + (1-P_0)P_1 - D_1}{P_0 - D_0 + (1-P_0)P_2 - D_2} \tag{2}$$

Since $V(S_0) \geq V(S_2)/m$ by the choice of the first source picked by Pick-a-Star and the fact that S_0 is irreducible,

$$V(\text{OPT}) \leq V(S_0) + V(S_2) \leq (m+1)V(S_0).$$

This implies

$$\frac{(1-P_0)P_2 - D_2}{P_0 - D_0 + (1-P_0)P_2 - D_2} \leq \frac{m}{m+1}.$$

Define $r = \frac{(1-P_0)P_1 - D_1}{(1-P_0)P_2 - D_2}$. To obtain a lower bound of $1/2$ for the ratio in equality 2, we need

$$\frac{1}{m+1} + r\frac{m}{m+1} \geq \frac{1}{2}, \quad i.e. \;\; r \geq \frac{m-1}{2m}. \tag{3}$$

The next lemma, whose proof uses Lemma 3.5, gives a clean lower bound for ratio r.

Lemma 3.6

$$r \geq \min_{\alpha_1 \leq 1-P_1 \leq \alpha_2} \frac{P_1 - (l(1-(1-P_1)^{1/l})\alpha_1}{(1-\alpha_2^{m/(m-1)})(1-\alpha_2)}.$$

By simplifying the above lower bound function for ratio r, we obtain the main theorem.

Theorem 3.7 *Pick-a-Star achieves an expected value that is at least half of the optimum value.*

Extending Pick-a-Star to a PTAS The extension of the algorithm is straightforward. Let $r \geq 1$ be any fixed constant. The new algorithm iterates over all possible choices of at most r sources and schedules the rest of the sources based on the cost to success probability ratio, using the same stopping criterion. It then outputs the best schedule found in all iterations. Call the new algorithm Pick-r-Stars. Clearly, it runs in $O(n^{r+1})$ time. We show that Pick-r-Stars achieves an approximation ratio of $\frac{r-1}{r+1}$. The analysis is different from the previous subsection in that we will make use of the r sources in the optimal schedule with the highest success probability instead of the the most profitable ones.

Let APPR be the schedule found by Pick-r-Stars and OPT an optimal schedule. We assume without loss of generality that (i) APPR contains the r sources in OPT with the highest success probability, and (ii) the collective failure probability of APPR is at least the ratio d_{last}/p_{last}, where *last* is the last source picked by Pick-r-Stars. Let $S_0 = \text{APPR} \cap \text{OPT}$, $S_1 = \text{APPR} - S_0$, and $S_2 = \text{OPT} - S_0$ and the corresponding collective costs and success probabilities D_i and P_i, for each $i = 0, 1, 2$. We also have $d_i/p_i \leq d_j/p_j$ for all $i \in S_1, j \in S_2$. Define $l = |S_1|$, $m = |S_2|$, and

$$\alpha_0 = \max_{i \in S_0} \frac{d_i}{p_i}; \quad \alpha_1 = \max_{i \in S_1} \frac{d_i}{p_i} \leq \frac{d_{last}}{p_{last}(1-P_0)}$$

After making some more simplifications, we derive the following clean formulas for the expected values:

$$\begin{aligned} V(\text{APPR}) &= 1-(1-p)^r \frac{\alpha_1}{(1-p)^r} - \alpha_0 rp \\ &- l(1-(\frac{\alpha_1}{(1-p)^r})^{1/l}) \\ V(\text{OPT}) &= 1-(1-p)^{r+m} - \alpha_0 rp - \alpha_1 mp \end{aligned}$$

By further simplifying the formulas and a lot of careful mathematical manipulations, we obtain the next main theorem.

Theorem 3.8 *Pick-r-Stars produces a single-batch schedule with an expected value that is at least* $(r-1)/(r+1)$ *of the optimum.*

3.2.2 Approximating Optimal k-Batch Schedules

We present an algorithm, Back-and-Forth, that approximates optimal k-batch schedules with a constant ratio $1/6$. Back-and-Forth works in two phases. In the first phase, it greedily constructs a schedule batch by batch, starting from the last batch and going backward. For each batch, it invokes the single-batch algorithm Pick-a-Star, but with a modified stopping criterion derived from Lemma 3.3. In the second phase, the algorithm splits the schedule obtained in the first phase into three k-batch schedules: one is a schedule obtained by taking the first source picked in each batch and arranging these sources in an optimal serial order; one is a schedule obtained by taking the last source picked in each batch and arranging these sources in an optimal serial order; and the third consists of the rest of the sources but with the batch ordering completely *reversed*. It then compares these three schedules with the original one and returns the best of the four.

For any schedule $\mathcal{P}$, $\mathcal{P}^R$ denotes the schedule obtained by reversing the batches. We will also use set operations on k-batch schedules when there is no ambiguity. Back-and-Forth is illustrated in Figure 2. Clearly Back-and-Forth can be implemented to run in time $O(kn^2)$.

The analysis of Back-and-Forth uses Theorem 3.7. The difficulty here is that because the sources can be scheduled in different batches, some batches of an optimal k-batch schedule could be better individually than their counterparts in APPR by an arbitrarily large factor. To get around this, we relate a k-batch schedule to its optimally serialized version. For any schedule $\mathcal{P}$, let $\overline{\mathcal{P}}$ denote the serial schedule obtained by scheduling the sources in $\mathcal{P}$ in an optimal order (i.e. in decreasing order of d/p). In general $V(\overline{\mathcal{P}})$ is better than $V(\mathcal{P})$ and could be arbitrarily better than $V(\mathcal{P})$. Before we give the complete analysis, we observe the following useful facts:

Corollary 3.9 *Let* S_1 *and* S_2 *be two sets and* $S_1 \subseteq S_2$. *Then* $V(\overline{S_1}) \leq V(\overline{S_2})$.

The following lemma, which is somewhat surprising, is a key to our analysis.

Lemma 3.10 *For any irreducible set* S *of sources,* $V(S) \geq V(\overline{S})/2$.

The next corollary follows from the above lemma and Lemma 3.4.

Corollary 3.11 *Let* $\mathcal{P}$ *be a* k*-batch schedule consisting of batches* $S_1, \ldots, S_k$. *Suppose that (i) each* S_i *is irreducible and (ii) for any* $s_l \in S_i$ *and* $s_m \in S_j$, *where* $i < j$, $c_l/p_l \leq c_m/p_m$. *Then* $V(\mathcal{P}) \geq V(\overline{\mathcal{P}})/2$.

Now we analyze the performance of Back-and-Forth. Denote the optimal schedule as OPT, and partition OPT as $\text{OPT}_1 = \text{APPR}_0 \cap \text{OPT}$ and $\text{OPT}_2 = \text{OPT} - \text{OPT}_1$, where the sources in OPT_1 and OPT_2 are scheduled in the same batches as they are in OPT. By Lemma 3.2,

$$V(\text{OPT}) \leq V(\text{OPT}_1) + V(\text{OPT}_2).$$

We compare the performances of OPT_1 and OPT_2 with that of APPR separately. The proof of the following lemma uses Lemma 3.3 and Theorem 3.7.

Lemma 3.12 $V(\text{OPT}_2) \leq 2V(\text{APPR})$.

The proof of the next lemma uses Lemma 3.2 and Corollaries 3.9 and 3.11.

Lemma 3.13 $V(\text{OPT}_1) \leq 4V(\text{APPR})$.

Lemmas 3.12 and 3.13 together give the following theorem.

Theorem 3.14 *Algorithm Back-and-Forth returns a* k*-batch schedule with an expected value at least* $1/6$ *of the optimum.*

3.3 Approximation Algorithms for the Cost Threshold Models

Optimal schedules in the cost-threshold models **TL** and **TT** are much easier to approximate. We first present an FPTAS for model **TL** under a weak assumption: $p_i - t_i \geq 0$ for every source i, *i.e.* every source considered is profitable by itself. The extension to model **TT** (with no restriction) is

1. Sort the sources so that $c_1/p_1 \leq \cdots \leq c_n/p_n$.
(* Phase I *)
2. $\mathrm{APPR}_0 = \emptyset$. (* APPR_0 denotes a k-batch schedule. *)
3. For $i := k$ downto 1
4. $S = \emptyset$. (* S is the best i-th batch found so far. *)
5. For $j := 1$ to n, where $s_j \notin \mathrm{APPR}_0$
6. $S_1 := \{s_j\}$.
7. $Q := 1 - p_j$. (* Q is the collective failure probability of S_1. *)
8. For $l := 1$ to n, where $l \neq j$ and $s_l \notin \mathrm{APPR}_0$
9. If $Q(1 - V(APPR_0)) > c_l/p_l$ then
10. $S_1 := S_1 \cup \{s_l\}$; $Q := Q(1 - p_l)$.
11. else exit to step 13.
12. If $V(S) < V(S_1)$ then $S := S_1$.
13. Add S to APPR_0 as the i-th batch.
14. Record the first and last sources picked for S.
(* Phase II *)
15. Let $APPR_1$ and APPR_2 be the optimal serial schedules consisting of the first and last sources picked in Phase I for each batch, resp.
16 $\mathrm{APPR}_3 := (\mathrm{APPR}_0 - \{\mathrm{APPR}_1 \cup \mathrm{APPR}_2\})^R$.
17 Output schedule APPR as the best of $\mathrm{APPR}_0, \mathrm{APPR}_1, \mathrm{APPR}_2, \mathrm{APPR}_3$.

Figure 2: The algorithm Back-and-Forth.

straightforward. The main idea is the rounding technique introduced in [6] for Knapsack. It is easy to see that, in model **TL**, an optimal schedule should be in fact a single-batch schedule. Let $\mathcal{P} = \{i_1, \ldots, i_m\}$ be a single-batch schedule, where $t_{i_1} \leq \cdots \leq t_{i_m}$. Then,

$$V(\mathcal{P}) = \sum_{j=1}^{m-1} \prod_{l=1}^{j-1} (1 - p_{i_l}) p_{i_j} (1 - t_{i_j}) + \prod_{j=1}^{m-1} (1 - p_{i_j})(p_{i_m} - t_{i_m})$$

Since $p_i - t_i \geq 0$ by our assumption, every term is nonnegative in equation 4, and we can round each $p_{i_m} - t_{i_m}$, $p_{i_j}(1 - t_{i_j})$ and $\log(1 - p_{i_j})$, and then use dynamic programming to obtain an FPTAS.

Theorem 3.15 *Assume that $p_i - t_i \geq 0$ for every source i. There is an FPTAS for the problem of computing optimal schedules in model* **TL**.

Corollary 3.16 *There is an FPTAS for the problem of computing optimal schedules in model* **TT**.

References

[1] O. Etzioni, S. Hanks, T. Jiang and O. Madani. Optimal Information Gathering on the Internet with Time and Cost Constraints. Manuscript 1996.

[2] O. Etzioni, R. M. Karp, and O. Waarts. Efficient Access to Information Sources on the Internet. Manuscript 1996.

[3] P. Feigin and G. Harel. Minimizing costs of personnel testing programs. *Naval Research Logistics Quarterly* 29, 87-95, 1982.

[4] M. Garey. Optimal task scheduling with precedence constraints. *Discrete Mathematics*, 4, 37-56 (1973).

[5] M. Henig and D. Simchi-Levy. Scheduling tasks with failure probabilities to minimize expected cost. *Naval Research Logistics* 37,99-109, 1990.

[6] O. Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subsets problems. *Journal of the ACM* 22, 463-368, 1975.

[7] B. Krulwich. The BargainFinder agent: Comparison price shopping on the Internet. *Bots and Other Internet Beasties.* 1996.

[8] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. *Designing Decision Support Systems Notes NFL 11.89/03,* Eindhoven University of Technology, 1989.

[9] New York Times, June 7, 1992

[10] S. Sahni. Approximation algorithms for the 0/1-knapsack problem. *Journal of the ACM* 22, 115-124, 1975.

[11] Secure Transaction Technology. In `http://www.visa.com/cgi-bin/vee/sf/set/intro.html`.

[12] E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. *Proc. 4th World Wide Web Conf.*, 195-208, Boston, MA, 1995.

[13] H. Simon and J. Kadane. Optimal problem-solving search: all-or-none solutions. *Artificial Intelligence* 6, 235-247, 1975.

[14] M. Sirbu, and J.D. Tygar. NetBill: An Internet Commerce System Optimized for Network Delivered Services. Manuscript 1995. To appear in *IEEE CompCon Conference.*