Scheduling Cooperative Emergency Response (or how the Meek shall overcome the Greedy)

Carol Dottin
Department of Math.& Computer Science
John Jay College, CUNY
New York, 10016
cdottin@jjay.cuny.edu

Bilal Khan
Department of Math.& Computer Science
John Jay College, CUNY
New York, 10016
bkhan@jjay.cuny.edu

ABSTRACT

We consider the problem of scheduling emergency responders to geospatially located finite duration temporally bounded tasks. We consider two different schedulers, Greedy and Meek. schedulers: the Greedy algorithm schedules the closest available qualified emergency responder to a task, while the Meek algorithm, assigns the qualified emergency responder that minimizes the expected future cost of the Greedy algorithm. We show that to be effective emergency response scheduling must take into consideration future costs (as the Meek algorithm does), and not merely instantaneous costs (as the Greedy algorithm does).

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design–*wireless communication*

General Terms

Algorithms, Performance

Keywords

Ad hoc networks, cooperative mobility

1. INTRODUCTION

In the event of natural or man-made disasters, public emergency response services play a critical role in saving lives and reducing property losses. In such crisis situations, each emergency response team member must fuse data from their immediate vicinity with relevant information obtained from other rescuers and data warehouses, in order to maintain situational awareness to execute effective actions which would lead to mission objectives. This paper concerns the design and development of components of a system that is capable of efficiently handling the tasking requirements of emergency first responders in disaster-recovery settings.

IWCMC '09 , June 21-24, 2009, Leipzig, GERMANY

2. BACKGROUND

Timing, execution and allocation of resources become even more important during emergencies. Disaster response and recovery efforts require timely interaction and coordination of all resources in order to be effective [2]. Recent research efforts have sought to address the challenge of allocating resources in disasters, in a manner that is both efficient and effective. Some researchers have identified the data recovery scheduling problem as an optimization problem, in which the goal is to minimize the financial penalties due to downtime, data loss [4] and vulnerability to subsequent failures [1]. They have presented several methods for finding optimal or near-optimal solutions, including priority-based, randomized and genetic algorithm-guided ad hoc heuristics [1], as well as strategy for mapping a network using a collection of cooperating mobile agents [3].

3. PROBLEM FORMULATION

The system to be designed will be presented with a sequence of tasks and must manage the allocation of resources (a set of first responders) effectively towards the fulfilment of these tasks – to the extent that such an objective is attainable. In what follows, we will identify each first responder with the responder's mobile phone. This is not an unreasonable supposition, since we assume that the mobile phone is the only reliable way for the system to exchange information with the emergency response personnel. Each task has a duration and a window of time during which the task needs to be performed. Clearly, the window is always greater than or equal to the duration. Each task in general will require one or more capabilities (on the part of the responder) in order for the task to be executed.

This system is part of a larger research initiative at John Jay College of Criminal Justice (City University of New York), known as the Cooperative Networking for Emergency Response Teams (CONCERT) Project, whose objective is to develop a distributed mobile phone application that addresses the problem of context and interest-based data acquisition and delivery for teams of disaster recovery responders using effective algorithms for optimal agent scheduling. The system may choose to reject a task (e.g. because it determines that it does not have any qualified available responders to perform it). If the system rejects the task, then the submitter is free to seek alternate means to meet his objectives (i.e. using resources outside the purview of the CONCERT system). The system must however, provide quality of service (QoS) assurance in the following sense: *If the system accepts a task, then the task must indeed be*

fulfilled.

In order for a task to be fulfilled it must be assigned to a responder (i.e. a phone) that is both available and that has all the capabilities necessary for the execution of the task. If a task is accepted, the system must (at the time of acceptance) assign the task to a specific phone, and specify the precise subinterval of the task window during which to perform the task.

For any specific sequence of tasks, the most natural measure of a task assignment algorithm is the percentage of tasks that the scheduler successfully assigns. To further differentiate between schedulers that have comparable task acceptance rates, we will consider the distances moved by the first responders (i.e phones) in the process of fulfilling their assigned tasks. Minimizing this latter quantity is a reasonable secondary objective since it reflects a real systemic cost, both in terms of time and energy. We now make this rigorous.

3.1 Input

The System's Resources consist of a set of phones $\mathbb{P} = \{1, \cdots, n\}$ where each phone $i \in \mathbb{P}$ is specified by: a set of capabilities $C_i \subseteq \mathbb{C}$, where $\mathbb{C} = \{1, \cdots, m\}$, an initial location, given by $x_i(0), y_i(0)$, where $-180^\circ \leq x_i(0) \leq 180^\circ$ and $-90^\circ \leq y_i(0) \leq 90^\circ$, a maximum speed $v_i > 0$.

While resources are fixed at the outset, the System's Tasks arrive in an "online" manner as a sequence of k tasks $\mathbb{T} = (T_1, \cdots, T_k)$, where each task T_j is specified by a location (lat_j, lon_j), a time window $(start_j, end_j)$, a time duration d_j, a set of requirements $R_j \subseteq \mathbb{C}$, where $\mathbb{C} = \{1, \cdots, m\}$, and an arrival time a_j. The earliest time a task can be started is denoted $start_j$ and the time by which the task must be completed (if the task is accepted) is denoted end_j. The amount of time the task will take to be executed is denoted d_j. The subset of capabilities which the chosen phone must possess in order to execute the task is denoted $R_j \subseteq \mathbb{C}$. The time at which the task is presented to the system (for assignment or rejection) is denoted a_j.

In order for a task T_j to be a **syntactically valid** part of sequence $\mathbb{T}$ it must satisfy these constraints: Arrival time precedes start time: $a_j \leq start_j$, duration cannot exceed the window: $start_j + d_j \leq end_j$, duration must be non-negative: $0 \leq d_j$, tasks arrive sequentially: $a_j \leq a_{j+1}$.

3.2 Algorithm Correctness

Given the resources $\mathbb{P}$ and task sequence $\mathbb{T}$, we seek to design algorithms for scheduling phones (resources) to tasks. Consider a hypothetical algorithm $\mathbb{N}$ presented with a sequence of syntactically valid tasks, $\mathbb{T} = (T_1, \cdots, T_j, \cdots, T_k)$, one at a time. Upon receiving T_j at time a_j, algorithm $\mathbb{N}$ decides to **Accept** T_j *or* **Reject** T_j.

If it chooses to accept then it must assign task T_j to some phone, denoted $A_{\mathbb{N}}(T_j) \in \mathbb{P}$ and simultaneously specify the **execution interval**, $[\varepsilon_{\mathbb{N}}(T_j), \varepsilon_{\mathbb{N}}(T_j) + d_j]$ during which the task T_j will actually be executed by phone $A_{\mathbb{N}}(T_j)$. If, however, algorithm $\mathbb{N}$ chooses to reject the task, then phone $A_{\mathbb{N}}(T_j)$ and $\varepsilon_{\mathbb{N}}$ are both undefined and set to a sentinel value of *"reject"*. Formally then, algorithm $\mathbb{N}$'s operation on a syntactically valid sequence of tasks $\mathbb{T}$ gives rise to two implicit maps: $A_{\mathbb{N}} : \mathbb{T} \to \mathbb{P} \cup \{reject\}$, and $\varepsilon_{\mathbb{N}} : \mathbb{T} \to \mathbb{R}^{\geq 0} \cup \{reject\}$.

An algorithm's assignment $(A_{\mathbb{N}}, \varepsilon_{\mathbb{N}})$ is said to be a **feasible solution** for a sequence of tasks $\mathbb{T}$ given resources $\mathbb{P}$, if the following five conditions hold:

Feasibility Criteria (1) $A_{\mathbb{N}}$ and $\varepsilon_{\mathbb{N}}$ are mutually consistent, i.e. $A_{\mathbb{N}}$ rejects if and only if $\varepsilon_{\mathbb{N}}$ rejects. (2) The phone (responder) possesses the necessary capabilities to perform the task. (3) The execution interval of the task is within the window defined for the task. (4) No phone is requested to do more than one task at any given time. (5) No phone must be asked to move faster than its maximum speed. To make this mathematically precise, some intermediate definitions are needed.

3.3 Performance Measures

Suppose algorithm $\mathbb{N}$, given resources $\mathbb{P}$ and a syntactically valid task sequence $\mathbb{T}$ acts via feasible maps $A_{\mathbb{N}}, \varepsilon_{\mathbb{N}}$. To evaluate the effectiveness of an algorithm, we consider the set $\Gamma_{\mathbb{N}}(\mathbb{T}) \stackrel{\text{def}}{=} \{T_j \in \mathbb{T} \mid A_{\mathbb{N}}(T_j) \neq rejected\}$ of accepted tasks, and use this to define performance measures: (I) *acceptance rate* of $\mathbb{N}$ on $\mathbb{T}$ is defined as: $\gamma_{\mathbb{N}}(\mathbb{T}) \stackrel{\text{def}}{=} \frac{|\Gamma_{\mathbb{N}}(\mathbb{T})|}{|\mathbb{T}|}$ where the numerator is the cardinality of the set of tasks which were accepted, and the denominator is the cardinality of the set of all tasks, and (II) *average movement cost* of $\mathbb{N}$ on $\mathbb{T}$ is $\beta_{\mathbb{N}}(\mathbb{T}) \stackrel{\text{def}}{=} \frac{1}{|\mathbb{T}| \cdot \gamma_{\mathbb{N}}(\mathbb{T})} \sum_{T_j \in \Gamma_{\mathbb{N}}(\mathbb{T})} d((lat_j, lon_j), prevloc(T_j))$ where $prevloc(T_j)$ is the location of the last task completed by phone $A_{\mathbb{N}}(T_j)$, prior to its starting task T_j. Thus, the numerator is the total distance moved to fulfil each accepted task, and the denominator is the cardinality of the set of all accepted tasks.

4. PROPOSED ALGORITHMS

We now proceed to describe the operation of three concrete algorithms: Random, Greedy, and Meek, all of which produce feasible solutions. Then we will use simulations to evaluate the three algorithms and present and interpret the outcomes of these experiments.

Random Algorithm. The Random algorithm considers a submitted task T_j and seeks to find any available phone i that is capable of fulfilling T_j via an assignment that fulfils the feasibility criteria 1-5 described in Section 2. If more than one phone fulfils the criteria it chooses one of the phones arbitrarily.

Greedy Algorithm. The behavior of the Greedy algorithm differs from the Random algorithm, only in the case where the set of phones meeting the feasibility criteria $f_{2-5}(T_j)$ has cardinality greater than one. In this case, the Greedy algorithm assigns the task to that phone $i \in f_{2-5}(T_j)$ that would have to move the shortest distance in order to execute the task. In practice, this minimization requires the Greedy algorithm to consider every phone in $f_{2-5}(T_j)$.

Meek Algorithm. While the Greedy and Random algorithms do not consider future implications of their assignment decisions, the Meek algorithm is obsessed with the future impact of its present decisions. In this regard, it must make certain assumptions about the nature of tasks it anticipates seeing in the future. We assume that future tasks are determined by a **probabilistic non-malicious adversary**, which creates tasks a fixed interval in advance of their starting time, distributed uniformly random in space, and requiring a randomly cardinality τ subset of capabilities. The window of a task is a fixed size W, and the duration of the task is always a constant *slack* ϵ times W, $0 \leq \epsilon \leq 1$.

Given these assumptions about the behavior of the probabilistic adversary's construction of tasks, the Meek algo-

rithm operates as follows: When presented with a new task T_j, it computes the *expected cost* in fulfilling the probabilistic adversary's hypothetical future task T_{j+1} under the assumption $A_{Meek}(T_j) = i$, for each feasible assignment $i \in f_{2-5}(T_j)$. It then makes the assignment of the current task T_j to the phone that minimizes the expected cost of fulfilling the future task. It should be noted that in computing the expected cost of fulfilling the future task T_{j+1}, the Meek algorithm assumes that the future task is going to be fulfilled by the Greedy algorithm, even though (somewhat paradoxically) when the future task T_{j+1} actually arrives, it will be processed using the Meek algorithm (in the same manner as T_j was) and not by the Greedy algorithm. There is no way however to avoid this reference to the Greedy algorithm, without leading to a self-referential definition of the Meek algorithm.

We make this assignment algorithm more explicit: Given a task T_j, the Meek algorithm acts as follows:

1: Find all phones in $f_{2-5}(T_j)$.

2: For each phone $i \in f_{2-5}(T_j)$ compute the expected cost of using the Greedy algorithm to fulfil the future task T_{j+1} generated by the probabilistic adversary, as follows: (a) For each possible location (x, y) and each possible cardinality τ subset of capabilities z, compute the cost of fulfilling a hypothetical task at location (x, y) requiring capabilities z using the Greedy algorithm, under the assumption that T_j is assigned to phone i. (b) Compute the average value of the set of costs and call this the expected future cost if T_j is assigned to phone i, denoted $E[fc(j, i)]$.

3: Choose phone i for which $E[fc(j, i)]$ is minimal.

Note that the actual computation in steps 2a and 2b must in practice be an approximation of the expected value of the future cost. It is implemented as "For each location (x, y), taken from a finite set L of candidate placements of the future task T_{j+1}..." In this work, the finite set of candidate placements $L \subset [-180°, 180°] \times [-90°, 90°]$ is taken to be a Cartesian lattice of uniformly spaced points at intervals of 1 degree spacing. Since it is not the absolute values of the expected future costs that are important but only their relative values that are used to assess the merits of different assignments, the approximation introduced by the Riemann sum is not a cause for concern.

5. COMPLEXITY OF TASK ASSIGNMENT

Random and Greedy Algorithms. The Random algorithm searches for phones meeting feasibility criteria (2) which requires time $O(|R_j| \sum_{i=1}^{n} \log |C_i|)$. We denote the resulting set of phones capable of performing task T_j as $f_2(T_j)$. The average size of $f_2(T_j)$ may be estimated combinatorially. If each phone i has χ capabilities randomly from the set of all m capabilities, and each task T_j has $\tau \leq \chi$ requirements, the expected complexity of search is $O(\tau n \log \chi)$ to determine the phones satisfying criteria 2-5, which we denote $f_{2-5}(T_j)$. Since tasks are of finite duration, we extend the previous probabilistic formulation by assuming that on average δ tasks are active in the system at any given time, making the *expected time complexity* of task assignment $O(\delta + \tau n \log \chi)$.

The Meek Algorithm. The time complexity of the Meek algorithm requires simulating the Greedy algorithm on each of the $|L|$ possible placements of the hypothetical future task T_{j+1}. Unraveling the above three steps, we see that this takes expected time $O(n(\tau/m)^{\tau} \cdot |L|(me/\tau)^{\tau} \cdot (\delta + \tau n \log \chi))$, which implies the expected cost of assigning a task is $O(n|L|e^{\tau}(\delta + \tau n \log \chi))$ making the Meek algorithm require a factor of $n|L|e^{\tau}$ longer to assign a task. In the next section, we quantify the benefits of this additional overhead in terms of the algorithm's performance.

6. EXPERIMENTS

To better evaluate the effectiveness of the three algorithms described in Section 3, we analyze several key input parameters over a range of values using the simulation framework. Unless specified otherwise, each phone $i \in \mathbb{P}$ has $|C_i| = 1$ and $\upsilon_i = 1.0 \div 3600$, while each task T_i was taken to have the following *default specification*: $a_i = 3600 \cdot i$, $start_i = N(a_i + L, 1hr)$, *where* L *is* (24×3600), $end_i = N(start_i + W, 1hr)$, *where* W *is* (4×3600), *slack* $(\epsilon) = 0.8$, $|R_i| = 1$. where $N(\mu, \sigma)$ denotes a normal distribution with mean μ and standard deviation σ. Each data point represents the mean of 15 trials, in which each trial consists of a sequence of tasks spanning 7 days. Standard deviations of measurements are depicted by vertical error bars.

6.1 Capabilities

In the first set of experiments, we considered the influence of the size of the set of capabilities $\mathbb{C}$ from which phones and tasks draw their capabilities and requirements. We varied $|\mathbb{C}|$ by allowing m to range from 1 to 20, while the set of phones $\mathbb{P}$ was fixed to be of size $n = 2$.

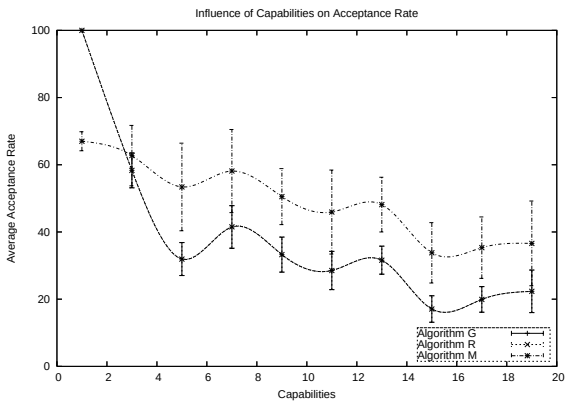

Figure 1: Acceptance Rate vs Capabilities

The performance of the algorithms is shown in Figures 1 and 2. The general trend is that acceptance rate decreases as the size of the capabilities set increases. This is plausible since a typical phone is less likely to fulfil the requirements of a typical task as $|\mathbb{C}|$ increases. Performance drops more sharply for the Greedy algorithm, compared to the Meek algorithm; the latter also consistently succeeds in assigning a greater fraction of tasks. The average movement cost incurred per task by the Meek algorithm is approximately 50% of the cost incurred by Greedy and Random.

6.2 Task Arrival Time

In this set of experiments, the inter-task time Δ was varied, by taking task i to arrive at time $a_i = \Delta \cdot i$. The set of

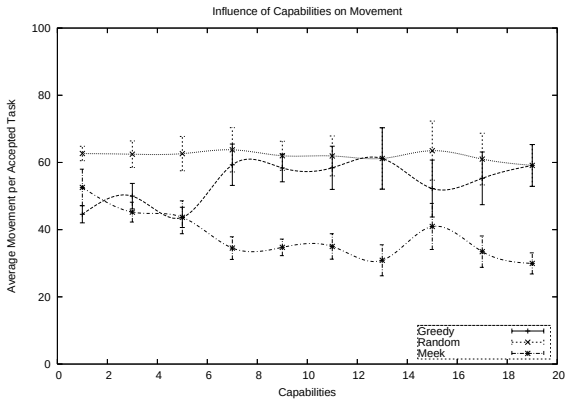

Figure 2: Movement Cost vs Capabilities

capabilities was fixed to be of size $m = |\mathbb{C}| = 2$, and the set of phones $\mathbb{P}$ was fixed to be of size $n = 2$.

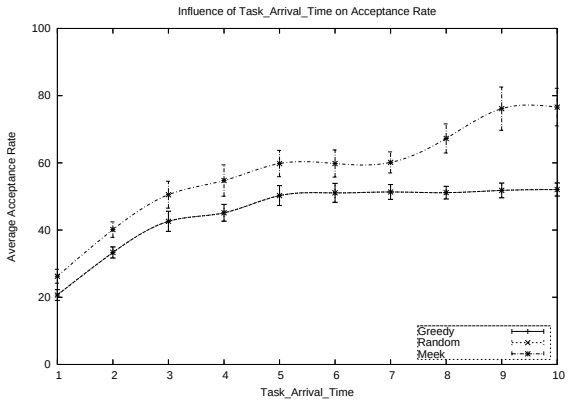

Figure 3: Acceptance Rate vs Arrival Times

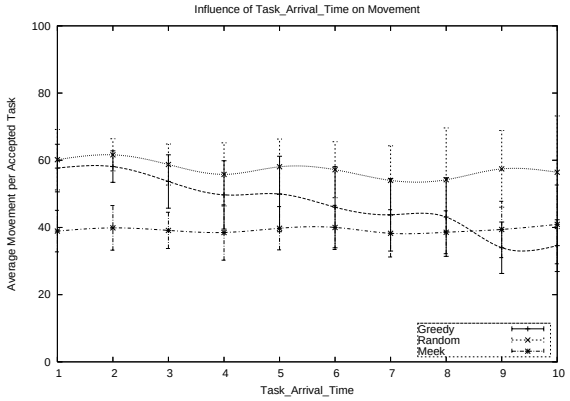

Figure 4: Movement Cost vs Arrival Times

As the time between arriving tasks increases, the rate at which tasks are accepted and successfully assigned resources increases. This is plausible, since contention for resources is reduced when fewer tasks are active at any given time, as is the case when inter-task arrival time is high. We note that the Meek algorithm exhibits a 20-30% higher acceptance rate than Greedy and Random. An analysis of the movement cost of each algorithm against changes in the arrival times of tasks (Figure 4) shows a different trend for each algorithm.

The Greedy algorithm exhibited a downward trend in movement cost as the time between arriving tasks increased. This can be further explained by the fact that at higher inter-task arrival times, all phones available for assignment by the algorithm have already completed their tasks, since the duration of any tasks is now less than the time when new tasks arrive; thus the algorithm has its full complement of resources (i.e phones) available, so that the closest phone is always available. The Meek algorithm once again exhibits a fairly constant movement cost regardless of inter-task arrival time, exhibiting on average the lowest movement cost of all three algorithms.

6.3 Resources

In analyzing how resource availability influences performance, we varied $|\mathbb{P}|$ in the range of $1 \leq |\mathbb{P}| \leq 10$ and examined how increases in the number of responders available affected the acceptance rate and the average movement cost of the three algorithms. The other parameters to the simulator were set at $m = |\mathbb{C}| = 2$. Each phone $i \in \mathbb{P}$ has $|C_i| = 1$ and $v_i = 1.0 \div 3600$. Tasks were generated with default specifications.

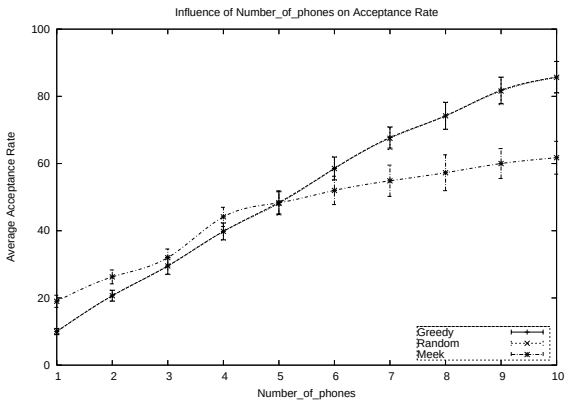

Figure 5: Acceptance Rate vs Resources

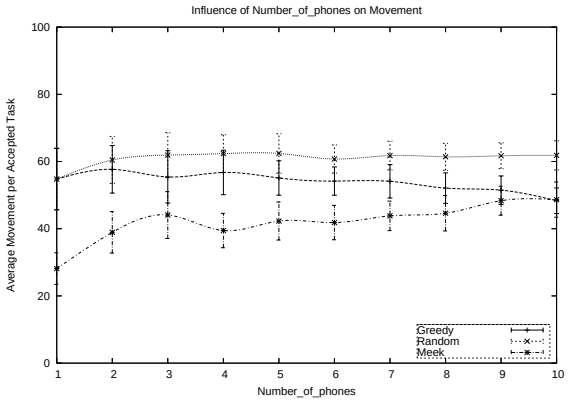

Figure 6: Movement vs Resources

The expected general trend that acceptance rate will increase as the number of phones increases is supported by the graph in Figure 5. A interesting point on the graph is where the number of phones available for assignment is 5. At that point all three Controllers accepted approximately 50% of the tasks presented to them. As the number of capabilities available was two and any phone had a 50% probability of having one of the 2 features and arriving tasks also had a 50% probability of having the required capability, we can conclude that of the 5 phones available, half met the feasibility criteria to execute any task that arrived. As the number of phones increased beyond this, the Meek algorithm was more conservative in its acceptance of tasks and thus had a lower acceptance rate than the Greedy algorithm. However, when resources were limited, the Meek algorithm was more effective at assigning tasks.

6.4 Task Duration

Since the duration of each task equals ϵW, to analyze the influence of task duration we considered how changes to slack (ϵ) affected the acceptance rate and the average movement cost of the three algorithms when varying ϵ in the range of $0.1 \leq \epsilon \leq 1.0$. The set of capabilities was taken as $m = |\mathbb{C}| = 2$, and the set of phones $\mathbb{P}$ was taken to be of size $n = 2$. Each phone $i \in \mathbb{P}$ has $|C_i| = 1$, $\upsilon_i = 1.0 \div 3600$. Each task T_i was constructed using the default specification.

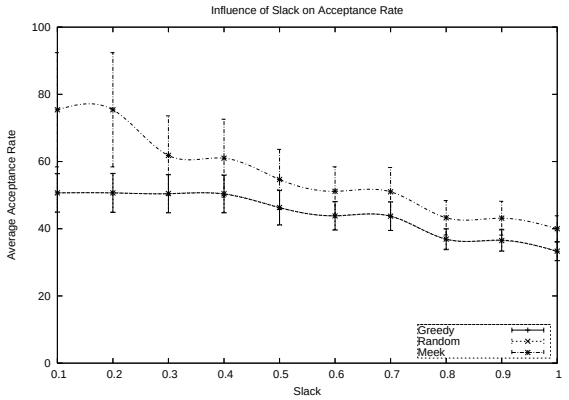

Figure 7: Acceptance Rate vs Slack

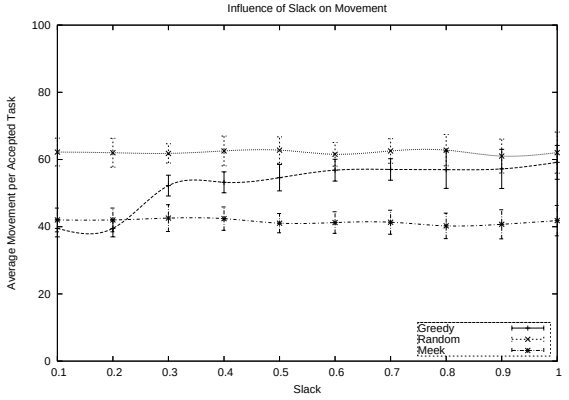

Figure 8: Movement Cost vs Slack

As ϵ increases, tasks take longer, resources are unavailable for longer periods of time, and so it is expected that the number of tasks accepted by each algorithm will decline (Figure 7). While all three algorithms show a decline in the acceptance rates, the Meek algorithm maintains a 10% higher acceptance rate compared to the Greedy algorithm. Once the duration of arriving tasks exceeds half the window of the task or greater (i.e. $\epsilon > 0.5$), the acceptance rate trends of the three algorithms coincide, though the Meek maintains a 10% higher acceptance rate compared to the greedy algorithm. The trend we expected for the average movement cost was that as the duration of tasks increased, the movement cost would also increase. However as shown in (Figure 8), the only algorithm that clearly exhibited this trend was the Greedy algorithm. The future awareness of the Meek algorithm's decisions made its performance scale favorable with respect to the ϵ parameter.

7. CONCLUSION

We considered the problem of scheduling emergency responders to geospatially located, finite duration, temporally bounded tasks, where both tasks and responders have a finite set of capabilities. We described two classes of strategies for scheduling emergency responders to tasks: the Greedy strategy schedules the closest available qualified emergency responder, while the Meek strategy assigns the qualified emergency responder that minimizes the expected future cost of the Greedy algorithm. We showed that the task acceptance rate for the Meek algorithm surpasses that of Greedy under almost all circumstances and yields lower per-task movement costs, and moreover, that the Meek algorithm's performance exhibits better scalability with respect to several key system parameters. Thus our results demonstrate that effective emergency response scheduling must take into consideration future costs (as the Meek algorithm does), and not merely instantaneous costs (as the Greedy algorithm does). The second author would like to acknowledge tha this research effort was partly supported by NSA grant H98230-07-1-0015.

8. REFERENCES

[1] Kimberly Keeton, Dirk Beyer, Ernesto Brau, Arif Merchant, Cipriano Santos, and Alex Zhang. On the road to recovery: restoring data after disasters. In *In Proceedings of the European Systems Conference*, pages 235–248. ACM Press, 2006.

[2] Andreas Meissner, Thomas Luckenbach, Thomas Risse, Thomas Kirste, and Holger Kirchner. Design challenges for an integrated disaster management communication and. In *Information SystemŤ, DIREN 2002 (co-located with IEEE INFOCOM 2002*, 2002.

[3] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. Cooperating mobile agents for mapping networks. In *Proceedings of the First Hungarian National Conference on Agent Based Computing*, pages 34–41, 1998.

[4] Nuno Neves and W. Kent Fuchs. Adaptive recovery for mobile environments. *Communications of the ACM*, 40:68–74, 1997.