Planning, Scheduling, and Deploying for Computational Ferrying

Sebastian A Zanlongo
and
Alexander C Wilson
and
Leonardo Bobadilla
and
Tamim Sookoor

Mobile Cyber-Physical Systems are expected to perform resource-intensive tasks that exceed their hardware (storage and computational power) capabilities. One traditional approach to solve this limitation is through cloud-based solutions. However, this approach may fail in scenarios with limited communication connectivity that can arise due to natural or adversarial conditions. One such situation is tactical battlefield missions where soldiers may require access to significant processing and storage capabilities for a task such as tracking or battlefield awareness without sacrificing their mobility capabilities. In this paper, we extend previous work on computational ferrying, where *Mobile High-Performance Computers (MHPCs)* can physically move the necessary hardware into the proximity of mobile units. Our extension proposes several improvements over state of the art. First, we present path planning algorithms to find reliable *a priori* estimation of distances between locations to obtain accurate scheduling. Second, we model MHPCs as autonomous vehicles which leads to more realistic scenarios in environments with obstacles. Third, we explicitly characterize the computational complexity of our proposed work. Finally, we implemented and tested in computer simulation all our algorithms to understand the effect of different obstacles, processors, number of MHPCs in completion time and deadlines met.

Keywords: Motion planning, Path Planning, Data Muling, Message Ferrying, Computational Ferrying, Autonomous Systems, Communication Challenged Environments

1. INTRODUCTION

Current mobile cyber-physical systems (CPS) are operating in complex, dynamic, and adversarial environments, yet they are expected to provide robust computing capabilities. Mobile devices may be required to execute tasks with potentially large storage and processing requirements, even when faced with limited processing power, battery life, and time [Kemp et al. 2012]. In spite of these limitations, the system must provide fast, secure service to as many users as possible. Conventional cloud computing approaches could provide a solution [Cuervo et al. 2010; Gordon et al. 2012]; however, several relevant applications are often located far from stable or secure high-capacity established communication infrastructure networks. Moreover, users may not be able to move towards the vicinity of traditional stationary cloudlets, instead of requiring the cloudlets to move towards the users.

The initial motivation for our ideas are battlefield scenarios where lightweight units might require to solve computational tasks with significant storage or processing requirements such as facial or object recognition and tracking. Without reliable high-bandwidth communication and far from stationary resources, the alternative of carrying the hardware needed to complete these tasks may not be feasible for many reasons, including but not limited to power, mobility, weight, and reliability. This issue has attracted the attention of the several researchers [Shires et al. 2012; Sookoor et al. 2013; Sookoor et al. 2014; Dawson and Doria 2015]. Other civilian applications such as rural and natural disaster areas have the same constraints. Another example of this

is Google's *Project Loon* [Carlson 2015] where high-altitude balloons moved around the globe providing Internet access to remote areas. Our work could provide more efficient coordination of these balloons or similar systems to maximize their efficacy.

This paper is an extension of the preliminary work presented in [Zanlongo et al. 2016]. This work improves and extends the conference submission in several aspects. We have broadened the related work and motivation, addressed in detail the computational complexity of the formulated computational ferrying problems, and improved the presentation of the algorithms. We also present new results and discussions related to the effects of removing tasks on deadlines missed and completion steps.

The rest of the paper is organized as follows. In Section III, we describe the system architecture, modeling of the problem, and describe the mathematical notation used in the rest of the paper. In Section IV, we discuss the computational complexity of the formulated problem, and we proposed methods for allocating tasks to each MHPC and scheduling the MHPC visit orders. We also present methods for generating and selecting trajectories between locations. In Section V, we present simulation results and a comparison against state of the art methods. Finally, in section Section VI, we finish with a discussion of the findings, conclusions, and potential avenues for future work.

2. RELATED WORK

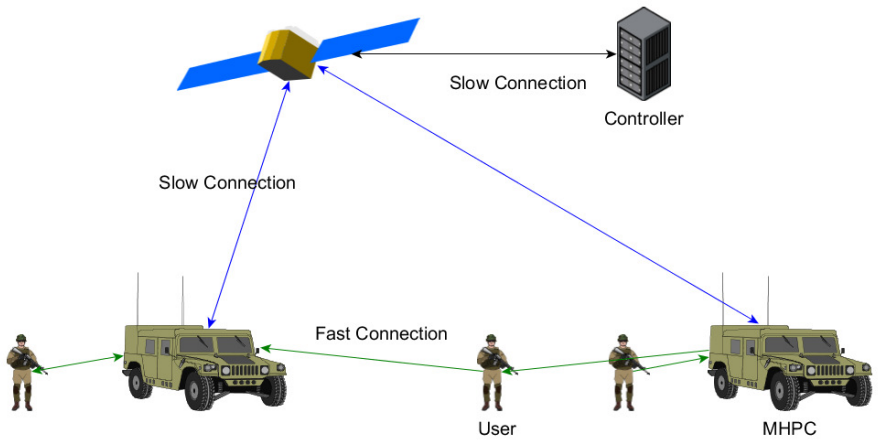
Our research is directly related to traditional *computational ferrying* approaches which aim to solve the problem of transporting messages from one location to another [Zhao et al. 2004]. In computational ferrying, traditional message ferrying [Skordylis and Trigoni 2008; Guo and Keshav 2007] is augmented by allowing for computation capabilities. Our methodology improves upon that of previous computational ferrying approaches by [Monfared et al. 2015].

In Monfared et al. [Monfared et al. 2015], a set of Mobile High-Performance Computers (MHPCs) are coordinated in order to satisfy the requests of multiple users scattered throughout an environment. This allows users to offload computation for mobile devices onto mobile cloudlets, providing greater processing and storage capabilities, as well as reduced power consumption. In order to allow for efficient usage of the ferries, the visit schedules must be carefully designed [Zhao et al. 2005].

However, the trajectories followed by MHPCs in [Monfared et al. 2015] are direct lines between the pickup and delivery locations. In real applications, there are always obstacles and varying terrain between locations. These obstacles will affect vehicle trajectories and the travel time from one location to the next. All of these factors complicate scheduling estimates and should be modeled for effective planning and scheduling strategies. Considering these issues, we add obstacles and path planning to the original problem formulation. These additions will enable a more robust performance in contested and dynamic environments. Operators gain the critical ability to view planned trajectories for vehicles, as well as more realistic travel time estimates [Usbeck et al. 2014]. Moreover, we also incorporate the ability to schedule given different priorities for tasks, and the requirement for specific hardware or software capabilities provided by different MHPCs.

Our ideas relate to *data muling* [Bhadauria et al. 2011; Tekdas et al. 2009; Dunbabin et al. 2006] which similarly to data ferrying explore the idea of a powerful unit that moves around different locations downloading, uploading, and distributing data in wireless sensor networks. In our work, in addition to path planning capabilities, we explicitly model tasks and their deadlines as well as reactive motion strategies to navigate an environment filled with obstacles.

Our work can be seen as an extension to classical *scheduling* problems in Computer Science [Pinedo 2016; Liu and Layland 1973]. A significant difference of our approach with traditional load balancing and scheduling algorithms is that the schedulers need to move to physical locations and the distance needs to be taken into account when calculating load balancing. Also, *multi-robot planning* algorithms naturally connect with our ideas [Parker 2008; LaValle 2006].



Scenario

3. PRELIMINARIES

In this section, we define the mathematical notation that will be used throughout the rest of the paper and then proceed to formulate the problems of interest. The problem setup is composed of a Controller, Tasks, MHPCs, and Obstacles; an example illustration of the relationship between these components is found in Figure 1.

We will assume that we have a group of *users* moving in an *obstacle-filled* environment. These lightweight users can communicate using a high-bandwidth connection with *MHPCs (Mobile High-Performance Computers)* that can help them carry out demanding tasks. This high-bandwidth communication link can be implemented, for example, using visible light communication or other fast and secure modalities of communication. We also assume that the MHPCs can communicate using a low-bandwidth communication link to a remote location where a centralized controller is present. We will give a detailed description of these elements in the next subsections.

3.1 Notation

The environment will be modeled as a 2-dimensional, partially known workspace $\mathcal{W} = \mathbb{R}^2$. In this work, partially known means that we have both known $\mathcal{O}_{known}$ and unknown $\mathcal{O}_{unknown}$ obstacles. The complete set of obstacles is defined as $\mathcal{O} = \mathcal{O}_{known} \bigcup \mathcal{O}_{unknown}$ with $\mathcal{O} \subset \mathcal{W}$. The obstacles $\mathcal{O}$ will be represented as polygons. Let $\mathcal{E} = \mathcal{W} \setminus \mathcal{O}$ represent the free space where units and MHPCs can move without colliding with the obstacles.

Users in the workspace will generate or receive the results of tasks. There will be n tasks, where each task will be modeled as a tuple T^i with pickup and delivery locations $T^i_{LP}, T^i_{LD} \in \mathcal{E}$ (which can be the same location), job length $T^i_L \in \mathbb{R}_{\geq 0}$, a start time $T^i_S \in \mathbb{R}_{\geq 0}$, and a deadline by which the task must be delivered $T^i_D \in \mathbb{R}_{\geq 0}$ with the constraint $T^i_D \geqslant T^i_S + T^i_L$. Tasks may also have additional features which can be used to weight them when scheduling. As an example, in our computations we use priority $T^i_P \in \mathbb{N}$, where completing a higher-priority task is preferred at the cost of missing the deadlines of lower-priority tasks. When scheduling, our algorithms will augment the tasks with additional features: expected pickup and delivery times $T^i_{TP}, T^i_{TD} \in \mathbb{R}_{\geq 0}$, and status indicators for whether the task has been picked up or delivered $T^i_{PU}, T^i_{DL} \in \{0, 1\}$. Multiple tasks can be organized into groups $T^i_G \in \mathbb{N}$ while the system is running.

In the environment there are also present *Mobile High Performance Computers* (MHPCs). Table I) summarizes the parameters of MHPCs. MHPCs are computer platforms with greater processor and storage capabilities compared with the users, and can have computational tasks offloaded to them. Computation can take place regardless of whether the MHPC is adjacent

to a user or is moving. Each MHPC also has p processors with $p \geqslant 1$, and a visiting order vo which is the order in which tasks are scheduled to be picked up and delivered. As an illustration, consider $vo = (1_p, 2_p, 2_d, 1_d)$, which would correspond to picking first task 1 followed by picking task 2, then delivering task 2 followed by task 1. $\mathcal{A} \subseteq E$ is the polygonal representation of the MHPC centered at location $x \in \mathcal{E}$. We assume that MHPCs are capable of short-range, high-bandwidth communications to exchange task data with Users and that they have a long-range, but low-bandwidth communication link to exchange location and planning data with the remote controller. We will assume that they can also sense their environment locally, can detect both known and unknown obstacles, and are capable of limited path-planning. We denote X^j to be the set of feasible configurations for $\mathcal{M}^j$. In the case of multiple MHPCs, the state space over all of the MHPCs is $X = X^1 \times X^2 \times ... \times X^m$ [LaValle 2006].

Number of CPUs M_p	Available Resources M_r	Polygonal Representation $M_{\mathcal{A}}$
Position M_x	Visit Order M_{vo}	Path $M_{\tilde{x}}$

Table I: MHPC Parameters

The obstacle state space X_{obs} consists of collisions between MHPC-MHPC which can be written as:

$$X_{obs}^{jl} = \left\{ x \in X | \mathcal{A}^j(x^j) \bigcap \mathcal{A}^l(x^l) \neq \emptyset \right\}, \tag{1}$$

and MHPC-obstacle collisions modeled as:

$$X_{obs}^{j} = \left\{ x \in X | \mathcal{A}^j(x^j) \bigcap O \neq \emptyset \right\}. \tag{2}$$

The union of these equations yields the complete obstacle region:

$$X_{obs} = \left(\bigcup_{j=1}^{m} X_{obs}^{j} \right) \bigcup \left(\bigcup_{jl, j \neq l} X_{obs}^{jl} \right). \tag{3}$$

The obstacle-free region in the configuration space is defined as $X_{free} = X \setminus X_{obs}$.

As users generate tasks, they will announce them to a controller using its long-distance, low-bandwidth wireless connection. This central controller is capable of receiving user requests and MHPC status updates (location and task progress). Given all the information gathered by the controller, we must determine a schedule and trajectory for each MHPC such that tasks are picked up, computed, and delivered to satisfy best the specified criteria (e.g., priorities and number of tasks completed.). Given the known obstacles, trajectories will be precomputed to allow for more accurate schedule estimation. If an MHPC encounters an unknown obstacle, it can report the location to the controller, which will incorporate it into its map, which will improve the estimations of completion.

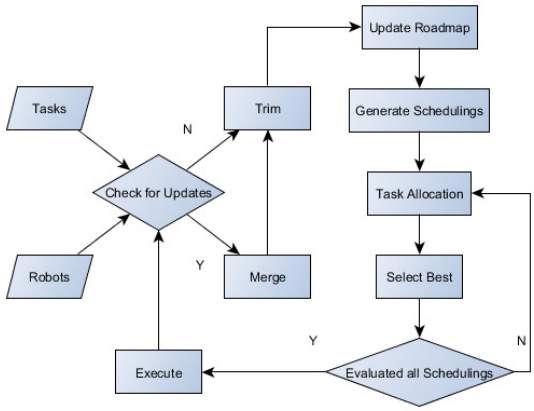
Given these elements, we will define our problems of interest in the next subsection.

3.2 Problem Definition

In our work, we are interested in four problems related to the deployment of MHPCs:

Problem 1 - Computational Complexity: Understand the computational complexity of the computational ferrying problem.

Problem 2 - Task Delivery Estimation: Given a set of tasks T, Obstacles $\mathcal{O}$, and MHPC's $\mathcal{M}$, determine when each task will be picked up and delivered.



Workflow

Problem 3 - Task Allocation and Scheduling: Given a set of tasks T, MHPCs $\mathcal{M}$, and the ability to estimate when tasks will be picked up and delivered, allocate the tasks to MHPCs *and* design a plan for each MHPC that attempts to meet as many task deadlines as possible.

Problem 4 - Path Planning and Obstacle Avoidance: Each MHPC starts in an initial state $x_I^j \in X_{free}^j$ and ends in a goal $x_G^j \subset X_{free}^j$. Given an unbounded time interval $t = [0, \infty)$, we calculate a state trajectory $\tilde{x}$ where the initial state is $x(0) = x_I$ and the final state is $x(t) = x_G$ that takes the MHPC through X_{free}^j avoiding both known and unknown obstacles.

4. METHODS

4.1 Computational Complexity

In our first item of interest, we would like to understand the computational complexity of the computational ferrying with obstacles. To do that, we use the technique of *restriction* [Garey and Johnson 1979]. We want to show that the *Computational Ferrying Problem* contains a well known NP-hard problem as a special case (chosen to be the *Partition Problem*) [Garey and Johnson 1979]. The *Partition Problem* is deciding whether a set of positive integers S can be split into two subsets $S^1 \subset S$ and $S^2 \subset S$ such that the sum of numbers in each set is equal $\sum_{s \in S^1} s = \sum_{s \in S^2} s$.

To connect the computational ferry with obstacles to the partition problem, we will formulate a simplified computational ferrying problem where pick up and delivery locations are equal:

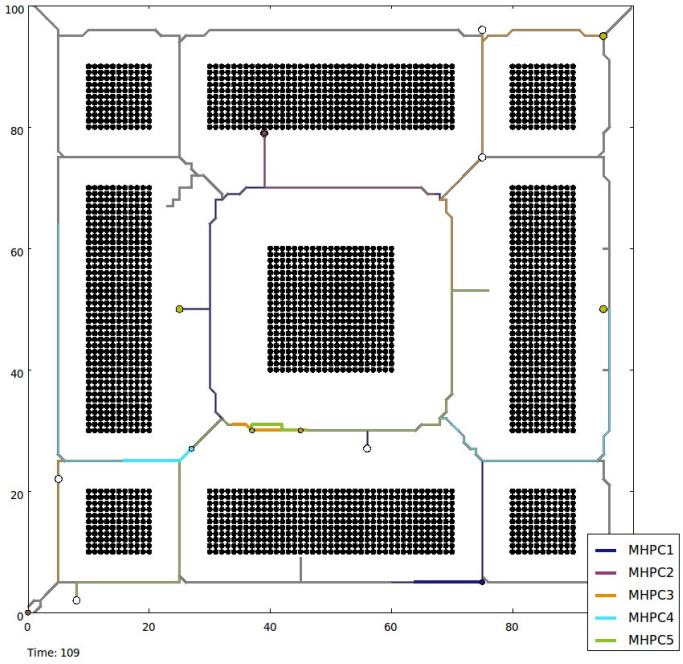
$$T_{LP}^i = T_{LD}^i \forall T^i \in T$$

In this simplified problem, deadlines are ignored ($T_D^i = \infty$) and tasks are available from the beginning at ($t = 0$) for all tasks. Furthermore, we assume that $\mathcal{M}$ consists of just two static MHPCs $\mathcal{M}^1, \mathcal{M}^2$ where their locations are also identical to the tasks $\mathcal{M}_x^1 = \mathcal{M}_x^2 = T_{LP}^1$. Each MHPC is set to have a single processor $\mathcal{M}_p^1 = \mathcal{M}_p^2 = 1$.

Using this restriction, we have removed the effects of MHPCs' motions and arrive at the *Partition Problem* where the positive integers are the task lengths T_L^i, which we desire to partition equally between both MHPCs. Since the *Partition Problem* is NP-Hard, *Computational Ferrying* must also be NP-Hard since it contains *Partition Problem* as a special case.

4.2 Task Completion Estimation

In this subsection, we describe the methods used for solving the problem of allocating tasks to MHPCs and finding their trajectories. The flowchart in Figure 2 illustrates the relationships between the various sub-components of our approach and how information flows from beginning to finish.



Example roadmap around an environment with buildings

Initially, we are given a set of MHPCs $\mathcal{M}$, tasks T, and obstacles $\mathcal{O}$. The factors we will consider when estimating the time at which a task will be picked up or delivered are task execution time, distance to travel, and task execution schedule. The task's execution time will be provided by the user when a task is generated. The schedule will be influenced in part by this task length and the distance an MHPC must travel. The open problem that we need to tackle initially is to estimate travel times.

4.2.1 *Roadmap Construction.* Our first step to estimate travel times is to create a *roadmap* with routes that can be followed to pick up and deliver tasks. Roadmaps are widely used in Mobile Robotics since they provide a convenient way to discretize a continuous environment into a combinatorial graph representation. The central controller will create this roadmap utilizing the set of known obstacles $\mathcal{O}_{known}$ and will create a *Voronoi decomposition* to obtain a graph G. We will also use G for path planning in later sections. The Voronoi decomposition was used since it: 1) discretizes the configuration space (as opposed to path-planning in the entire workspace) simplifying the problem and 2) provides maximum clearance from known obstacles (e.g., roadmap represents a set of safe paths which are equidistant from known obstacles).

An illustration of the roadmap construction is shown in Figure 3. We add to the roadmap task pickup and delivery locations and the location of each MHPC and connect these points to the roadmap. Using this pre-calculated roadmap allows for faster path planning as the search is constrained to this graph, rather than an entire continuous environment. Furthermore, if a particular area in the environment is considered unsafe, operators could also place "virtual obstacles" avoid visiting an area and ensure that the roadmap does not cross these dangerous zones.

Once the roadmap is available to calculate paths, we can use it to estimate trajectory times.

4.2.2 *Tentative Paths.* In this section, we propose Algorithm 1 to generate an initial path for an MHPC and update its visiting order $\mathcal{M}^j_{vo}$. First, a graph search is performed on the roadmap between each pair of consecutive locations and then it is concatenated to form a complete initial tentative path written as $\tilde{x}$. The length of the path (composed of linear segments) and the time it takes to traverse is also calculated as these values are used in the scheduling process.

Algorithm 1 MHPC Tentative Path

Input: $\mathcal{M}^j$
Output: Tentative path representing MHPC trajectory

```
tentativePath ⇐ []
prevLoc ⇐ M^j_x
nextLoc ⇐ Null
scheduled ⇐ [false, ..., false]
for T^i ∈ M^j_vo do
    if T^i_PU == true or scheduled[i] == true then
        nextLoc ⇐ T^i_LD
    else
        nextLoc ⇐ T^i_LP
        scheduled[i] ⇐ true
    τ^σ ⇐ graphSearch(prevLoc, nextLoc)
    tentativePath.append(τ^σ)
    prevLoc ⇐ nextLoc
return tentativePath
```

Once the algorithm calculates an initial tentative path and its length, we will use this information to allocate tasks to MHPCs and schedule their execution.

4.2.3 *Task Timing (Algorithm 2, Line 7).* In order to calculate the schedule, we will first need to estimate the times for pickup, start, and delivery of the tasks based on the following factors: a) MHPC location, b) task pickup/delivery locations, c) task job length, d) MHPC processor schedule, e) MHPC visit order and f) estimated travel distance. In the case of tasks of unknown length, we ignore the execution time and only focus on the time needed to travel between locations. The Task Timing algorithm 2 handles this calculation.

$\mathcal{M}^j$'s tasks are distributed across its processors $\mathcal{M}^j_p$, each task sequentially assigned to the processor with the shortest queue. Using the MHPC's visit order, a graph search on the roadmap finds the shortest path between consecutive visit points and returns an estimate of the distance that needs to be traversed. This estimate is sufficient when picking up a task, as travel time is the only factor that needs to be taken into account. When delivering tasks, we also need to incorporate the remaining task processing time which will be the maximum between the estimated travel time and the remaining task computation time. This maximum value will be the final delivery time. When this calculation is completed the tuple $pInfo = (FT, MD, TM)$ is returned where FT is the finish time of the last task in the visit order, MD is the number of deadlines expected to be missed, and TM is the set of tasks which will be missing their deadlines.

4.3 Task Allocation and Scheduling

The goal of this subsection is to find a set of task allocations for the available MHPCs and a schedule for each MHPC that maximizes the number of tasks meeting their deadlines. At the same time, we need to enforce the constraints imposed by available resources, distances, and computing time.

In order to handle the task-length, we consider two cases:

—The task length is known or can be approximated using historical data

3	3	1	2	2	1
3	1	3	2	2	1
3	1	2	3	2	1
...	...	...	...	...	...
1	2	2	3	1	3
1	2	2	1	3	3

Table II: Illustrative example of the placement generation subroutine

—The task length is unknown

4.3.1 *Placement Generation (Algorithm 2, Line 5).* We first propose an algorithm to update the visiting order $\mathcal{M}^j_{vo}$ to include a new task T^i. The task is inserted twice (representing pickup and delivery) in every valid location of the visiting order. A temporary list *pList* will store each of these placements. In the case of tasks of unknown length, we place the task behind all tasks with higher priority, and ahead of tasks with lower priority. Among tasks of the same priority, tasks with unknown computation length will be executed last, giving priority to tasks with all known parameters. Once all placements have been generated in time complexity $O(n^2)$ the list *pList* is returned.

As an illustration of this subroutine, consider Table II where a visiting order of $(1, 2, 2, 1)$ is processed indicating first picking up task 1, followed by a pickup of task 2, and then the delivery of tasks 2 and 1. We now wish to generate valid orderings of pickup and delivery locations for new task 3, without perturbing the existing task orderings.

4.3.2 *Task Group Scheduling (Algorithm 2).* In this subsection, we will describe a task scheduling Algorithm. The controller first sorts the set of tasks in T' by their deadlines (Line 1). Then, it iterates over each task in the sorted list T' and every MHPC in $\mathcal{M}'$ that is capable of executing that task (with the required capabilities to handle the task),and then we generate candidate placements (line 5) of the visiting order with the new task inserted in feasible location of $\mathcal{M}^j$'s visiting order. In line 9, the Task Timing Algorithm estimates for each task, each its pickup, delivery, and start times.

This process is repeated for all the MHPC $\mathcal{M}^j \in \mathcal{M}'$ leading to a $O(mn^3)$ time complexity. Once the algorithm calculates all orderings and time, the solution is the placement order which fulfills the highest number of requirements in descending order (priorities, deadlines). The new solution is then used in the next iteration over T'.

Algorithm 2 Task Group Scheduling Algorithm

Input: T, $\mathcal{M}$
Output: $\mathcal{M}'$

```
1:  T' ⇐ Sort T by ascending deadlines
2:  for T^i ∈ T' do
3:      orderings ⇐ []
4:      for M^j ∈ M do
5:          pList ⇐ placementGeneration(M^j, T^i)
6:          for p ∈ pList do
7:              timedPermInfo ⇐ taskTiming(M^j, p)
8:              orderings.append(timedPermInfo)
9:      Sort orderings by user conditions
10:     Assign tasks to MHPCs based on orderings[0]
    return M
```

During a plan's execution in a dynamic environment, many events may happen that make the

plan invalid. For instance, unknown obstacles can appear, or MHPCs can fail and need to be taken offline. In these cases, we must modify our initial plan.

4.4 Path Planning and Obstacle Avoidance

The Task Allocation algorithm 2 generates different task placements and estimate the timings to evaluate the solutions. Finally, we pick the best trajectories for the MHPCs that allow them to visit the tasks' locations.

4.5 Update

Once initial plans are calculated, new tasks will be arriving and need to be processed. We will handle these new tasks as follows: 1) check for new incoming tasks and MHPCs, 2) remove completed tasks, and 3) update the status of tasks. This procedure will be explained below.

First, upon receiving a group of MHPCs, $\mathcal{M}$, and a list of tasks T', we combine existing tasks not yet picked up by an MHPC with the incoming list of tasks. The tasks that have already been picked up by an MHPC are not modified.

Second, given a generated initial solution, we can check each MHPC's processor schedule and each task's processing time to determine if the task's deadline is missed or met. If an MHPC, $\mathcal{M}^j$, is at the current task's pickup location, the picked up variable is set to true, $T^i_{PU} = true$. When a task has completed processing and $\mathcal{M}^j$ is at its delivery location, then the algorithm sets $T^i_{DL} = true$.

Third, jointly with Task Scheduling, we also monitor for new tasks and MHPCs being added or removed to ensure that all information is up-to-date. When a new group of tasks is available, we handle new or removed tasks and MHPCs. The result is sent to Task Group Scheduling which outputs the new set $\mathcal{M}$ that represents the MHPCs and their assigned tasks and paths.

When an MHCP completes a task with a previously-unknown execution length, we execute Update. If the remaining schedule for the MHPC's task queue has been shifted back by the task's execution length, we can treat the remaining tasks as a set of new incoming tasks, allowing them to be reassigned among the MHPC's as needed to maximize the completion rate.

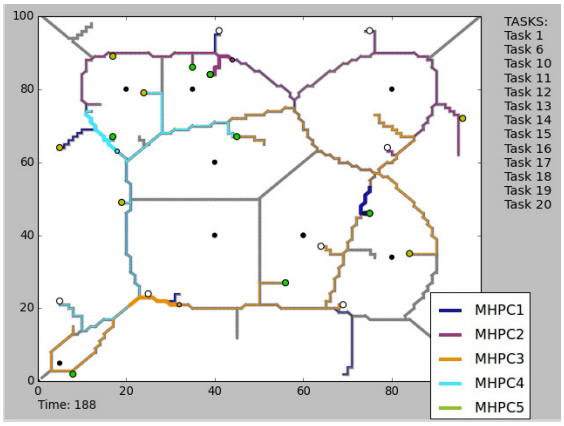
4.6 Path Planning

One essential capability that MHPCs require to carry out their tasks is the ability to navigate in their environment to visit pickup and delivery locations. The path planning component of the approach is divided into two parts. In the first part, the controller is responsible for designing a tentative path that avoids known obstacles. This procedure also allows operators to designate known safe lanes which are expected to be obstacle-free and that composed the initial paths. In addition to this, we propose a reactive planning component that can be implemented on-board on each MHPC. The sensor requirements to implement this strategy are low and can be realized using inexpensive sensing modalities. By using these capabilities, MHPCs can navigate around dynamic or unknown obstacles.

When new tasks are available for processing, the central controller adds pickup and delivery locations of each task to the computed roadmap. Since the roadmap avoids all known obstacle, an obstacle-free straight line connecting the roadmap to the task location ensures obstacle avoidance. We repeat the same process to add MHPCs original locations.

4.6.1 *Hybrid Paths and Path Splicing.* The paths obtained by the above process have the advantages of being relatively simple to design and search. However, they do not take into account unknown obstacles that may appear and were not part of the initial calculations. Furthermore, since the communication link between the controller and the MHPCs has low-bandwidth, it may not be possible for the controller to deal directly with obstacles that appear.

To solve this issue, we must allow the MHPCs to react to new obstacles. If a previously unknown obstacle $o \in \mathcal{O}_{unknown}$ is encountered, the MHPC will re-plan its path around it using A* search [Zeng and Church 2009] avoiding X^j_{obs}. The MHPC selects a preliminary goal x_G



Simulation Example

located r steps ahead from the start of the encountered obstacle. This constant r is a parameter that depends on the environment and must be larger than most obstacles (in order to navigate around them) but small enough to avoid excessive calculations. Using r an obstacle avoiding path is calculated through X^j_{free}. If the MHPC can not reach x_G because of additional obstacles in $\mathcal{O}_{unknown}$, the path is re-plan a further r steps ahead. The same idea will also be used in MHPC-MHPC collisions where a path path $\tilde{x}$ through X^{ij}_{obs}.

5. EXPERIMENTAL RESULTS

We have implemented the algorithms described in the sections above in a computer simulation implemented in the Python programming language. In this section, we present the results of several experiments in different scenarios to test the practical feasibility of our ideas.

5.1 Software Simulation

We created a custom simulator to have better control over the MHPC's functioning, permitting fine-grained control over the Customized path-planning, unknown obstacle avoidance, and processor/task scheduling and prioritization.

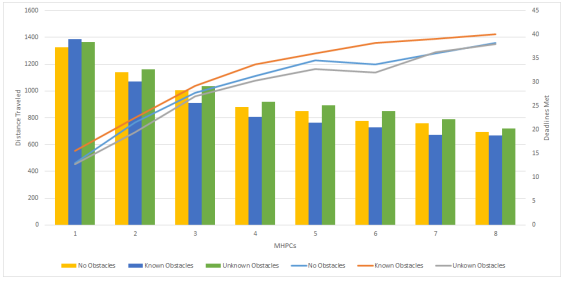
Simulations were performed using between 1 and 8 MHPCs and 1 to 4 processors. We varied the number of obstacles as follows: 0 known and 0 unknown obstacles, 8 known and 0 unknown obstacles, or 0 known and 8 unknown obstacles. There were 40 tasks present in the workspace, 10 each with 4 different priorities. A snapshot of the simulation running is presented in Figure 4.

5.2 Effects of Obstacles on MHPC Movement and Performance

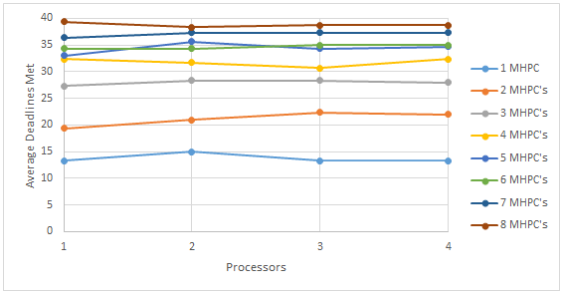
We want to understand how obstacles in the environment affect the performance of our algorithms. As illustrated in Figure 5, the presence of unknown obstacles in the environment causes MHPCs to travel further than with known obstacles, as this path planning component cannot be optimized. An interesting effect is that the first few known obstacles decrease the path length. This is due in part to a more complex roadmap that has more alternative routes.

We are also interested in understanding the number of deadlines met. The behavior of this variable is similar to the effect on travel distances. With a roadmap generated for an environment containing no known or unknown obstacles, we see an average decline of 2.28 deadlines met compared to an environment with known obstacles. Similarly, when comparing an environment that has 8 known obstacles vs one that has 8 unknown obstacles, there is a decline on deadlines less being met (of 3.22).

In experiments that have more than 1 MHPC, we find that having an environment with known obstacles consistently performs better than all cases. As expected, the presence of unknown obstacles leads to worse performance.



Visibility of Obstacles vs Average Distance Traveled and Deadlines Met



Average Deadlines Met vs Number of Processors

5.3 Effects of Number of MHPCs and Processors on Deadlines

In Figure 6 it is shown that increasing the number of processors across MHPCs has a negligible effect. However, this is likely a consequence of our simulation's parameters, where tasks generally had a short run time. This results in the most influential factor in task completion is distance between locations, rather than the duration of tasks or the number of tasks being processed. It is expected that in the case of geographically close tasks with longer run times, better results with less MHPCs can be achieved so long as they are equipped with more processors.

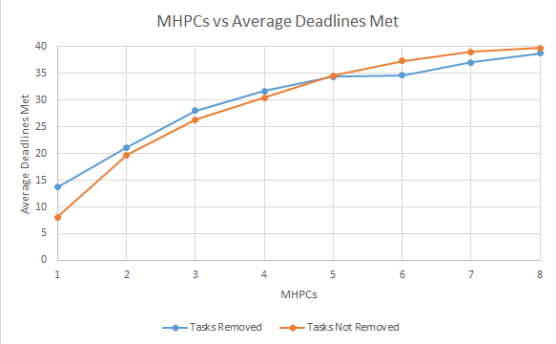
5.4 Effects of Removing Tasks on on Deadlines and Steps

We have done experiments to evaluate the effect of removing or not tasks. As illustrated in Figure 7 the more available MHPCs, the higher the average deadlines met. Interestingly, there seems to be no effect of removing tasks, and the number of deadlines met. In contrast, as shown in Figure 8, eliminating tasks affects the average steps taken. However, as the number of MHPCs increase, this effect is reduced. Further investigations may be required to understand these effects fully.

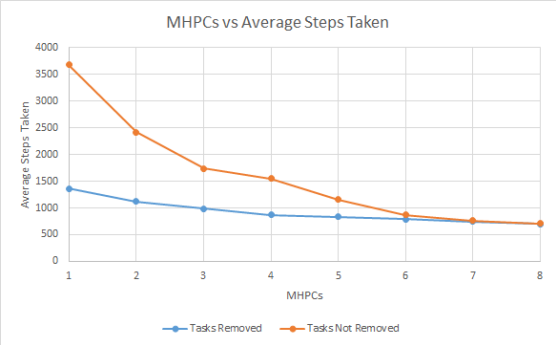
6. CONCLUSIONS AND FUTURE WORK

In this paper, we have extended and formulated a problem to schedule, plan, and deploy Mobile High Performance Computers (MHPCs) that can physically move to serve mobile units to provide additional storage and processing capabilities to lightweight units. We have improved and made additions to different aspects of the current state of the art.

We have modeled MHPCs as autonomous vehicles which leads to more realistic computations in realistic environments that have obstacles. We have proposed path planning algorithms that can find reliable *a priori* estimation of distances between tasks pickup and delivery locations to obtain accurate scheduling times. We introduced several prioritization schemes that allow operators to assign weights to tasks. In this paper, our algorithms are implemented and tested in a computer simulation to understand the effect of completion time due to obstacles, number of MHPCs, and the number of processors.



Effects of Removing Tasks on Deadlines



Effects of Removing Tasks on Steps

Our contributions and improvements with respect to the previous related work include: 1) We schedule tasks to MHPCs each with one or more processors, 2) We consider MHPC capabilities when adding and removing MHPCs to the scheduler, 3) We incorporate path planning and dynamic obstacle avoidance creating a more accurate distance measure.

A significant contribution of this paper compared to prior work is the addition of path planning abilities to MHPCs. In related work [Monfared et al. 2015], MHPCs are only able to move in a straight line between task locations and the modeling not explicitly incorporated obstacles. Given the dynamic nature of the environment where this problem takes place, this assumption should be lifted, and MHPCs must be able to navigate around obstacles and other vehicles or MHPCs. The roadmap approach [Bhattacharya and Gavrilova 2008] and path planning algorithms proposed allow more flexibility in carrying out tasks. Furthermore, through the use of a roadmap, we have a more realistic estimate of the travel distance between locations and are able to safeguard against known obstacles more effectively.

In our formulation, we have allowed for greater flexibility when defining tasks. For instance, specific tasks may be given priority over others, such that high-priority tasks are executed at the cost of lower-priority tasks. One possible extension is to model tasks that require specialized resources from appropriately equipped MHPCs. As an example, consider an MHPC that might have hardware capabilities not available in other MHPCs or may have valuable data stored that is not replicated on other MHPCs. This problem requires the scheduling algorithm to allocate the MHPC to whichever tasks need those specific capabilities.

One of the suggested future works outlined in [Monfared et al. 2015] calls for the ability to remove and add MHPCs dynamically at runtime. We added this capability to our work. Any time a new MHPC is added or removed, its location is updated on the roadmap, and a rescheduling takes place. Additionally, the original implementation in [Monfared et al. 2015]

assumed an identical number of processors across all MHPCs. While not covered in detail here, our formulation also allows for each MHPC to have a different number of processors.

Other directions for future work includes given users the ability to move, and update the controller of their new locations. This ability would require our algorithm to reschedule the MHPCs and also estimate the likely location of a moving user. In our current work, we provided a greedy approach for scheduling tasks. We would also like to formally calculate the approximation ratio to determine how effective this solution is compared to the optimal solution.

7. ACKNOWLEDGEMENTS

Mr. Zanlongo would like to thank FIU's Applied Research Center (ARC) and the Department of Energy Office of Environmental Management for providing the opportunity to be a DOE Fellow under the Science and Technology Workforce Development Initiative and facilitating research opportunities under the DOE-FIU Cooperative Agreement DE-EM0000598. Part of this material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 2017ST062000002.

REFERENCES

Bhadauria, D., Tekdas, O., and Isler, V. 2011. Robotic data mules for collecting data over sparse sensor fields. *Journal of Field Robotics 28,* 3, 388–404.

Bhattacharya, P. and Gavrilova, M. L. 2008. Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path. *Robotics & Automation Magazine, IEEE 15,* 2, 58–66.

Carlson, M. 2015. Project loon.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. 2010. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 49–62.

Dawson, B. and Doria, D. L. 2015. Real-time visualization system for computational offloading. Tech. rep., DTIC Document.

Dunbabin, M., Corke, P., Vasilescu, I., and Rus, D. 2006. Data muling over underwater wireless sensor networks using an autonomous underwater vehicle. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2091–2098.

Garey, M. R. and Johnson, D. S. 1979. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*.

Gordon, M. S., Jamshidi, D. A., Mahlke, S. A., Mao, Z. M., and Chen, X. 2012. Comet: Code offload by migrating execution transparently. In *OSDI*. 93–106.

Guo, S. and Keshav, S. 2007. Fair and efficient scheduling in data ferrying networks. In *Proceedings of the 2007 ACM CoNEXT conference*. ACM, 13.

Kemp, R., Palmer, N., Kielmann, T., and Bal, H. 2012. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*. Springer, 59–79.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge university press.

Liu, C. L. and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM) 20,* 1, 46–61.

Monfared, A., Ammar, M., Zegura, E., Doria, D., and Bruno, D. 2015. Computational ferrying: Challenges in deploying a mobile high performance computer. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a*. IEEE, 1–6.

Parker, L. E. 2008. Multiple mobile robot systems. In *Springer Handbook of Robotics*. Springer, 921–941.

Pinedo, M. L. 2016. *Scheduling: theory, algorithms, and systems*. Springer.

Shires, D., Henz, B., Park, S., and Clarke, J. 2012. Cloudlet seeding: Spatial deployment for high performance tactical clouds. In *Parallel and Distributed Processing Techniques and Applications*.

Skordylis, A. and Trigoni, N. 2008. Delay-bounded routing in vehicular ad-hoc networks. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 341–350.

Sookoor, T., Doria, D., Bruno, D., Shires, D., Swenson, B., and Pollock, L. 2014. Offload destination selection to enable distributed computation on battlefields. In *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 841–848.

Sookoor, T. I., Bruno, D. L., and Shires, D. R. 2013. Allocating tactical high-performance computer (hpc) resources to offloaded computation in battlefield scenarios. Tech. rep., DTIC Document.

Tekdas, O., Isler, V., Lim, J. H., and Terzis, A. 2009. Using mobile robots to harvest data from sensor fields. *IEEE Wireless Communications 16,* 1, 22–28.

Usbeck, K., Gillen, M., Loyall, J., Gronosky, A., Sterling, J., Kohler, R., Newkirk, R., and Canestrare, D. 2014. Data ferrying to the tactical edge: A field experiment in exchanging mission plans and intelligence in austere environments. In *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 1311–1317.

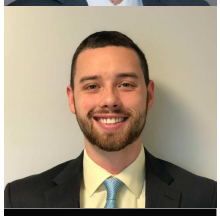
Zanlongo, S. A., Wilson, A. C., Bobadilla, L., and Sookoor, T. 2016. Scheduling and path planning for computational ferrying. In *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 636–641.

Zeng, W. and Church, R. 2009. Finding shortest paths on real road networks: the case for A*. *International Journal of Geographical Information Science 23,* 4, 531–543.

Zhao, W., Ammar, M., and Zegura, E. 2004. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 187–198.

Zhao, W., Ammar, M., and Zegura, E. 2005. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *INFOCOM 2005. 24th annual joint conference of the IEEE computer and communications societies. Proceedings IEEE*. Vol. 2. IEEE, 1407–1418.

Sebastian Zanlongo is currently a PhD candidate in the School of Committing and Information Sciences at Florida International University. He is interested in multi-agent systems, cooperative control, and nuclear robotics. He has published works on multi-robot coordination for military applications, human-robot-interaction, and informative path planning. His research has been sponsored by the Department of Energy, Department of Homeland Security, and Intel.



Alexander Wilson is a Senior Full Stack Engineer at T. Rowe Price in Baltimore, MD. He is interested in software architecture, designing front-end components for optimal reusability, and developing fault-tolerant, scalable APIs. He received a Bachelors degree in Computer Science from Towson University in Towson, MD and has worked in government, healthcare, and financial sectors. He has given talks about development frameworks and best practices and has been published by IEEE.

Dr. Tamim Sookoor (JHU/APL) is currently an Acting Section Supervisor at the Johns Hopkins University Applied Physics Laboratory. He is interested in developing technology to secure Cyber-Physical Systems in order to protect critical infrastructure such as public utilities and military platforms. He earned a B.E. in Computer Engineering from Vanderbilt University. He received his M.S. and Ph.D. degrees in Computer Science from the University of Virginia. He has published several research papers in journals and refereed conference proceedings in Cyber-Physical Systems, Smart Cities, and Wireless Sensor Networks.

Dr. Leonardo Bobadilla (FIU) is currently an Assistant Professor in the School of Computing and Information Sciences at Florida International University. He is interested in understanding the information requirements for solving fundamental robotics tasks such as navigation, patrolling, tracking, and motion safety and has deployed testbeds that can track and control a large number of mobile units that require minimal sensing, actuation, and computation. He received his Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign. He has published several research papers journals and refereed conference proceedings in Robotics, Automation, and Sensor Networks. His research has been sponsored by the Army Research Office, Department of Homeland Security, and the Ware Foundation.