Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Multi-Agent Sensor Data Collection with Attrition Risk

Jeffrey Hudack
Air Force Research Laboratory
Rome, NY, USA
jeffrey.hudack@us.af.mil

Jae C. Oh
Syracuse University
Syracuse, NY, USA
jcoh@ecs.syr.edu

Abstract

We introduce a multi-agent route planning problem for collecting sensor data in hostile or dangerous environments when communication is unavailable. Solutions must consider the risk of losing robots as they travel through the environment, maximizing the expected value of a plan. This requires plans that balance the number of agents used with the risk of losing them and the data they have collected so far. While there are existing approaches that mitigate risk during task assignment, they do not explicitly account for the loss of robots as part of the planning process. We analyze the unique properties of the problem and provide a hierarchical agglomerative clustering algorithm that finds high value solutions with low computational overhead. We show that our solution is highly scalable, exhibiting performance gains on large problem instances with thousands of tasks.

Introduction

We introduce a collection planning problem with attrition risk that includes a risk of losing agents during execution as part of the planning process. Collection tasks require agents to collect sensor data at a location and then return to a base location in order to complete the task. The objective is to maximize the expected value of the completed tasks, taking into account the path chosen for each agent and the potential for losing agents. The problem definition does not specify how many agents to dispatch and does not require that all tasks are completed, both of which are often assumed by similar problems. The value of a solution is determined by the probability of successful collection (and return) of the sensor data, the probability of losing each robot, and the relative value of the individual agents with respect to the tasks being performed. In some cases, where the risk is too high or the asset is highly valued, doing none of the tasks is optimal. This intuitive concept is unique among task allocation problems, that often seek to minimize the cost of completing all tasks.

This work derives from diverse bodies of work in Multi-Robot Task Assignment (MRTA) (Gerkey 2004) and Vehicle Routing Problems (VRP) (Pillac *et al.* 2013). These fields, while having been generally distinct in their definitions and solution concepts, have common goals. Both seek to minimize or maximize some objective function while providing a plan that achieves all of the specified tasks. In almost all cases, the number of agents to be used is specified a priori, and the expectation is that all tasks will be completed as part of a valid solution. Additionally, there is an assumption that all specified tasks are to be completed as part of a valid solution. The collection planning problem introduced here relaxes these constraints, providing a number of unique challenges and motivating new solution methods.

To illustrate our problem, consider a team of hunters gathering food for their village from a surrounding landscape that contains dangerous predators. Any food that is collected must be carried back to the village by the hunter, so if the hunter is killed by a predator then nothing gathered by that hunter is returned. There are a number of hunting locations that are known to have food available, and we wish to maximize the amount of food gathered without unnecessarily exposing the hunters to danger. Sending someone out to gather one rabbit may not be worth the risk of losing a skilled hunter en route, but sending her out to visit all of the hunting sites has a high likelihood of encountering a predator along the very long route. We seek an allocation of hunters to ordered sets of sites that will find a balance between the expected amount of food returned to the village, and the risk of losing the hunters.

In general, risk-based allocation seeks to minimize the chance of losing one or more things of value. There is work in both the MRTA and VRP communities that include a notion of risk mitigation. MRTA algorithms are often resilient against the loss of agents during execution (Choi *et al.* 2009), but their planning process does not explicitly consider minimizing the effects of threats. Risk has been modeled in a number of ways; uncertainties on cost of task completion (Kang and Ouyang 2011) (Hazon *et al.* 2013), uncertainty of travel cost on edges (Nikolova *et al.* 2006), and adding risk as a cost constraint to be optimized (Erkut and Verter 1998) (Talarico *et al.* 2013). However, these approaches do not consider the expected loss of vehicles as part of the planning process.

This work is has a number of similarities with existing work in robustness for multi-robot systems (Krieger and Billeter 2000), where the goal is to minimize the risk of mission failure. However, this is often not the primary objective be-

ing measured, but rather an additional constraint to be analyzed. When the goal is to maximize utility, also referred to as soft goals, there are also ways to recompile a problem definition to make finding solutions more tractable (Keyder and Geffner 2009). Such approaches rely on a logical definition of a problem space, which can be difficult to engineer from a problem defined in a continuous space.

Our solution methods, referred to as Progressive Risk-aware Clustering (PRC), uses hierarchical agglomerative clustering to generate plans for groups of robots to complete a set of collection tasks with the risk of attrition. Clustering is a common method for solving multi-agent routing problems, often used to generate initial solutions to guide the search process. The overwhelming majority of clustering solutions are intended for problems with a known number of agents (Campbell *et al.* 2008) (Zhang *et al.* 2010) (Lagoudakis *et al.* 2004) (Boctor *et al.* 2003). This is not the first work to use hierarchical clustering to find task groupings (Sabo *et al.* 2014), but we do not specify the number of agents to be used.

In the next section, we provide a definition for the Collection Planning Problem with Attrition-Based Risk (CPPAR) and provide examples that illustrate the unique aspects of this problem. We motivate solutions that use a variable number of agents and show the conditions under which agents will choose to not complete some or all of the proposed tasks. Then, we introduce our Progressive Risk-aware Clustering algorithm, which generates multi-agent plans using a form of hierarchical agglomerative clustering. Finally, we provide experimental results using our algorithm on a range problem instances.

Background

A *task allocation problem* specifies a nonempty set of agents $A = \{a_1, a_2, ..., a_n\}$ and a nonempty set of tasks $T = \{t_1, t_2, ...t_m\}$. The goal is to find a matching of tasks to agents that maximizes a global reward function. The reward function is assumed to be a sum of local rewards, with the local reward begin determined by the tasks assigned to a single agent. We adopt an objective function similar to that found in (Choi *et al.* 2009),

$$\max \sum_{i=1}^{n} \left(\sum_{j=1}^{m} c_{ij}(x_i, p_i) x_{ij} \right), \tag{1}$$

where c_{ij} is the score for agent i completing task j, and $x_{ij} = 1$ if agent i is assigned to task j and 0 otherwise. The vector $x_i \in \{0,1\}^m$ is the set of tasks assigned to agent i where the jth element is x_{ij}. The vector $p_i \in (T \cup \{\emptyset\})^{|T|}$ is an ordered sequence of tasks assigned to agent i, and the kth element is $j \in T$ if agent i executes task j at the kth point within the path.

We assume each task $t_i \in T$ has an associated location $v_i \in V$ in 2-dimensional space. Additionally, we specify the location of the base b, where the agents start. This yields the full set of all location vertices $V = \{b, 0, 1, ..., m\}$. We are also given a set of edges $E = \{e_{ij} = (i,j) : i \neq j \text{ and } i, j \in V\}$. Each edge e_{ij} has a non-negative distance represented as $d(e_{ij})$, or d_{ij} for short . We can represent this structure as a graph $G =< V, E >$.

Collection Planning Problem with Attrition Risk (CPPAR)

Planning for collection adds the additional constraint that each task in $T = \{t_1, t_2, ...t_m\}$ is a collection task, requiring the agent collect a sensor reading from the task location and return it to the start location. Additionally, the environment has inherent dangers that may, with some probability, disable or destroy the agent. A task is considered complete only when the sensor data from the associated location is delivered to the base station. Because we are providing an a priori plan, if an agent is disabled during the execution of a path, all tasks $\{t_1, ..t_k\}$ on the path will remain incomplete. All gathered data is lost with the platform and the remaining sensor data on the path is never collected.

The probability of the agent surviving an edge traversal is $p_s(e_{ij})$, and each edge has an independent probability of survival. Each edge is a series of events, with a probability of successfully traversing the edge as a cumulative probability over traversing a collection of unit length segments in sequence. Let ψ be the probability of successfully traversing a unit distance. For an edge e_{ij} with distance d_{ij}, the probability of successfully traversing the entire edge is

$$p_s(e_{ij}) = \psi^{d_{ij}} \tag{2}$$

such that $\lim_{d_{ij} \to \infty} p_s(e_{ij}) = 0$. The value used for ψ may be derived from a number of sources: hardware failure during operation of the vehicle, the platform may become unreliable after repeated use, or threats may exist that can destroy the vehicle. While we use discrete distance values in our examples, this formulation also applies to continuous distances.

Consider a path for agent a_i, $\pi_i = (b, e_{b1}, v_1, e_{12}, ..., v_j, e_{jk}, v_k, e_{kb}, b)$. The *task expected value* of the single agent path is defined as

$$E^t(\pi_i) = \prod_{e_{jk} \in \pi_i} p_s(e_{jk}) \sum_{v_p \in \pi_i} c_{ap}. \tag{3}$$

where c_{ik} is the score for agent a_i completing the task k. Intuitively, the expected value of the path is the probability of successfully traversing the entire path times the sum over the scores for agent a_i completing each of the tasks in the path. In the multi-agent case, the objective is to maximize the expected value over all agents. Given a set of agent paths $\pi_i \in \Pi$, the overall expected task value is

$$E^t(\Pi) = \sum_{\pi_i} E_t(\pi_i). \tag{4}$$

To maximize $E_t(\Pi)$, we must find an assignment of tasks among agents that maximizes the sum over their individual expected value. This requires ensuring that the distance of the path traveled by each agent through their assigned tasks is also minimized. The result is a trade-off between how many agents are used, how long their paths are, and the ordering of the tasks within each path.

Including Asset Value

While task completion is an important objective, we must also consider the relative value of the assets being deployed. Failure to complete a path not only loses the value of the assigned tasks, but also the asset itself. We use θ_i to represent the asset value for agent a_i, which may represent the cost of manufacturing the vehicle, the relative abundance or scarcity of the platform, or an external force, such as needing the platforms for another mission the next planning cycle.

In general, θ provides the value of the platforms with respect to the value of the tasks being completed. For example, consider a sophisticated aircraft piloted by agent a_i with a suite of expensive sensors. This aircraft has a high cost to manufacture, and sending it into a contested environment needs to be justified. If there is a single, low-value task located deep within dangerous territory, it's likely not worth sending the platform. Such a platform would have a very large θ_i value. However, we may also have the option of using low-cost, expendable platforms that can be sacrificed to complete a task, which would be indicated by a low θ_i value. The *attrition-based expected value* for a path is

$$E(\pi_i) = \prod_{e_{jk} \in \pi_i} p_s(e_{jk}) \sum_{v_p \in \pi_i} c_{ip} - \theta_i \cdot (1 - \prod_{e_{jk} \in \pi_i} p_s(e_{jk})). \tag{5}$$

This considers not only the expected value of the tasks, but also the probability and value of losing the asset controlled by the agent. As the path length increases, the probability of successfully completing the k tasks on the path decreases, and the probability of losing the asset of value θ_a increases. This results in a function that varies based on the number of tasks on an agent's path, the distance they are from the base, and the distance of the path between the tasks being collected. Similarly to Equation 4, we define the overall expected value for the set of all agent paths as

$$E(\Pi) = \sum_{\pi_i \in \Pi} E(\pi_i). \tag{6}$$

When $\theta = 0$ for all agents, meaning the assets are expendable, the best solution is to send one asset to collect on each task location. This minimizes the distance traveled to collect on each task, which also minimizes the risk of not completing that task. On the other hand, if θ is very large, the assets may not worth risking to collect on any task. This can yield valid solutions that perform none of the tasks. Examples of the effects of θ are given in the following sections.

Effects of Number of Tasks

In the general case, where tasks have an arbitrary location, it can be difficult to show how the length of an agent path effects the expected value for an agent. Distances between tasks and the base can vary, and minor differences can yield significantly different solutions. If an agent is visiting a set of tasks, the value of that path is determined by the relative location of the tasks in the space and the order of tasks visited. Ideally, the path chosen is the shortest path visiting all tasks, which is equivalent to solving an instance of the traveling salesperson problem on a subset of problem vertices.

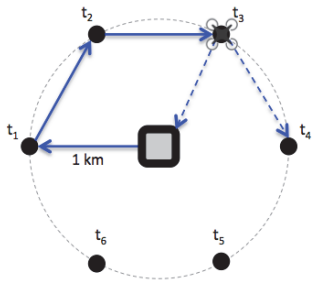

Figure 1: Example single agent problem with 6 tasks 1km apart on a 1km unit circle (left). An optimal plan will visit a series of adjacent tasks in order before returning to the base.

In order to isolate the effects of the number of tasks on expected value, we first present a trivial example with one agent and tasks evenly space on a 1 kilometer unit circle (with $u = 1$ km), shown in Figure 1. Because the distances between tasks are equal, the only decision is to determine how many tasks to visit before returning to the base.

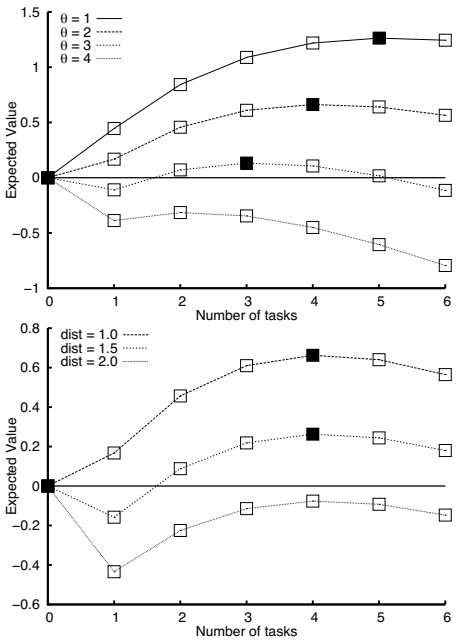

Figure 2: For the example in Figure 1, when $\psi = 0.85$ the number of tasks in the highest value assignment (filled boxes) depends on the asset value, θ (top). As the distance of the tasks from the base is increased (when $\theta = 2$), the value of completing the tasks is reduced by increased risk to the asset (bottom). The line plotted between these discrete points is included for clarity.

In Figure 2, we show the resulting expected value for an agent plan with an increasing number of tasks from the example in Figure 1. Because of the equidistant layout, the order of the tasks visited does not effect the distance of the path. As the value of θ is increased, the number of tasks that

should be collected is reduced. This is due to the additional risk incurred by the length of the path, which exposes the agent to more opportunities to be destroyed. If θ is large enough, the best course of action is to collect on zero tasks and remain at the base.

We also show the effects of task distance from the base, which can be thought of as increasing the radius of the ring of tasks in Figure 1. In this case, while the value of the tasks is reduced by the additional travel distance, the number of tasks that maximizes the expected value remains the same. However, if the distance of the tasks is too far, it is not worth incurring the risk of traveling to and from the tasks and the agent remains at the base.

Multi-Robot Interaction

When we are solving for more than one agent, we must consider the joint expected value over all agents. The subset of tasks that would maximize the expected value for one agent may not be the best subset to choose when there are other agents available. As with the single-agent case, the value of a solution is affected by the distance between tasks, their distance from the base, and the value of the asset.

To illustrate the effects of θ on multi-agent plans we provide another simple example, shown in Figure 3, with a base and 3 collection tasks. The labeled edges indicate the distance between tasks in kilometers, and the probability of survival per unit moved is $\psi = 0.8$. The goal is to generate an assignment of up to three agents that maximizes the expected value.

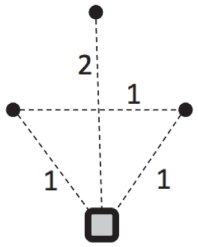

Figure 3: Example set of 3 tasks, denoting the distance between task locations. The probability of survival per unit traveled is $\psi = 0.8$. The objective to to find an assignment of up to 3 agents that maximizes the expected utility.

In Figure 4, we show three possible assignments using a different number of agents. We also include the empty assignment, $\emptyset$, in which no agents are assigned to any tasks. These are only a subset of all possible assignments, but they are useful to demonstrate the effects of different asset values. We consider four scenarios with different platforms of increasing value, with each having the same value of θ drawn from $\{0, 1, 2, 3\}$. In Table 1 we provide the resulting expected value for each assignment for different values of θ, with the best choice shown in bold.

When the assets have no value ($\theta = 0$), the user is willing to risk losing them to maximize collection. In this case, the optimal solution is to always assign one agent to each task, as it will minimize the risk of any one task not being

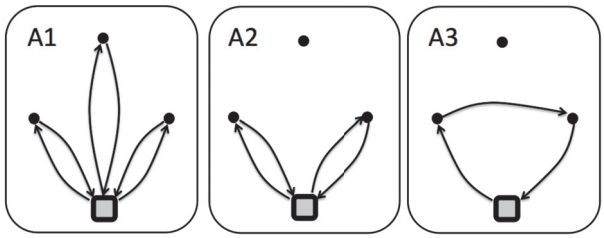

Figure 4: Three possible path allocations of agents to tasks.

	Task Assignment			
θ (Asset Value)	**A1**	**A2**	**A3**	$\emptyset$
0 (Expendable)	**1.69**	1.28	1.02	0
1 (Low Value)	0.38	**0.56**	0.54	0
2 (Medium Value)	-0.93	-0.16	**0.05**	0
3 (High Value)	-2.24	-0.88	-0.44	**0**

Table 1: Task assignment with the maximum expected value for varying asset value, θ.

completed. This is analogous to the minimum latency problem when $|A| = |T|$, as discussed in [Ngueveu, 2010]. If the assets have low value (θ = 1), the middle task no longer provides sufficient score to offset the risk incurred to visit it. For medium value assets ($\theta = 2$), it is better to risk only one agent in order to complete the two closest tasks. Finally, when the assets are of high value ($\theta = 3$), it is no longer feasible to send agents to any task, as the risk of losing the asset outweighs the value of completing the tasks.

Some general properties of higher value solutions to the collection planning problem are:

- Agents will prefer tasks that are closer to the base, as they have less risk. As θ increase, some tasks may be so far from the base that they are not worth the risk to complete them.
- Tasks that are close together will provide an increased value as a group, since the value of the tasks will exceed the total distance to travel between them.
- As the distance from the base increases, larger groups of closely grouped tasks are needed to offset the risk of traveling the distance required to reach them.

Observing these properties led us to look at clustering techniques for finding agent assignments. While most forms of clustering that we tested were able to quickly find clusters that were feasible solutions, hierarchical agglomerative clustering (HAC) (Cormack 1971) proposed task assignments with the highest observed values. This led us to explore new ways to adapt it more specifically to this unique problem. The resulting algorithm is discussed in the next section.

Clustering for Agent Assignments

Hierarchical agglomerative clustering (Cormack 1971) is a method for grouping instances in a metric space. We use a

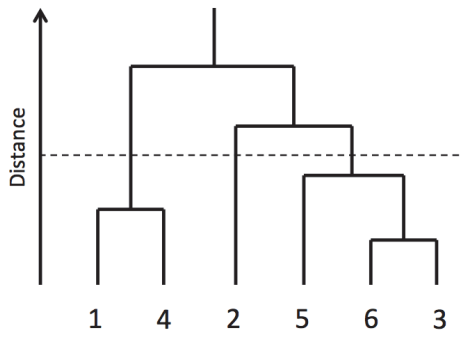

Figure 5: An example dendogram for six instances. Horizontal lines indicate a merge of two clusters, with the height along the the vertical axis being the distance between the clusters. The dotted line is an example of using a distance cutoff value to generate a set of clusters. In this case, the resulting clusters are $\{1, 4\}$, $\{2\}$, and $\{5, 6, 3\}$.

bottom-up approach, where each instance starts as a singleton cluster. Then, a series of successive cluster merges is executed, terminating when all instances are part of a single cluster. The result of this process can be visualized as a dendogram, as shown in Figure 5. By moving from the bottom to the top, we can reconstruct the series of merges that occurred and choose a specific distance at which to truncate the structure, resulting in a set of clusters. When a pair of clusters are merged, the distance to the other clusters is updated. There are a large number of strategies used for this update procedure, refer to (Manning *et al.* 2008) for further details and discussion.

This approach is appealing for this problem because it allows for a variable number of clusters depending on the problem structure and it evaluates groupings of tasks based on their proximity, which should also minimize inter-task distance. However, using only the distance between tasks as the similarity measure has some drawbacks:

- Combining task clusters (or singletons) into a single cluster may increase or decrease the utility, depending on how large the clusters are.
- The value of a cluster is also dependent on how far from the base the tasks are, which is not captured by the inter-task distance alone.
- There is no clear cutoff value for the distance at which we should stop the clustering process to yield the best groups of task assignments for all agents.

To address these challenges, we compute the distance between clusters as the gain in expected value form the merge. This yields clusters that are variable based on their size, distance between tasks, and the distance from the base. We refer to this clustering process as "risk-aware", with details provided in the next section.

Progressive Risk-Aware Clustering

We introduce an agglomerative clustering method for finding solutions to the collection planning problem with attrition-based risk. It provides solutions that exceed the performance of sequential greedy bundle-based methods with orders of magnitude less computation required. Unlike other clustering methods used for multi-robot tasking, we are not required to specify how many clusters we wish to form. This allows the problem instance and user parameters to guide the number of agents used.

We set the distance between clusters as the gain in expected value, which is a function of the numbers of tasks, the distance between tasks, and the distance of the task grouping from the base. We then iteratively combine tasks into clusters based on the gain in expected value until no positive gain can be found, or all tasks are in a single cluster.

Given a cluster $\mathcal{C}_i$, let π_i^* be the optimal path for agent i to visit all tasks in $\mathcal{C}_i$, starting and ending at the base. The optimal expected value for a cluster of tasks is:

$$EV^*(\mathcal{C}_i) = \prod_{e_{jk} \in \pi_i^*} p_s(e_{jk}) \sum_{t_p \in \mathcal{C}_i} c_{ip} - \theta_i \cdot (1 - \prod_{e_{jk} \in \pi_i^*} p_s(e_{jk})). \quad (7)$$

The exact gain in expected value from merging two clusters is defined as:

$$GAIN^*(\mathcal{C}_i, \mathcal{C}_j) = EV^*(\{\mathcal{C}_i \cup \mathcal{C}_j\}) - EV^*(\mathcal{C}_i) - EV^*(\mathcal{C}_j). \quad (8)$$

In practice, computing the optimal value can be expensive. Finding the optimal path through a set of tasks is an NP-complete problem, and the path would need to be computed for every possible merging of clusters. Even using agglomerative clustering with average-linkage has a complexity of $O(n^3)$, making the approximation of cluster value beneficial to scaling up to hundreds or thousands of tasks.

In order to approximate the value of a cluster, we keep track of the length of the minimum spanning tree (MST) over the tasks in the cluster. When two clusters are merged we can easily update the MST by adding only the closest edge between the two original MSTs. Finding the edge between the closest tasks can be done in $O(n^2)$ time. Additionally, we keep track of only the closest element to the base in the cluster, an $O(1)$ operation.

Let $dist(b, \mathcal{C}_i)$ be the distance between the base and the nearest element of cluster $\mathcal{C}_i$, and $MST(\mathcal{C}_i)$ return the distance of the minimum spanning tree. The approximate path length for cluster $\mathcal{C}_i$ is denoted as $\tilde{d}(\mathcal{C}_i) = MST(\mathcal{C}_i) + 2 \cdot dist(b, \mathcal{C}_i)$. Our approximate value function for a cluster is defined as:

$$\tilde{V}(\mathcal{C}_i) = |\mathcal{C}_i| \cdot \psi^{\tilde{d}(\mathcal{C}_i)} - \theta_i \cdot (1 - \psi^{\tilde{d}(\mathcal{C}_i)}). \quad (9)$$

Using this value function, we can compute our approximate gain for merging two clusters:

$$GAIN(\mathcal{C}_i, \mathcal{C}_j) = \tilde{V}(\{\mathcal{C}_i \cup \mathcal{C}_i\}) - \tilde{V}(\mathcal{C}_i) - \tilde{V}(\mathcal{C}_j). \quad (10)$$

The Progressive Risk-aware Clustering (PRC) algorithm (Algorithm 1) is shown below. The merging of two clusters is conducted as follows: $P[\,]$ is used to track the cluster parents as clusters are merged. Each cluster is represented by a single task, with all other tasks in that cluster referring to that task. $H[\,]$ is used to track merge height, and keeps track of what gain value caused the merge. It is used as an index to truncate the dendogram of tasks and form the final clusters, as explained in more detail in the next section.

The properties of a cluster are also tracked and updated, allowing for fast merging operations. $S[\,]$ indicates the number of tasks in a cluster. The distance to the base for each cluster is stored in $B[\,]$, which is updated as the minimum base distance over both merged clusters. Finally, $M[\,]$ tracks the distance over the minimum spanning tree for each cluster, and is updated by summing the merged clusters' MSTs, plus the shortest distance between a pair of tasks between the clusters. This shortest distance is equivalent to the single-linkage strategy for hierachical clustering.

The steps for generating the multi-agent plan (and their associated commands in Algorithm 1) are as follows:

- **Step 1.** [Lines 1-12] Initialize singleton clusters from tasks and associated data structures.
- **Step 2.** [Lines 13-22] Iteratively merge clusters until they form a single cluster of all tasks. At each merge track the gain in EV and update the properties of the new cluster and it's relation to other clusters.
- **Step 3.** [Line 25] Extract clusters from the resulting dendogram by returning clusters that were formed with a positive gain in expected value.
- **Step 4.** [Lines 26-30] Generate agent path from each cluster. Remove any agent paths that do not have a positive expected value.

In order to speed up this procedure we utilize ELKI (Achtert and Hans-peter Kriegel 2008), which provides data structures that are optimized for large-scale clustering, including fast indexing and iteration of distances between a collection of instances and optimized handling of the resulting dendogram data structure.

Algorithm 1 Progressive Risk-aware Clustering

Input: Task set $T = \{t_1 ... t_n\}$, graph $G =< V, E >$, where $V = T \cup \{b\}$
Output: Set of clusters $\{\mathcal{C}_1, ..., \mathcal{C}_m\}$.

1: $\mathcal{A} \leftarrow \emptyset$
2: **for** $i \leftarrow 1...n$ **do**
3: $\quad \mathcal{A} \leftarrow \mathcal{A} \cup \{\{t_i\}\}$
4: $\quad P[i] \leftarrow i$
5: $\quad H[i] \leftarrow \infty$
6: $\quad S[i] \leftarrow 1$
7: $\quad B[i] \leftarrow dist(b, t_i)$
8: $\quad M[i] \leftarrow 0$
9: $\quad$ **for** $j = (i+1)...n$ **do**
10: $\quad\quad$ dist[i][j] $\leftarrow dist(t_i, t_j)$
11: $\quad$ **end for**
12: **end for**

13: **while** $|\mathcal{A}| > 1$ **do**
14: $\quad \mathcal{C}_1^*, \mathcal{C}_2^* \leftarrow \underset{\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{A}}{\arg\max}\ GAIN(\mathcal{C}_1, \mathcal{C}_2)$
15: $\quad P[\mathcal{C}_1^*] \leftarrow \mathcal{C}_2^*$
16: $\quad H[\mathcal{C}_1^*] \leftarrow GAIN(\mathcal{C}_1^*, \mathcal{C}_2^*)$
17: $\quad S[\mathcal{C}_2^*] \leftarrow S[\mathcal{C}_1^*] + S[\mathcal{C}_2^*]$
18: $\quad B[\mathcal{C}_2^*] \leftarrow min(B[\mathcal{C}_1^*], B[\mathcal{C}_2^*])$
19: $\quad M[\mathcal{C}_2^*] \leftarrow M[\mathcal{C}_1^*] + M[\mathcal{C}_2^*] + dist(\mathcal{C}_1^*, \mathcal{C}_2^*)$
20: $\quad$ **for all** $C_i \in \mathcal{A} \setminus \mathcal{C}_2^*$ **do**
21: $\quad\quad t_i^*, t_j^* \leftarrow \underset{x_i \in \mathcal{C}_i, x_j \in \mathcal{C}_2^*}{\arg\min}$
22: $\quad\quad dist[\mathcal{C}_i][\mathcal{C}_2^*] \leftarrow dist(t_i^*, t_j^*)$
23: $\quad$ **end for**
24: **end while**

25: ExtractClusters()

26: **for all** $\mathcal{C}_i \in \mathcal{A}$ **do**
27: $\quad$ **if** $EV^*(\mathcal{C}_i) \leq 0$ **then**
28: $\quad\quad \mathcal{A} \leftarrow \mathcal{A} \setminus \mathcal{C}_i$
29: $\quad$ **end if**
30: **end for**

Performance and Evaluation

We evaluate the performance of PRC over a number of randomly generated problem instances. Tasks are placed in a 100km $\times$ 100km 2D Euclidean space, according to a uniform random distribution. The probability of survival is $\psi = 0.99$ for each km traveled. Each approach is evaluated based on the expected value (EV) of the solution as well as the CPU time required to find a solution.

To evaluate the performance of the PRC algorithm, we compare against the sequential greedy (SG) algorithm specified in (Choi *et al.* 2009), which is shown to provide solutions equivalent to those provided by CBBA. We use the gain in expected value for adding a new task as the scoring function. All experiments results are provided for 100 random instances for each parameter setting.

In order to allow for a fair comparison we provide SG with one agent per task. While the solutions generated by the SG algorithm will not necessarily use all of the agents provided, this allows the algorithm to maximize value. For example, when $\theta = 0$, the optimal solution is to assign one agent to each task. An agent that is not used provides no utility, but also incurs no risk of being lost, so there is no penalty or gain for any unused agents.

Because our problem has a score function that does not have a diminishing marginal gain (DMG) property (Choi *et al.* 2009) and allows for values less than zero, we also consider an alternative sequential greedy solver that bids only on positive gains in score when adding a task to a bundle. This sequential greedy gain (SG-Gain) algorithm performs significantly better on this problem than SG, as shown in Figures 7 and 8. SG allows for negative bids, assigning tasks to the agent that has the "least negative" score and effectively bidding on all tasks. Therefore, we use SG-Gain as the baseline for comparison for our experiments.

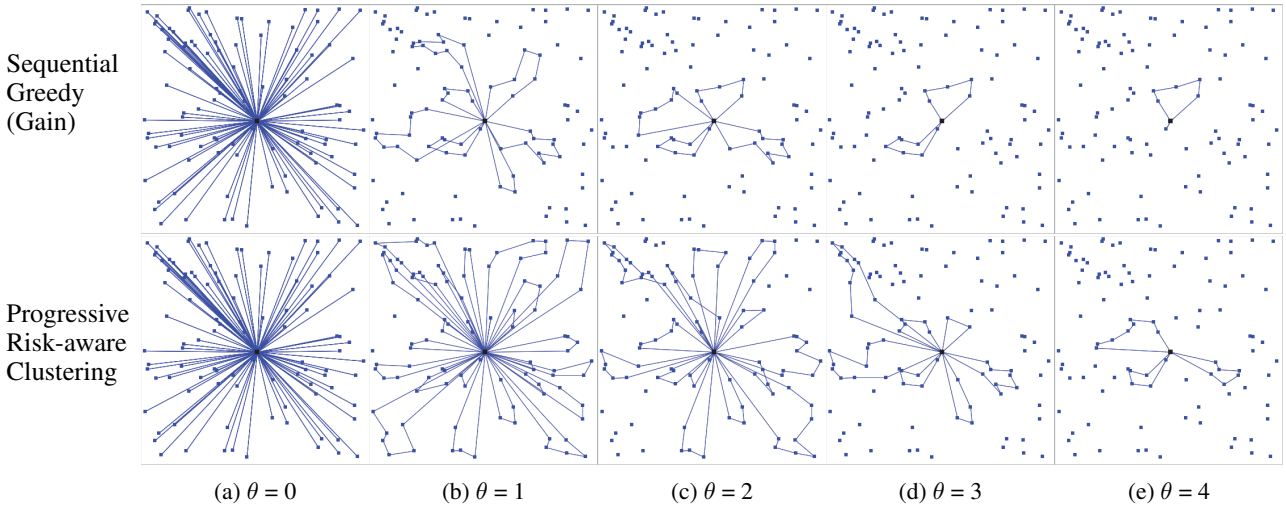

Figure 6: Comparison of solutions provided by Sequential Greedy (Gain) and PRC for a problem instance with 100 sites. The base is located in the center and the task collection sites are depicted as squares. Each line is a path for a single agent. Solutions are for $\theta \in \{0, 1, 2, 3, 4\}$, from left to right.

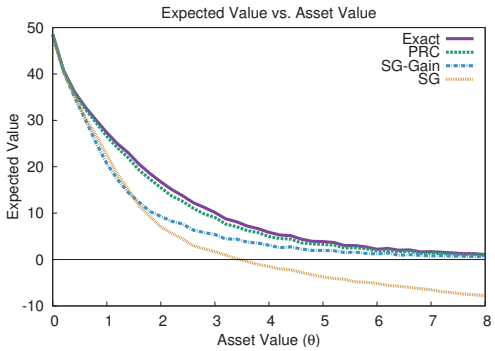

Figure 7: The expected value of multi-agent plans generated by each algorithm on 100 tasks. PRC achieves performance near Exact Merge, with improvement over SG and SG-Gain.

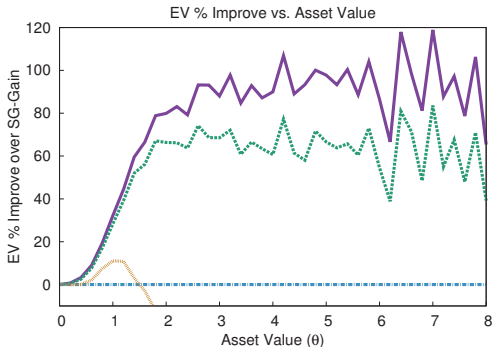

Figure 8: The % improvement over SG-Gain for increasing asset value with 100 tasks. SG quickly degrades in performance as the asset value increases, Exact and PRC show improvement over SG-Gain except when $\theta = 0$.

An example of the solutions generated by SG-Gain and PRC for a single 100 task instance with varying asset values are shown in Figure 6. When $\theta = 0$, assets have no value and the best solution sends one agent to each task. As θ increases, both approaches reduce the number of tasks that are assigned. Because SG-Gain only bids on positive gains, it fails to capture clusters where all tasks are far away. This is because the sequential bidding process is limited to generating values for one task at a time. Therefore, it does not bid on an initial task to begin forming the path, as it would require a negative gain (a loss in value compared to a value of zero for an empty path).

In Figure 7 we show the expected value on 100 tasks for PRC and SG-Gain compared to the Exact Merge Algorithm, (Algorithm 2) which uses the exact gain in expected value in order to evaluate cluster merges, rather than approximating the gain. This shows that PRC performs nearly as well as the exact merge solution, with improvement over SG-Gain.

Alternatively, we compute the percent improvement over SG-Gain for the same problem instances, as shown in Figure 8. We use this measure to show performance over a set problem instances with a variable number of tasks. With respect to CPU time, PRC provides these performance results at a significant computational savings in almost all cases, as shown in Figure 9.

In Figure 10 we show that the PRC algorithm scales well to large problem instances, with the gains in expected value increasing along with problem size. This increase in gain is due to there being more tasks outside of the threshold at which one task is worth completing.

In order to understand why PRC outperforms the greedy sequential methods, we can look at the properties of the solutions generated by each approach. In Figure 11, the number of task sites visited by each planner is shown. The Sequential Greedy algorithm will always collect on all of the tasks,

Algorithm 2 Exact Merge Algorithm

$\mathcal{A} \leftarrow \emptyset$
maxGain $\leftarrow -\infty$
for $i \leftarrow 1...N$ **do**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\{x_i\}\}$
end for
while maxGain > 0 AND $|\mathcal{A}| > 1$ **do**
    $\mathcal{C}_1^*, \mathcal{C}_2^* \leftarrow \arg\max_{\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{A}} GAIN(\mathcal{C}_1, \mathcal{C}_2)$
    maxGain $\leftarrow GAIN(\mathcal{C}_1^*, \mathcal{C}_2^*)$
    $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\{\mathcal{C}_1^*\}, \{\mathcal{C}_2^*\}\}$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathcal{C}_1^* \cup \mathcal{C}_2^*\}$
end while

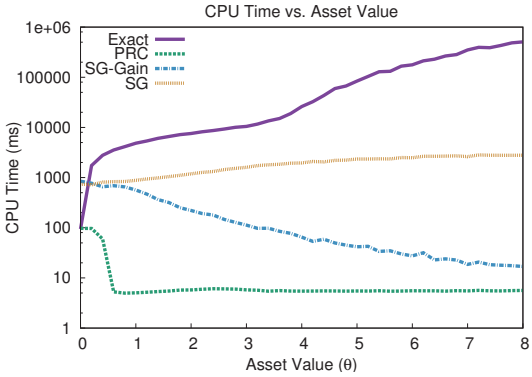

Figure 9: CPU Time in milliseconds for each algorithm with respect to asset value for 100 tasks. Note the logarithm scale used for the vertical axis.

which is detrimental when the asset value, θ, is large. The SG-Gain variant, which does not bid on tasks that have a negative utility, suffers from visiting too few tasks, resulting in the lower performance shown in Figure 7.

Finally, we show how many agents were dispatched by each algorithm in Figure 12. The sequential greedy approaches suffer from sending too few agents, missing out on opportunities for additional value. The SG methods choose individual tasks with the highest perceived value, which can discard tasks that may be part of a large cluster far from the base. On the other hand, PRC recognizes these clusters and determines their value, motivating sending an agent to collect and gain the associated value.

Conclusions and Future Work

We have introduced a new multi-agent task assignment problem with collection tasks and the risk of losing agents as part during execution. We outlined the fundamental aspects of the problem, provided clear examples of the problem dynamics, and implemented a risk-aware hierarchical agglomerative clustering algorithm that can outperform sequential greedy planning methods. Our results show that this algorithm not only provides good performance, but can also scale to thousands of tasks.

The clearest extension of this work is to allow for agents

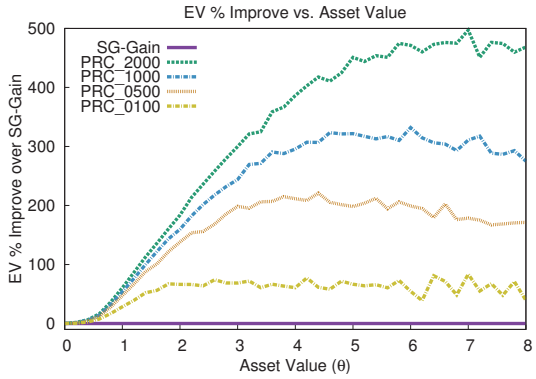

Figure 10: The % improvement over SG-Gain grows as the problem size increases. Results are shown for PRC on problems with the 100, 500, 1000, and 2000 tasks.

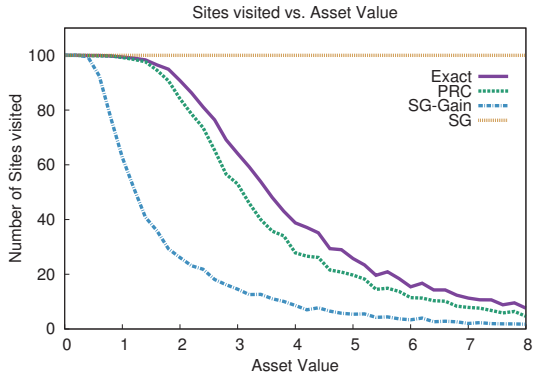

Figure 11: The number of sites visited by all agents with respect to asset value θ, for 100 tasks. Sequential Greedy (SG) is designed to visit all of the sites under all conditions, the SG-Gain variant over corrects, visiting too few sites.

to redundantly collect data to increase the expected value. Our model considers digital goods that can be collected by more than one agent, which provides opportunities for multiple collections to increase the probability of success. It's worth noting that credit for collection is only given for one instance of the collected data, and may expose agents to undue risk, so excessive redundancy should be avoided.

We use a simplified model of asset value and risk in order to present the problem clearly, but there would likely be heterogeneous assets with different value, and risk that would vary depending on location. Similarly, the risk of attrition may be a function of the tasks visited rather than movement through the environment. Finally, there is potential for the inclusion of perceiving sources of risk based on the detection of threats, requiring dynamic re-planning and negotiation.

Acknowledgements

This work was funded by the Air Force Office of Scientific Research under the Computational Cognition and Machine Intelligence program.

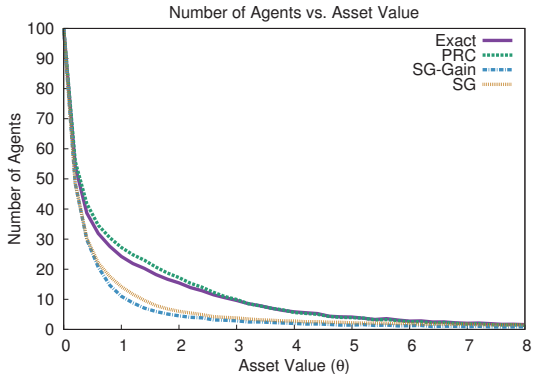

Figure 12: Number of agents used by each algorithm with respect to asset value θ, for 100 tasks. The Exact Merge algorithms provides better performance using less agents than PRC, but at the cost of CPU Time, as shown in Figure 9.

References

Elke Achtert and Arthur Zimek Hans-peter Kriegel. ELKI: A Software System for Evaluation of Subspace Clustering Algorithms. (Ssdbm):580–585, 2008.

Fayez F. Boctor, Gilbert Laporte, and Jacques Renaud. Heuristics for the traveling purchaser problem. *Computers & Operations Research*, 30(4):491–504, April 2003.

a. M. Campbell, D. Vandenbussche, and W. Hermann. Routing for Relief Efforts. *Transportation Science*, 42(2):127–145, 2008.

Han Lim Choi, Luc Brunet, and Jonathan P. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009.

Richard M Cormack. A review of classification. *Journal of the Royal Statistical Society. Series A (General)*, pages 321–367, 1971.

E. Erkut and V. Verter. Modeling of Transport Risk for Hazardous Materials, 1998.

B. P. Gerkey. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

Noam Hazon, Yonatan Aumann, Sarit Kraus, and David Sarne. Physical search problems with probabilistic knowledge. *Artificial Intelligence*, 196:26–52, March 2013.

Seungmo Kang and Yanfeng Ouyang. The traveling purchaser problem with stochastic prices: Exact and approximate algorithms. *European Journal of Operational Research*, 209(3):265–272, March 2011.

Emil Keyder and Hector Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, pages 547–556, 2009.

Michael JB Krieger and Jean-Bernard Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1):65–84, 2000.

Michail G Lagoudakis, Sven Koenigt, and Anton J Kleywegt. Simple Auctions with Performance Guarantees for Multi-Robot Task Allocation. 2004.

Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press, 2008.

E Nikolova, Matthew Brand, and DR Karger. Optimal Route Planning under Uncertainty. *ICAPS*, pages 131–140, 2006.

Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, February 2013.

Chelsea Sabo, Derek Kingston, and Kelly Cohen. A formulation and heuristic approach to task allocation and routing of uavs under limited communication. *Unmanned Systems*, 2(01):1–17, 2014.

Luca Talarico, Kenneth Sörensen, and Johan Springael. The risk constrained cash-in-transit vehicle routing problem with time windows. 32(0), 2013.

K A I Zhang, A Florida, and Emmanuel G Collins. Centralized and Distributed Task Allocation in Multi-Robot Teams via a Stochastic Clustering Auction. 2(3), 2010.