December 2016

Risk-Aware Planning for Sensor Data Collection

Jeffrey Hudack
Syracuse University

ABSTRACT

With the emergence of low-cost unmanned air vehicles, civilian and military organizations are quickly identifying new applications for affordable, large-scale collectives to support and augment human efforts via sensor data collection. In order to be viable, these collectives must be resilient to the risk and uncertainty of operating in real-world environments. Previous work in multi-agent planning has avoided planning for the loss of agents in environments with risk. In contrast, this dissertation presents a problem formulation that includes the risk of losing agents, the effect of those losses on the mission being executed, and provides anticipatory planning algorithms that consider risk. We conduct a thorough analysis of the effects of risk on path-based planning, motivating new solution methods. We then use hierarchical clustering to generate risk-aware plans for a variable number of agents, outperforming traditional planning methods. Next, we provide a mechanism for distributed negotiation of stable plans, utilizing coalitional game theory to provide cost allocation methods that we prove to be fair and stable. Centralized planning with redundancy is then explored, planning for parallel task completion to mitigate risk and provide further increased expected value. Finally, we explore the role of cost uncertainty as additional source of risk, using bi-objective optimization to generate sets of alternative plans. We demonstrate the capability of our algorithms on randomly generated problem instances, showing an improvement over traditional multi-agent planning methods as high as 500% on very large problem instances.

RISK-AWARE PLANNING FOR SENSOR DATA COLLECTION

By

Jeffrey Hudack
B.S. State University of New York at Utica-Rome, 2006
M.S. Syracuse University, 2010

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer & Information Science & Engineering

Syracuse University
December 2016

TABLE OF CONTENTS

LIST OF TABLES

List of Figures

1
INTRODUCTION

Unmanned aircraft are quickly becoming an affordable option for a wide range of applications, offering diverse capabilities without requiring human operators to be exposed to risk. As the cost and size of these aircraft are reduced, the number of these platforms being used is expected to increase dramatically, outpacing the availability of human operators to pilot these assets. As we move towards affordable, large-scale collectives of these unmanned aircraft we will need planning methods that will allow the platforms to operate without continuous, direct control.

A key element of conducting operations in real world environments is the presence of risk and uncertainty. In a military context, we often wish to operate in hostile environments with adversarial threats. For humanitarian aid, resources are dispatched to regions where natural disasters, such as damaged buildings that could fall, pose a threat. Even civilian applications are exposed to risk of equipment malfunction, weather, or other natural phenomenon. These challenges can make planning difficult, and require strategies that are robust to the presence of risk in the environment.

In this dissertation, we explore *anticipatory risk-aware planning*, which generates robust plans that will not be altered after execution begins. These plans will provide a series of actions that maximize expected value, with consideration for the probability of detrimental

events occurring. Rather than treating the loss of agents as a catastrophic failure, we plan with the expectation of losing some platforms to these events and continuing operations without interruption. In addition, we provide planning methods that are flexible enough to allocate a variable number of aircraft, in response to the value of the tasks provided by the user, and the expected risk in the operating environment.

This work spans various fields of research across multiple communities of interest. Development of the initial plan has similarities to the multi-vehicle routing and team orienteering problems that have been widely studied in the operation research community. Artificial intelligence research has shown much interest in algorithms for the maximizing the value for assignment of agents to tasks. Finally, the game theory community seeks mechanisms to describe interactions between self-interested agents, providing formal guarantees on the outcome of complex negotiation. We draw from each of these fields to provide a thorough examination of the effects of risk on path-based planning, offering new solution methods for each facet that we explore.

1.1 Risk-Aware Sensor Data Collection Problem

We start with a set of collection tasks to be serviced, each of which has a location specified by a user. Tasks are serviced by unmanned air vehicles that are dispatched from a base station that contains a large number of these vehicles. Each task has a value, which is gained only when a vehicle visits the task location, collects the data, and then returns to the base station. Each vehicle also has value, which is incurred as a cost if the vehicle is lost during a mission. The goal is to maximize the total value of the mission plan by specifying a path for a number of air vehicles to travel, each visiting a series of collection tasks along the way and returning to the base with all collected data.

Maximizing the value of collection is complicated by the presence of risk, which can lead to losing the air vehicles before they are able to return to the base. Losing a vehicle

has a number of effects:

- Any data that was previously collected is lost with the vehicle, preventing the value of those tasks from being gained.
- Tasks that were planned for collection after the loss of the vehicle will not be collected, forfeiting their value.
- The value of the vehicle is incurred as cost, further reducing the total value of the mission.

The total value of the mission is represented as *expected value*, which is the sum over the value for each air vehicle's collection tasks times the probability of successfully completing the path with that aircraft. We seek to specify mission plans that maximize expected value for a given set of tasks and the value of the air vehicles used. This requires solving a set of intertwined problems, which include assigning tasks to vehicles, constructing paths between those tasks, and balancing the risk of collection with the rewards for task completion.

Consider the analogy of hunters tasked to collect food for a village, requiring visiting one or more known hunting spots, and returning to the village with the food. The landscape contains predators that will strike at random and kill the hunters, an event that results in the loss of the hunter and all of the food that was gathered and not yet returned to the village. The inhabitants of the village wish to maximize the amount of food that is collected, but want to minimize the risk of losing trained hunters, who have invested time into learning to hunt effectively.

Having lived in this village for many years, the village elders have observed the behavior of the predators. They know that on hot and sunny days the predators often seek a shady place to sleep, waiting for cooler weather. Under these conditions, the risk to the hunters is low and many can be dispatched to gather as much food as possible. On cloudy days and at dusk the predators are more active, increasing the risk to the hunters that leave the village.

During these times, hunters should be sent on short missions to nearby hunting locations before quickly returning to the safety of the village. In the dead of night, the predators are at their most active, and hunters should avoid leaving the village, else risk the likely outcome of being a late night snack.

The location of these predators is not known, and their attacks are silent and without warning. Therefore, we can represent the likelihood of being attacked as a probability based on distance traveled. During the sunny days, the hunters may have a 1% chance of being attacked for every mile they travel, while at night their probability of being attacked is a terrifying 50% for every mile traveled. We explore the implications of these probabilities and their effects on the planning process, providing general purpose methods that can generate mission plans when provided information about task values, the values of the collection assets, and the probability of an asset being lost.

We now discuss a number of real-world applications that can benefit from considering risk as part of a multi-vehicle path planning process. Unlike the hunting scenario we described, we assume that the agents are collecting sensor data, requiring it to be stored on board the asset until it can be delivered to a base station. Each problem has unique challenges and objectives, but there are common elements of moving through physical space with inherent sources of risk in the environment.

1.2 Applications

The goal of this work is the completion of 'collection tasks', an intentionally abstract concept. In the course of the work presented in this dissertation, we keep in mind a number of applications of interest. The rapidly increasing availability of cost-effective air vehicles has provided new potential for applications that have not yet been realized. We expect that the applications discussed here will serve as a starting point for using large collectives of these vehicles in real world environments with inherent uncertainty and risk.

Each application requires assigning a collective of vehicles to collection tasks that are located in a physical environment, requiring both allocation of resources tasks and generation of paths between the tasks. We assume that communications are limited, often due to small size of the platforms and the large distances that must be covered. Additionally, each application has an element of risk that needs to be considered. In general, we assume that these applications are completed by multi-rotor air vehicles, allowing for straight paths between locations, and more vulnerable to airborne threats, such as weather. This work could be applied to other types of vehicles, such as ground vehicles with known road distance between locations, but these applications are not discussed here.

1.2.1 Utility Inspection

There is massive infrastructure required to provide utilities for both established and developing nations. As the scale and age of these systems increases, it becomes more difficult to to ensure consistent and reliable supply. Interruptions in power can not only provide inconvenience, but also cause serious financial losses to industry. One method for addressing this challenge is to dispatch mobile robots, which can provide constant observation of vital locations, detecting faults before they occur or finding damaged infrastructure after an event [56].

Fig. 1.1: Drones have the potential to provide persistent monitoring of power infrastructure, allowing for constant surveillance to detect current and potential faults.

There are a number of valuable roles that a collective of unmanned air vehicles could serve in this domain. The inspection of power lines and equipment can often only be performed within close view to detect anomalies, a task that can be dangerous and time-consuming for human technicians. Additionally, there is a need for monitoring of the safe distance that should be maintained from power equipment, providing safe operating parameters for other activities being conducted nearby.

Inspection tasks would be generated by faults that are detected at the control stations, routine maintenance, or damage reports that are called in. Collection could be performed using imagery or measuring the electrical power, and the vast distances traveled may require returning to a base location that can allow for communication of the collected data back to users. The electromagnetic field surrounding power equipment can be dangerous, especially when equipment is damaged, providing an element of risk for the vehicles as they conduct inspection. Conversely, these fields could also be exploited to provide recharging, which has interesting implications for the other applications mentioned here, which may want to integrate recharging into the planning process.

1.2.2 Crop Monitoring

With an ever-increasing demand for food, the agriculture industry is constantly looking for new way to automate processes and increase production. Currently, many crop monitoring applications utilize satellite imagery, which suffers from long delays between image acquisition [61] and the difficulties of acquiring the imagery in a timely manner from the producer. There is great interest in using collective of mobile robots to provide high quality imagery of crops on demand, especially in regions with weather conditions that may impede satellite-based monitoring. These methods can be used to better manage weed control and the application of water and nutrients [68].

There is also interest in using mobile robots to help harvest crops that are traditionally harvested by hand [48], requiring significant cost and human workload. For example,

Fig. 1.2: A helicopter drone used by Dr. Charlie Rush, Texas A& M AgriLife plant pathologist in Amarillo, flies over a wheat field to track disease progression. (Texas A& M AgriLife Research photo by Kay Ledbetter)

coffee blossoms do not develop regularly throughout a plantation, but rather have unpredictable patterns of maturity that may be localized. Mechanical harvesting of an entire field will likely yield unripened and overipened product, and methods direct harvesting are of significant monetary value.

Collectives of unmanned vehicles offer the potential to perform long-endurance, persistent collection of imagery to assist in managing large crops. Tasks can be generated from satellite imagery with low resolution, requiring platforms to get a closer look in order to increase accuracy. As the technology progresses, we can foresee more sophisticated capabilities, such as the ability to administer water or fertilizer, detect and remove weeds, or even provide harvesting of crops at their optimal ripeness.

The agriculture industry is not without risks, however. Weather and wildlife pose a threat to the vehicles, such as birds that may perceive these airborne intruders as a threat, or strong winds and rain arriving unexpectedly. All of these applications will require multi-agent planning capabilities that are robust to these uncertainties found in the real world.

Fig. 1.3: The ‘cockpit’ of a Predator drone, relying on a team of operators to conduct missions for a single platform. Extending this control model to hundreds or thousands of UAVs without autonomy would require a massive workforce and complex coordination.

1.2.3 Military Intelligence, Surveillance, and Reconnaissance

Currently, an unmanned vehicle conducting collection for and Intelligence, Surveillance, and Reconnaissance (ISR) mission requires a team of operators to fly the platform, monitor the visual input, communicate observations, and coordinate with other operations. While this is proving to be an effective way to extend our reach without risking human life, operating a large collective of platforms would be prohibitively expensive, time-consuming, and complex. It is difficult to imagine 100 platforms being managed by a team of hundreds operators that are all coordinating effectively as the mission is being executed. Additionally, in environments where communication with platforms is limited or infeasible this level of direct control is simply not possible.

In this context, the tasks are generated by users that have collection requests they wish to have fulfilled. These requests are compiled by the controlling organization and are used to generate paths for the platforms. In more complex scenarios, the distance between tasks may be altered to reflect known threat regions or areas where flight is prohibited. The limitations on long-distance communications, paired with conducting operations that may require remaining undetected, require the platforms to return to the base to submit collected

Fig. 1.4: The Aeryon Scout quad copter, weighing 1.3kg, is an example of a small aircraft with onboard electronics that could support autonomous flight and tasking of collection.

data.

While there is increased interest in utilizing collectives of unmanned air vehicles (UAVs) to enable autonomous operations [92], there is uncertainty that is introduced when operating in adversarial environments. This uncertainty may arise from directed threats from an adversary, the possibility of malfunction in harsh conditions, or general civilian aggression towards drones. Additionally, requests are serviced from a number of organizations, requiring planning for many collection tasks that may be in regions of significant danger.

This difficulty motivates the use of autonomous vehicles that can meet the needs of the mission while being resilient to threats that arise during execution. In high threat environments we would expect that some or all of the platforms may be lost, requiring that our investment not be focused on a single large aircraft, but rather a collective of small, low-cost vehicles that can meet mission needs while still being expendable.

With the increased availability of relatively low-cost consumer aircraft, such as quad copters, we foresee new opportunities for ISR missions to be executed without needing expensive aircraft. This frees up more advanced platforms to be tasked for other missions requiring their advanced capabilities. These new assets have the potential to be a significant force multiplier across the ISR enterprise, requiring the development onboard software systems that ensure missions are executed safely and can react to unexpected changes in the area of operation.

Fig. 1.5: A drone captures imagery of building damage in Nepal following an earthquake. Photograph: Olivia Harris/Reuters

1.2.4 Disaster search and Rescue

When natural disasters occur, there is a need to quickly divert resources to multiple locations in order to provide timely discovery and rescue of injured persons. Because of the widespread destruction, lack of preparation, and the number of disjoint agencies involved, it can be difficult to effectively prepare and distribute resources appropriately. In these situations, having up-to-date information about the damage and potential for rescue is vital, and can directly effect the potential to save lives. Given the high demand on personnel, we seek solutions that can plan and dispatch mobile sensors with minimal human support.

Collection tasks would be generated by analysis of satellite imagery, calls made to emergency services, or knowledge of buildings where a high number of victims are likely to be located at a given time of day. Agents can be tasked with capturing imagery of collapsed buildings to aid in coordinating equipment, or collecting infrared or audio that can help determine where survivors are trapped. Because of damage to communications infrastructure, each platform would need to return to a base station to submit the collected data for processing.

Disaster management has already gained some momentum as an application for unmanned air vehicles [2]. Specific uses include structural damage inspection, overhead imagery, and monitoring of emerging weather conditions that could endanger rescue op-

Fig. 1.6: Drone images captured of temporary housing for disaster survivors in rural Kenya. Photograph: Trygve Bolstad/Panos

erations. The need for well-defined and realistic challenge problems using multi-agent planning for disaster scenarios has been noted [84], which should motivate further work in this domain.

1.2.5 Drone Journalism

Unmanned air vehicles provide new opportunities for journalists to report on stories in remote and dangerous locations. In addition, aerial coverage provides a sense of scale to the viewer that can often not be captured from the ground. The field of journalism has already started to embrace the use of drones in high-risk environments [26] such as embedding with military operations and covering disasters that prohibit reporters from entering a dangerous area.

However, not all governments support these activities [29], and there is a risk of losing the drones when reporting on activities that a ruling government would prefer to keep hidden. Additionally, the subject of the reporting may not be cooperative, and there is risk of the aircraft being attacked by those who do not wish to be seen. Journalists already embrace these risks as part of their job, and extending the risks to unmanned vehicles provides a growing area of research.

1.3 Terminology

We now define some of the common themes across this body of work. We do not claim to be an authoritative source of how these terms are defined, but rather seek to provide the reader with clarity regarding our problem exposition. Given the large body of work addressing multi-agent path-based planning, we feel it is necessary to assert our interpretation of these terms.

Definition 1.3.1. *The objective of this work to maximize **expected value**, which is the probability-weighted average over all possible outcomes.*

The value for this work is derived from the completion of tasks and the loss of collection assets in the presence of risk, all of which will be defined below. We do not specify *how many* agents to use in a problem instance, but rather seek solutions that specify the number of agents that will maximize the expected value.

Definition 1.3.2. *An **agent** is the software used to control a physical or simulated **asset**. For our purposes, these terms are interchangeable.*

In this work, we consider the planning process to be a one-time activity, with the result being a set of plans given to each individual agent via communication while located at the base station. After the agents have been dispatched, they are no longer given commands by the base, but rather must execute the plan given to them.

Each agent is deployed from the base, and must eventually return to the base to complete the mission. If an agent is destroyed or disabled, it is assumed to be permanently lost. Planning for the loss of assets is one of the primary contributions of this work.

Definition 1.3.3. *A **collective** is a set of homogeneous agents executing a behavior to attain macro effects.*

Homogeneous platforms provide a number of practical benefits. Scaling the collective requires only increasing the number of agents, without requiring the user to modify

how the software works. Homogeneous hardware configurations makes also enable mass production of assets, allowing for rapid construction of vehicles with interchangeable parts.

We consider problems that may requiring large collective of platforms, sometimes on the order of hundreds or thousands of agents. A vast majority work in multi-agent planning and coordination do not exceed tens of agents, often due to the complexity of planning, modeling and executing. We consider our ability to plan for large collectives of agents as another contribution of this work.

Definition 1.3.4. *A **base** is a fixed installation or large platform that is used to deploy the collective in a region.*

Unlike the agent platforms, that are relatively small and have limited capability, the base has significantly more resources and is used to house the collective of vehicles and store any information they may collect. While we do not specify exactly how the base is deployed, we envision it could be a building or even a large truck or plane that has been relocated to the area of operations. It is assumed that the base has sufficient storage for the air vehicles and the capability for refueling/recharging prior to their use.

Definition 1.3.5. *A **collection task** is a request for collection of sensor data to be delivered to a base station.*

Collection tasks are assumed to be provided before the agents are deployed, and are generated to meet the objectives of user(s). Completing a task requires capture of the sensor information and subsequent delivery to the base, which can be achieved by one or more agent. Tasks are assumed to be unique, assuming any redundant requests are deconflicted before tasking of the agents begins. A task can be completed only once, and if another agent completes a collection task redundantly no additional credit is given. The generation of plans with redundant collection will be discussed in Chapter 3.

Definition 1.3.6. *A **task location** is a physical point of interest an agent can visit in order to fulfill a collection task.*

In this work, each task location represents exactly one collection task, and vice versa. We assume collection is instantaneous, such as capturing an image, and meeting the collection requirement is achieved when the agent is located at the same location as the task. The locations are assumed to be in a 2D Euclidean space, but the methods presented are flexible enough to be used in higher dimensional or non-metric spaces.

Definition 1.3.7. *A **path** is an ordered series of task locations traveled by a single agent.*

Each agent follows a path assigned to them, visiting the task locations and collecting the task data. A path can be characterized abstractly by a cost, which can represent time, fuel costs, or any resource of value being expended. We also assign risk to each path, which has effects on both the expected value of the tasks in the path, and the risk to the asset.

Definition 1.3.8. *A **plan** is a collection of paths for all agents in a swarm prior to execution.*

A plan is formed based on the knowledge and perceptions available before the agents are deployed. Once execution begins, the agents follow their respective path until they are destroyed or return to the base unharmed. Because we allow for the deployment of a variable number of agents, the number of paths in a plan is not known prior to the planning process. Agents that are not dispatched on a path remain at the base, incurring no cost.

Definition 1.3.9. *The presence of **risk** implies there is a chance of losing one or more things of value.*

In this work, value of tasks and assets is quantified in some way that is measurable, although we acknowledge that value is subjective measure that may not be agreed upon by all involved parties. This value must also be comparable, such that we can define a partial order on the value of all things of interest in the problem space. For our purposes, we assume that this is a numeric measure.

We focus on the loss of (or lost opportunity to collect) sensor data, and the loss of physical assets used for collection, but this does not limit applicability of our models to

other losses that could be incurred. For example, we may wish to measure the cost of time for completing a task, or the cost of losing something more ephemeral, such as the trust of a customer.

Definition 1.3.10. *Losing members of the collective is referred to as* ***attrition****, a form of risk.*

In much of this dissertation, we will focus on attrition risk, which is a chance of losing one or more agents (and the controlled asset) performing collection of sensor data. When an agent is lost, we lose not only the sensor data that has been collected, but also the data that was planned for collection after the loss occurred. In Chapters 3 and 4 we provide various methods to mitigate this risk.

Unmanned vehicles represent a major shift in how planning methods may deal with risk. Tasks issued to manned vehicles must severely minimize the exposure to risk to avoid the loss of invaluable human lives. In contrast, unmanned vehicles have a cost that can be measured as a quantity of money or time to manufacture. In this context, resiliency can be measured as a trade-off between cost and value as interchangeable commodities.

1.4 Quantifying Risk

We define risk as the possibility of not achieving one or more mission objectives, or achieving the mission but with unnecessary use of resources. Without considering risk, we could think of sensor data collection as a traditional planning problem and expect that everything will proceed without complication. A vast majority of path-based planning problems make this assumption, making this one of the key contributions of this work.

We assume that risk is derived from events that are considered to be out of the control of the agent. Given knowledge of the probability of these events occurring, we can choose a series of actions that will strike a balance between minimizing risk and maximizing the value gained. The knowledge of the probabilities can be derived from historical data, such

as records of equipment failure, or more complex models, such as a map of hostile forces and their capability for destruction. Because the occurrence of these events is random, we can measure the expected value, which multiplies the value of a set of actions times the probability of successful completion of the actions.

Passive threats are not intentional, but rather a by-product of an environment, such as weather, or other natural phenomena. This includes the possibility of equipment malfunction due to unexpected defects, 'acts of god' such as lightning and hail damage, or even animals attacking the aircraft. We assume that any damage to the asset is unrecoverable, resulting in loss of the asset.

Active threats are caused by one or more adversarial entities making decisions based on perceptions of the environment. For example, an anti-air battery may attempt to eliminate any aircraft entering sensor range, making a specific region a no fly zone. Alternatively, a threat may emerge only when it perceives overt activities, such as flying lower to collect sensor data, or communication between platforms.

In this work we model risk as a uniform probability of failure based on movement, and risk is based time and distance of exposure. This is similar to the concept of mean time to first failure [47], which is used to characterize the reliability of non-repairable hardware. This general model assumes that the agents have no reliable means of perceiving the threat and may fail without warning. This could represent an active threat that is unknown, with a probability of being attacked anywhere in the area of operations. More complex threat models are compatible with our solution methods, as long as each edge in the graph can be assigned an accurate assignment of risk.

1.4.1 Assumptions

Throughout the work discussed here, we assume that the probabilities underlying the risk model are known and fixed after planning has occurred. Clearly, if these values change between planning and execution, then the computation of expected value can not be expected

to hold. We acknowledge that having full probabilistic information about all of the risks in an environment may be unrealistic, but we seek to explore the fundamental properties of problems with probabilistic risk with the expectation of extending these definitions as they are appropriate for more specific applications.

It is also worth noting that we consider all of our probabilities to be independent variables, with no conditional relationships between them. There are a number of interesting applications that would require modeling conditional probabilities, but we do not explore those alternatives in this work. For example, in an adversarial environment, collecting data at a task location may cause alarm and increase the risk to the platform for subsequent travel. Such scenarios are of great interest, and serve to provide a rich set of problems for analysis in future work.

1.5 Related Work

This work is derived from immense bodies of work in operations research and artificial intelligence, many of which help to guide our problem definitions and solution methods. Starting with the original Traveling Salesman Problem (TSP), which seeks the shortest route visiting a set of cities in a road network, there have been countless extensions and variations. This work derives inspiration from diverse bodies of work in Multi-Robot Task Assignment (MRTA) [42] and Vehicle Routing Problems (VRP) [76]. 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 these problems, the number of agents to be used is specified a priori, the expectation is that all tasks will be completed as part of a valid solution, and limited consideration is given to losing agents during execution. The collection planning problem introduced here relaxes these constraints, motivating new solution methods.

Table 1.1: A sample of research in single and multi-agent routing problems with various models of risk.

	Single-Agent		Multi-Agent	
Risk	**Task**	**Commodity**	**Task**	**Commodity**
None	path-TSP [4] Hamiltonian Path [12]	TSP [78] Hamiltonian Circuit [40]	MRTA [43] MR Coverage [67]	mTSP [10] VRP [59]
Constraint-based	MLP [18] Pathfinding [87]	TPP [94] PC-TSP [37] TSP-TW [83] Orienteering [93]	MRTA [43] MAPP [95]	RCTVRP [89] PC-VRP [91] MVTPP [16]
Stochastic Event	GSP [82]	Stochastic TSP [75] Canadian TSP [62] PSP-PK [46] SPSP [21]	Multi-Robot Search [49]	Stochastic VRP [41] TPPSP [55]
Attrition	Collection Planning Problem with Attrition Risk (CPPAR)			

Our primary focus in this work is to explore *anticipatory* methods for addressing risk during the planning process, maximizing our expectation of value prior to dispatching physical platforms to perform collection. The assumption is that communication is limited and agents are not able to perceive and reason over objects or revents in the environment. This shares similarities to conformant probabilistic planning (CPP) [63, 52], which assumes that the state can not be observed while the plan is begin executed, and robust optimization [11], which assumes some probability distribution over costs. However, we are not aware of any existing work in these fields that plans for the loss of actors as part of the planning process.

We provide a sample of the foundational and related problems in this dissertation in Table 1.1. We make a distinction between single and multi-agent problems, the type of actions required, and means by which risk is modeled. For actions we consider *tasks*, which are completed by the arrival of the agent to a location, and *commodities*, which require that the agent transport some object or resource from one location to another. We also categorize each problem's treatment of risk, which we will describe in more detail, starting with routing problems that do not consider risk.

1.5.1 Routing without Risk

Combinatorial optimization for routing has a long history, starting with the well-known Traveling Salesman Problem (TSP [78], which seeks the shortest distance cycle visiting all vertices in a graph. The TSP is thought to have originated in the 1800s before being formalized in the 1930s, and still remains a challenging problem that gets research attention today. The simplicity of the problem description, coupled with the diabolical difficulty of finding a solution in the general case, has made the TSP a canonical NP-hard problem.

If we consider only tasks with immediate reward at each vertex, then the cycle solution is unnecessary and a path that visits each vertex will suffice. This problem is known as the path-TSP [4], or a Hamiltonian Path [12]. Because many TSP solvers rely on the cycle constraint to find solution, allowing for a path that visits all vertices will often make the problem more difficult to solve [4]. If the salesman is collecting a commodity at each vertex and returning them all to the starting location, this is equivalent to the TSP. This is equivalent to finding a Hamiltonian Circuit [40] in a graph, and these terms are often used interchangeably.

For multiple agents completing tasks with immediate reward, the problem specifies the number of agents that are used ans seeks a solution that minimizes the sum of all agent paths. Variations of this problem exist in the field of Multi-Robot Task Allocation (MRTA) [43], which seeks to have a fleet of robots complete tasks in an environment. Similarly, the term Multi-Robot Coverage [67] refers to problems where robots must visit all locations, usually in a grid environment.

For commodities that must be returned to the start location, the multiple TSP [10] is a simple extension of the original TSP to a collective of salesman. A more robust and well-studied problem is the Vehicle Routing Problem (VRP) [59], where vehicles must purchase and deliver items from different sites to complete the tasks and earn a reward.

In each of these fundamental problem, risk is not considered. The focus is on optimization of distance and cost, with the large search space often being the primary challenge.

Countless solution methods exist, ranging from heuristic methods to linear programming. There are cases in which risk is referenced, characterized as a constraint on an additive quantity. In the next section we will discuss the benefits and shortcomings of this approach.

1.5.2 Constraint-based Risk

It is not uncommon to find problem formulations that quantify risk as a fixed value on cost or edges, allowing solvers to treat risk as an additional additive constraint. In many cases, the problem will specify a maximum risk for a valid solution, allowing only solutions that fall under this threshold. This is an effective way to use existing solvers to also minimize risk. A key benefit of constraint-based methods is that they provide bounds on the solution space, allowing us to eliminate portions of the search space that violate constraints.

There are a wide range of other constraints found in routing problems that could also be leveraged to model risk. The distance of an edge could be directly equated to exposure, making shorter paths result in reduced risk. For a single agent completing a task with an immediate reward, this is similar to the process of pathfinding [87], which seeks the shortest path from the source to the destination. For multiple tasks, the Minimum Latency Problem (MLP) [18], also referred to as the Traveling Repairman Problem, is a customer-centric model that encourages the a solution that minimize the average distance each task being serviced.

When a task requires transporting a commodity to some destination, the distance constraint must consider the paths to pickup and drop off the commodity. Minimizing distance for this two-stage journey is a common theme among route planning problems. The Traveling Purchaser Problem (TPP) [94] is a generalization of the TSP, specifying pickup and delivery of items while minizing total distance. The Prize Collecting TSP (PC-TSP) [37] and the Orienteering Problem [93] are instances of the TPP that seek to visit as many tasks as possible given a constraint on the maximum distance traveled. Both seek to also maximize the value of the tasks completed, which can yield a multi-objective optimization

problem depending on the model used.

Another popular constraint is the addition of time windows, which specify a range of time that a site can be visited. For tasks, this is a single range for getting immediate credit, while commodities may have different time windows for pickup and delivery, or for different types of commodities. A straightforward example of this is the TSP with Time Windows (TSP-TW) [83], which requires not only minimizing path length, but also respecting the time window constraints. If the time windows are based on periods of time when there is lower risk to perform pickup or drop off, then respecting these constraints could provide a mechanism for minimizing risk.

The above methods are equally effective when there are multiple agents. There are a number of problem formulations with similar constraints that fall under the umbrella of Multi-Robot Task Allocation (MRTA) [43]. One example is Multi-Agent Path Planning (MAPP) [95], which places constraints on how close multiple robots can be in order to minimize risk of collisions. For multiple agents, one example is the Risk-constrained Cash-in-Transit Vehicle Routing Problem (RCTVRP) [89] involves moving cash in armored trucks between locations. Each road the trucks may travel has a risk value assigned to it, and a constraint is placed on the maximum sum of risk that can be tolerated. Similarly, the transport of hazardous waste through populated areas can quantify risk by the number of people affected on a roadway if the transport were to be involved in an accident [32].

A major drawback of constraint-based measure of risk is that they make it difficult to reason over attrition as an event that affects the tasks on a path. Consider a constraint that specifies a vehicle should travel no more than 20 miles, as it yields at most a 10% chance of losing that vehicle. For a critical task, 10% may be an unacceptable risk that could result in catastrophic failure. This would require having constraints that change based on the tasks that are selected, preventing us from setting clear boundaries. Without these boundaries, risk is no longer a constraint and is part of the objective function, eliminating the benefit of being able to reduce the search space.

These challenges motivate us to preserve risk as probabilities, including them in the objective function. There is a body of emerging work that includes probabilistic knowledge in the route planning process, which we have leveraged in this work. We consider this work to be most closely aligned with our research.

1.5.3 Risk as a Stochastic Event

When we have events that can be modeled with stochastic variables we can characterize a solution by using the expected value, which is the average outcome over an infinite set of samples. This form of probabilistic evaluation is the foundation of characterizing reliability for physical systems, leading to approaches such as series-parallel systems for fault tolerance [88]. Similarly, we seek solutions that are tolerant to losing assets due to events that can be characterized as failure event probabilities.

An early model of routing for stochastic events is the graph Search Problem (GSP) [82], in which an agent is searching a graph to find an object of interest. At each vertex in the graph, there is a probability that the object is located there, which is realized the first time that agent visits the vertex. The goal is to find a path that maximizes the probability of finding the object as soon as possible. Multi-Robot Search [49] is the multi-agent extension of the GSP, with a number of variations that specify constraints on the environment or the robots.

For the collection of commodities, where the agent must return to some location after completing tasks, there are a number of problems that have probabilistic elements. The Stochastic TSP [75] extends the original TSP with lengths between cities that are drawn from independent, identically distributed random variables. Similarly, there are models that consider the randomness of traffic flows on the speed of travel over roads [72]. The Canadian TSP [62], which considers traveling in heavy snows with unpredictable road closures, takes this a step further. Each edge between cities has a probability of actually being in the graph, with the existence of an edge being observable only when you reach a

city connected to it.

The cost of acquiring the commodity can also be stochastic, introducing the risk of having insufficient resources to complete high value tasks. In the Physical Search Problem with Probabilistic Knowledge (PSP-PK) [46], there is a shared resource that is used to travel on a path and purchase a commodity, and each vertex has a stochastic cost for acquiring the commodity. In this problem excessive search for a low cost can exhaust resources, while purchasing too early may miss out on a lower cost. Similarly, the Stochastic Physical Search problem (SPSP) [21] explores this problem for general graphs.

The Stochastic VRP [41] is one of the earliest problem definitions with multiple agents gathering commodities with stochastic problem components. This model has been extended to consider randomizing the cost and existence of commodities, such as the Traveling Purchaser Problem with Stochastic Prices (TPPSP) [55]. These probabilistic models present risk in the form of limited resources, seeking to minimize the expectation over multiple trials.

1.5.4 Attrition-based Risk

While the previously mentioned problem definitions capture risk in a probabilistic form, stochastic costs are "randomized inconvenience", driving the cost of execution up for a specific instance but without risking failure of objectives. In this work, we explore how to plan for attrition, which involves losing the agent itself due to a stochastic event. This event can have affects on actions already take by an agent, and remove the ability to perform collection action in the future. For tasks with immediate reward, losing an agent means that any remaining tasks will not be completed. When an agent is collecting and delivering commodities, we risk losing everything that the agent has collected so far.

Because of the lack of work dealing with attrition, we have devoted much of this dissertation to exploring fundamental effects of probabilistic risk of losing agents to combinatorial optimization problems. This focus is driven by realistic scenarios where platforms

will be dispatched into the real world, vulnerable to equipment failure, external threats, or random acts of nature. We hope that this work will lay the foundation for considering the risk of losing assets for a wide range of planning problems.

1.6 Dissertation Overview

In this dissertation, we will provide an examination of implications of attrition risk on optimization problems. We start by applying risk to a set of simple knapsack problems and end with a multi-agent route planning problem, showing that each problem is changed significantly when there is a probability of loss included in the definition. For the simplest cases, such as the family of knapsack problems, risk provides a 'self-regulating' constraint that may require not filling the entire knapsack to maximize expected utility. This self regulation is a recurring theme throughout each problem type, and is a primary force that shapes solutions when risk is added to the optimization problems discussed in this work.

When applying risk to multi-agent routing, risk adds interactions between the allocation of tasks, the physical routes between tasks, the number of assets to use, and the value of the collection assets. This topology of inter-related problems yields a complex problem space that exponentially increases the difficulty of searching for high-valued solutions. Because of this complexity, which can make traditional optimization methods intractable at scale, we introduce a clustering methodology that can find high expected value solutions.

We then consider how a servicing organization could provide vehicles to complete tasks for customers, with mechanisms for fair allocation of costs. We use coalitional game theory to develop an algorithmic method for assigning costs and prove that the resulting allocations are fair and stable. We then use this algorithm to improve the approximate solutions generated by our clustering algorithm, providing cost allocation for large problems.

Finally, we explore an alternate model of risk that has an uncertain cost of completing a task at a given location. In this scenario, a vehicle risks expending limited resources in

search of a low cost, making it impossible to complete the task. We provide a bi-objective optimization approach that can generate sets of non-dominated solution for selection by a user, allowing them to choose between cost and acceptable risk prior to execution.

1.6.1 Problem Scope

The variables used to characterize instances of problems are numerous, including: severity of risk, communication, size of the continuous space, distance measure between tasks, and number of tasks. Each problem instance is a combination of one setting for each of these parameters, yielding a very large experimental space. Therefore, we use only a subset of instance parameters, focusing on regions of the instance space with what we consider to be the most interesting properties.

We model risk as a percent chance of losing the asset depending on how far it travels, which we often set to around 1%. If the risk value is too high, and the task values are not comparably high, then the risk of travel often dominates any benefit of completing tasks. It is worth noting that the ratio of the task value to the risk has strong effects on the solution. Similarly, the ratio of the task value to the asset value can also yield problem instances that prohibit sending any assets, or encourage sending as many as possible. Further exploring this space is of interest in future work.

A majority of our results are on problem instances with 100 tasks placed according to a uniform random distribution, but we also generate instances up to 2000 tasks to show that our clustering approach will scale. The number of tasks and the value of these tasks has a relationship with the size of the space containing the tasks. As the space increases in size, the tasks are farther apart and risk increases. If the space is fixed and more tasks are added the average inter-task distance will be reduced.

Finally, we assume that no communication is available, and that tasks must be physically returned to the base station. Allowing for communication introduces a wealth of new challenges, which we discuss in more detail in Chapter 6.

1.6.2 Contributions

Characterizing risk-aware planning for sensor data collection has proven to be a challenging and fulfilling venture, synthesizing a number of fields of study under a common goal. Because of this problems unique challenges, we have set out to lay a foundation for continued work in risk-aware planning for collection of digital data. This dissertation provides the following contributions:

- We introduce the Collections Planning Problem with Attrition Risk (CPPAR), which seeks to maximize the expected value for completing collection tasks when there is risk of losing assets as they travel through the environment. This multi-agent combinatorial optimization problem bridges the gap between Artificial Intelligence and Operations Research communities, adding probabilistic attrition events that require new solution methods not found in either community.

- Risk-based versions of well-studied optimization problems are formulated and analyzed, providing insight into the foundational challenges in risk-aware planning. We discuss existing theorems and solution methods, and how they can be adapted to provide solutions to CPPAR.

- We provide a clustering algorithm that generates solutions for the CPPAR, and share our results of using graph analysis methods on this problem. We develop an approximation method for computing solutions on a variable set of instances and on very large problem instances. This synthesis of multi-agent route planning and clustering is provided in the context of probabilistic risk, distinguishing it from previous work using clustering methods for multi-agent route planning problems.

- We formulate CPPAR as a hedonic coalitional game, providing a mechanism for a servicing organization to charge customers that request tasks. We develop an algorithm from this formulation, generating a cost allocation profile and also improving

our approximate solutions. This formulation combines the field of coalitional game theory (CGT) with route planning under probabilistic risk, extending previous work in applying CGT to routing problems.

- We present an alternative form of probabilistic risk in the form of uncertainty of collection costs, providing a bi-objective solution methodology that finds the pareto front between cost and task completion probability. This work is part of a relatively new branch in the field of route planning with task cost uncertainty, of which we are one of a few researchers performing work.

- We provide insight into the numerous potential extensions of this work, each representing realistic challenges for dispatching collectives of autonomous vehicles to perform collection tasks in real world environments.

The remainder of this thesis is organized as follows: Chapter 2 provides the formal definition of the Collection Planning Problem with Attrition Risk (CPPAR) and the Volatile Knapsack Problem, which is used to explore fundamental effects of probabilistic risk on combinatorial optimization problems. In Chapter 3, we discuss the use of clustering methods for CPPAR, and provide a clustering-based method for solving CPPAR instances. We also discuss the potential and limitations of using redundancy to improve performance. Chapter 5 provides an alternative form of risk that is characterized by constrained resources and probabilistic costs of data collection. Finally, in Chapter 6 we summarize this thesis and discuss the vast potential for additional research in this domain.

2
Collection Planning Problem with Attrition Risk

In this chapter, we introduce a collection planning problem with attrition risk that includes a risk of losing agents during execution as part of the planning process. We start with risk-based version of a number of classic optimization problems, iteratively increasing the complexity. 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.

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 discuss the effects of risk on a number of well-known combinatorial optimization problems. Finally, we provide insight into solution methods for combinatorial optimization and the inspiration for the clustering approach discussed in Chapter 3.

2.1 Task Allocation Problem

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 determined by the tasks assigned to a single agent. We adopt an objective function similar to that found in [24],

$$\max \sum_{i=1}^{n} \left(\sum_{j=1}^{m} c_{ij}(\mathbf{x}_i, \mathbf{p}_i) x_{ij} \right), \tag{2.1}$$

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 $\mathbf{x}_i \in \{0, 1\}^m$ is the set of tasks assigned to agent i where the jth element is x_{ij}, which can be used to compute any dependencies between task values. The vector $\mathbf{p}_i \in (T \cup \{\emptyset\})^{|T|}$ is an ordered sequence of tasks assigned to agent i, and can be used to determine the distance traveled and time at which each task is visited. The kth element is $j \in T$ if agent i executes task j at the kth point within the path. For our application, these vectors are not necessary, but can be important to relate our specific problem to more general definitions.

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 $G =< V, E >$.

Without any risk of losing the collection assets, and if the number of agents is specified, this problem is equivalent to the Vehicle Routing Problem (VRP) [59] with zero cost for purchasing items. In the next section, we discuss a variant in which there is attrition risk in the environment and the number of agents is unspecified. This problem will be the focus of the majority of this dissertation.

2.2 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

$$p_s(e_{ij}) = \psi^{d_{ij}} \tag{2.2}$$

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 supports continuous distances that have non-integers in the exponent.

Consider a path for agent a_i, $\pi_i = (b, e_{b1}, v_1, e_{12}, ..., v_j, e_{jk}, v_k, e_{kb}, b)$. Traversing a path can be considered a series of events, with a probability of successfully reaching the end as a cumulative probability over traversing a collection of unit length segments in sequence. Let

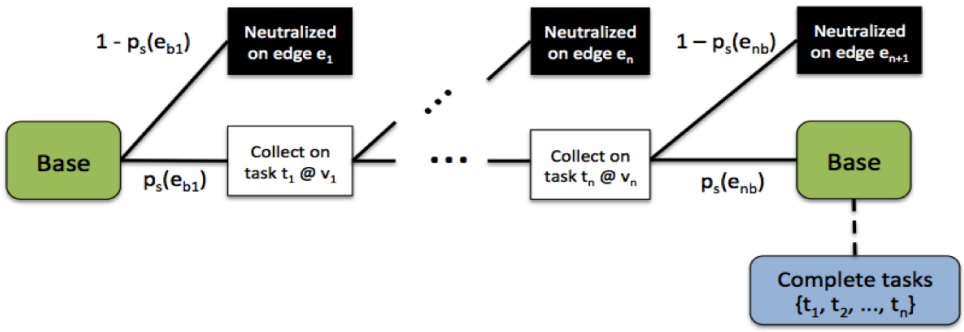


Fig. 2.1: A probability tree representation of the risk model for CPPAR. As the agent traverses each edge between tasks, there is a probability of being neutralized, ending the trip. Upon returning to the base, all collected tasks are completed. Being neutralized before reaching the base results in all collected task data to be lost, resulting in completion of none of the tasks.

ψ be the probability of successfully traversing a unit distance. For a path π_i with distance $d(\pi_i) = \sum_{e_{ij} \in \pi_i} d_{ij}$, the probability of successfully traversing the entire path is

$$S_i = \psi^{d(\pi_i)} \tag{2.3}$$

such that $\lim_{d(\pi_i) \to \infty} S_i = 0$. We use S_i to represent the *survivability* of a path, and use this value extensively in Section 3.4 and Chapter 4. Intuitively, an agent traversing a longer path has a lower likelihood of surviving, and must balance the gain of completing tasks with the possibility of being lost. The *task expected value* of the single agent path is defined as

$$E_t(\pi_i) = \prod_{e_{jk} \in \pi_i} p_s(e_{jk}) \sum_{v_p \in \pi_i} c_{ip}. \tag{2.4}$$

where c_{ip} is the score for agent a_i completing the task t_p at location v_p. Because the agent must return to the base with the task data, the order of collection is not important. The agent either collects and returns all of the task data, or none of it. Therefore, 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

$$E_t(\Pi) = \sum_{\pi_i} E_t(\pi_i). \tag{2.5}$$

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.

2.2.1 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

$$E(\pi_i) = \prod_{e_{jk} \in \pi_i} p_s(e_{jk}) \sum_{v_p \in \pi_i} c_{ip} - \theta_i \cdot (1 - \prod_{e_{jk} \in \pi_i} p_s(e_{jk})). \tag{2.6}$$

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_i 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 2.5, we define the overall expected value for the set of all agent paths as

$$E(\Pi) = \sum_{\pi_i \in \Pi} E(\pi_i). \tag{2.7}$$

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.

2.3 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

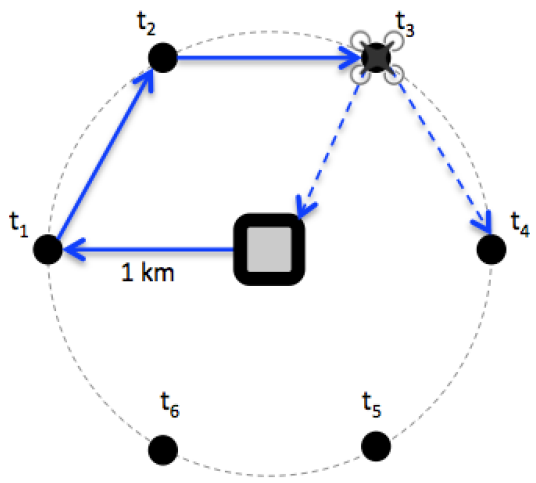


Fig. 2.2: Example single agent problem with 6 tasks 1km apart on a 1km unit circle. A valid plan will visit a sequence of adjacent tasks before returning to the center base.

salesperson problem on a subset of problem vertices.

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 2.2. Because the distances between tasks are equal, the only decision is to determine how many tasks to visit before returning to the base.

In Figure 2.3, we show the resulting expected value for an agent plan with an increasing number of tasks from the example in Figure 2.2. 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.

In Figure 2.4 we 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 2.2. 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

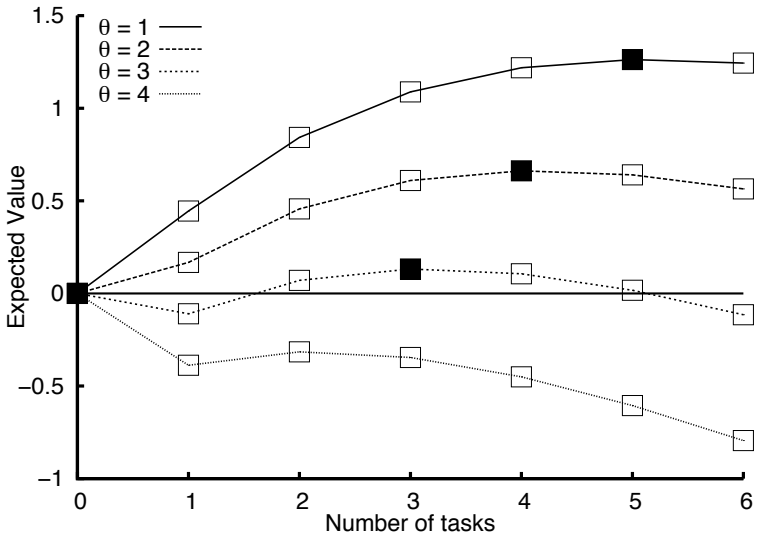


Fig. 2.3: For the example in Figure 2.2, when $\psi = 0.85$ the number of tasks in the highest value assignment (filled boxes) depends on the asset value, θ. The line plotted between the discrete points is included for clarity.

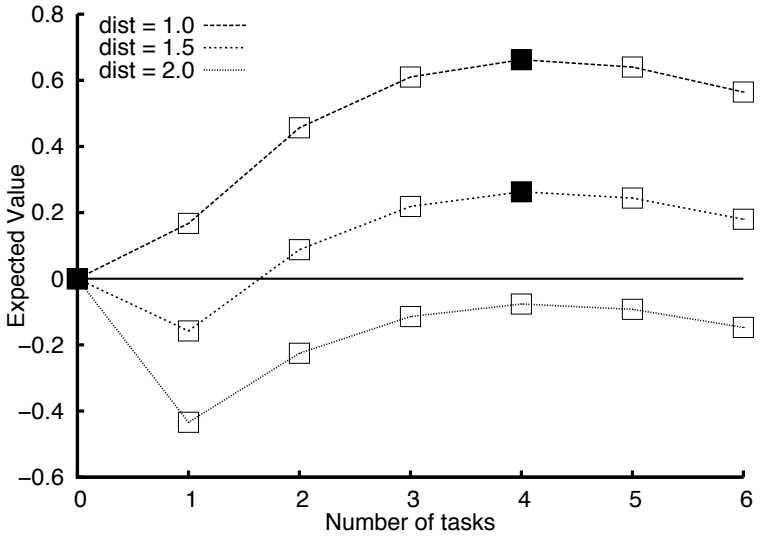


Fig. 2.4: As the distance of the tasks from the base is increased (when $\theta = 2$), the value of completing the tasks is reduced by increased risk to the asset.

base.

2.4 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 2.5, 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.

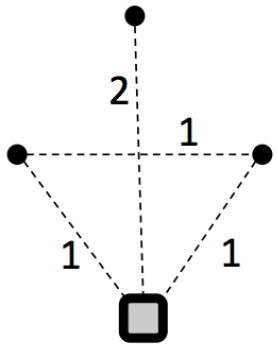


Fig. 2.5: Example set of 3 tasks, denoting the distance between task locations. The probability of survival per unit traveled is $\psi = 0.8$. The objective to to find an assignment of up to 3 agents that maximizes the expected utility.

In Figure 2.6, 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

2.1 we provide the resulting expected value for each assignment for different values of θ, with the best choice shown in bold.

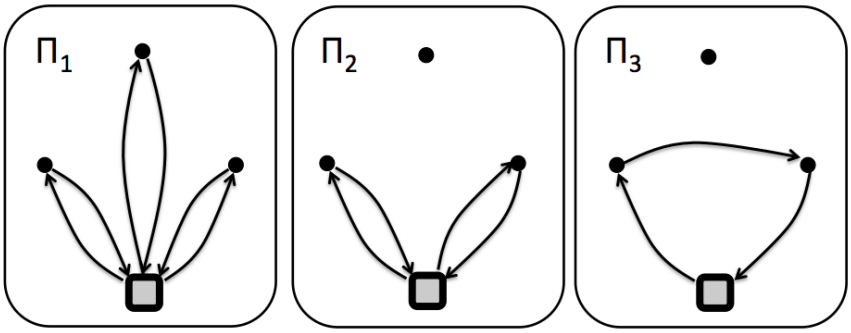


Fig. 2.6: Three possible path allocations of agents to tasks. Π_1 send one agent to each task, Π_2 send one agent to each of the two closer tasks, and Π_3 send one agent to both of the closer tasks.

	Task Assignment			
θ (Asset Value)	Π_1	Π_2	Π_3	$\emptyset$
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 2.1: Task assignment with the maximum expected value for varying asset value, θ. The optimal path allocation changes as the asset value θ is increased.

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 completed. This is analogous to the minimum latency problem when $|A| = |T|$ [70], where the maximum utility is gained by sending each agent to one task and then returning, minimizing the latency of each customer. 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.

These simple, theoretical examples provide insight into the problem space, but we are also interested in finding solutions to CPPAR problems in the general case. To this end, we explore existing methods for solving common combinatorial optimization problems, and evaluate if they can be used on problems that include the risk of attrition.

2.5 Combinatorial Optimization Problems with Attrition Risk

In order to make connections with existing work, we present 'volatile' versions of classical knapsack problems. We introduce risk as a probability of an item destructing, destroying the contents of a container. This shows how risk adds additional complexity, providing a foundation for the challenges of solving CPPAR instances. While some of these problems do not have clear practical applications to collection of sensor data, they provide a simple foundation on which to discuss the interesting properties of attrition risk and how it affects the utility of the optimization problem. We utilize much of the notation found in [31] for the risk-free version of each problem.

2.5.1 Single Item Knapsack

This problem variant is also referred to as the unbounded knapsack problem [5], but with only one type of item available. The item has a weight w and a positive value $v > 0$, and the objective is to maximize the value of the items placed into the knapsack with maximum weight capacity of W. Let x be the number of instances of the item that are placed in the

knapsack.

maximize vx

subject to $wx \leq W$ **and** $x \geq 0$

The solution to this problem can be found trivially by simply determining how many instances of the item can fit within the capacity W, but the introduction of risk yields a more interesting problem.

2.5.2 Single Item Volatile Knapsack

We now introduce the risk of attrition, referring to the modified problem as the *volatile knapsack problem*. Consider an item that is not stable, and may destruct with some probability, destroying all instances of the item stored in the knapsack. We are provided one opportunity to fill the knapsack with some number of the item, after which the probability of destruction is realized. In this problem formulation there is an item with weight w, positive value $v > 0$, and a probability of not destructing p, such that $0 \leq p \leq 1$. In this work, we will often use this 'negative destruction probability', as it aids in simplifying the presentation and computation. The objective is to maximize the *expected value* of the items in the knapsack with maximum capacity of W. Let x be the number of instances of the item that are placed in the knapsack.

maximize $p^x vx$

subject to $wx \leq W$ **and** $x \geq 0$

Unlike the traditional knapsack problem, where we seek to find a best use of the capacity to maximize the value, this objective function has a self-regulating property that limits the desired number of objects. In Figure 2.7 we show the non-discrete expected values for

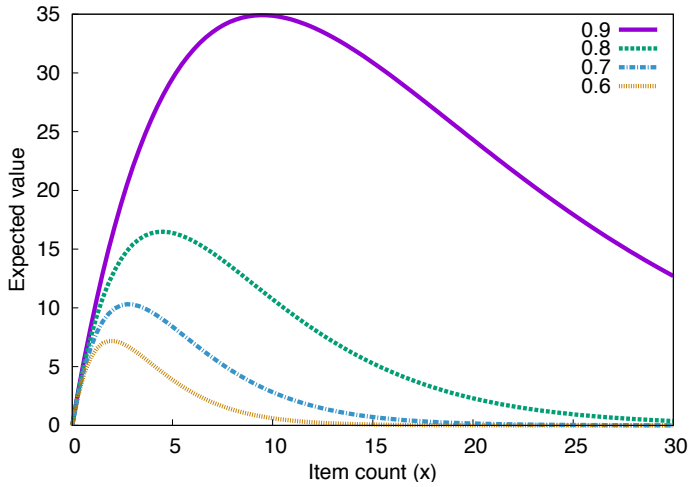


Fig. 2.7: The non-discrete expected value functions for the single item volatile knapsack problem, with $v = 10$ and varying values for the probability of not destructing, p.

adding items with value $v = 10$ to the knapsack. Because each item added incurs additional risk of losing everything in the knapsack, when $p < 1$ there is a maximum number of items at which we do not wish to risk losing everything in the knapsack in order to gain more value. In fact, the capacity of the knapsack W is only a constraint when it is less than the number of items that maximize the expected utility.

If we consider a fixed probability p and vary the value of the item, we find that the optimal number of items is fixed, as shown in Figure 2.8. Because we are considering multiple instances of a single item type with a fixed value, the optimal number of items to place in the knapsack is determined only by the probability of not destructing.

This self-regulating property of the attrition risk model yields interesting effects even in this very simple knapsack problem. In this problem, the value of the item has no bearing on the number of items needed to maximize expected value, as seen in Figure 2.8. To show this formally, consider the value function,

$$V(x) = p^x vx. \tag{2.8}$$

We then take the derivative, yielding

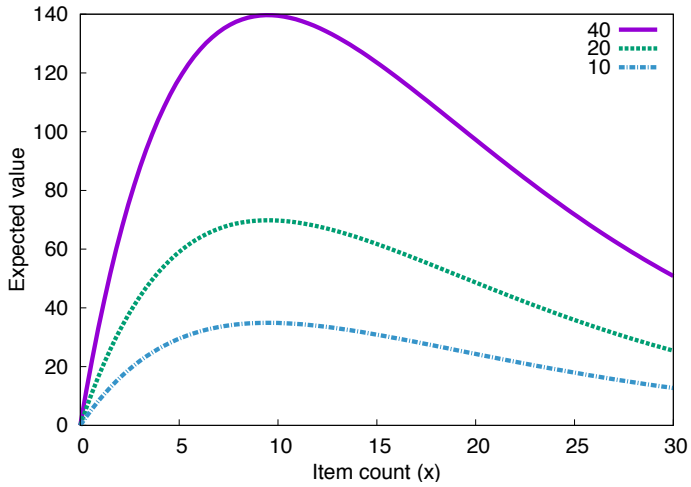


Fig. 2.8: The non-discrete expected value functions for the single item volatile knapsack problem, with $p = 0.9$ and varying values for the item value, v.

$$V'(x) = c(p^x + ln(p)p^x x). \tag{2.9}$$

If we let $V'(x) = 0$, we can solve for the value of x that maximizes $V(x)$,

$$x = -\frac{1}{ln(p)} \tag{2.10}$$

Because this function is in a continuous space, the number of items that maximizes value is the closest integer, $\lceil \frac{1}{ln(p)} \rceil$ or $\lfloor \frac{1}{ln(p)} \rfloor$. This is equivalent to our path planning problem with a single agent, all inter-task edges the same distance, and all tasks with the same value.

This is has similarities to a dice game referred to as Pig [69]. In this game, each player repeatedly rolls a die on their turn, accumulating the value on the die. However, if the player rolls a 1, they lose all points accumulated so far. Therefore, after each roll the player must decide if they will roll again or pass and take the points that have been accumulated on this turn. The challenge is to maximize score while mitigating the risk, and the optimal tactics are a complex function of players scores.

Geometrically, this is a very limited case with no clear practical application, but it

provides some insight into how the value is regulated by the probability. This is opposed to placing constraints on capacity or distance, as is standard practice in route planning problems. We continue by discussing more complex problems and the effects of attrition.

2.5.3 0-1 Knapsack

A commonly used version of the knapsack problem is the 0-1 knapsack problem [65], which provides multiple items to choose from and restricts the number of instances of each item to 0 or 1. This formulation is more intuitive, and in the context of storing data on a platform has clear applications. Consider a single asset that must choose which tasks to collect, given a limit on the storage space available on the platform. We seek to maximize the value of the completed tasks while respecting the space constraint.

We are given a set of n items, numbered from 1 to n, where each item i has a weight w_i (or size of task data) and a value v_i, and the knapsack has a capacity of W. Let $x_i \in \{0, 1\}$ be an indicator variable if item i is included in the knapsack.

$$\textbf{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\textbf{subject to} \sum_{i=1}^{n} w_i x_i \leq W \textbf{ and } x_i \in \{0, 1\}$$

This problem has been shown to be in the class of NP-complete problems, but it can be solved in pseudo-polynomial time using dynamic programming [5].

2.5.4 0-1 Volatile Knapsack

Once again, we consider the risk of attrition. Each item i has a weight w_i, a value v_i, and a probability of not destructing p_i when collected. As with the single item knapsack, we assume that distance between tasks is fixed, and the order in which they are collected has no effect on the probability of destruction. If any item destructs, all of the contents

of the knapsack are lost. Let $X = \{0,1\}^n$ be the vector of indicator variables for all possible items. Let $S(X)$ be the support of X, containing only the elements included in the knapsack.

$$\textbf{maximize} \prod_{x_i \in S(X)} p_i x_i \sum_{i=1}^{n} v_i x_i$$
$$\textbf{subject to} \sum_{i=1}^{n} w_i x_i \leq W \textbf{ and } x_i \in \{0,1\}$$

This can be thought of as a simple single agent version of the CPPAR, where each task has its own probability of destroying the platform, rather than deriving the probability from the distance traveled. In many cases, the maximum capacity W becomes an unnecessary constraint, as the risk of collecting the task data can how much should be collected before the storage limit is reached. However, as p values approach 1 (little or no risk), the capacity constraint has a stronger effect.

For this problem, we would see similar value function to those in Figures 2.7 and 2.8, but we now have varying destruct probabilities and value for each item. The interaction between these provides a rich space of optimization that is familiar among this class of problems. A difficulty that arises is the need to perform multiplication in order to compute the joint probabilities, which increases the computational difficulty over problems with linear relationships between variables.

2.6 Solution Methods for the 0-1 Knapsack Problem

Problems such as the knapsack problem are often solved efficiently using *dynamic programming* [28], which breaks a problem down into sub-problems, solves them individually, and stores their solution for later use. This method is applicable when the problem has optimal substructure and overlapping sub-problems. Knapsack problems are well-known to

have these properties, raising the question if our volatile problems have the same properties to allow us to make use of these methods. If not, then dynamic programming cannot be used effectively, motivating other solutions methods.

Similarly, a *turnpike theorem* [50] is a common method for reducing the problem space in knapsack problems, and are often applicable problems that can be solved using dynamic programming. It chooses the item with the highest density (value divided by weight) and determines if it should be added to the solution. The term turnpike is derived from the idea that if we are planning a long automobile trip, then we should expect to spend most of our time on the turnpike, or highway, as it is the fastest we can travel. Therefore, if we can identify the turnpike and include it in the solution, then we only need to solve for the sub-problem of getting on and off the turnpike. Similarly, if we can identify an item that will always be included in the solution, we can add it and solve the remaining sub-problem.

Using these approaches on the volatile version of these problems requires optimal sub-structure. We will show that this problem formulation does not have optimal substructure, preventing us from making use of these well-studied methods. The following example highlights the need for more sophisticated search methods in order to solve this class of optimization problems.

Let $X = \{0, 1\}^n$ be the vector of indicator variables for all items in a set of size n. Each item i has a value v_i, and a probability of not destructing p_i when collected. We can denote the probability of the items in X not destructing as p_X and the value of the items as v_X, with $V(X) = p_X v_X$ as the expected value of X. Assume for this example that the weight capacity for the knapsack exceeds the sum of all item weights. Adding a new item a with probability p_a and value v_a to X_a yields a new expected value,

$$V(X_a) = (p_X \cdot p_a)(v_X + v_a) \tag{2.11}$$

Let $X^* = \{0, 1\}^n$ be an optimal selection of items from a set of size n, such that $p_{X^*} = .9$

and $v_{X^*} = 30$. We are now given two additional items to consider, a and b, such that,

$$p_a = .25, v_a = 40 \qquad V(a) = 10$$
$$p_b = .9, v_b = 10 \qquad V(b) = 9$$

If we must choose one item from the pair, item a is of higher value. However, if we consider adding each item to the existing optimal solution, we get the following values,

$$V(X_a) = (.9 \cdot .25)(30 + 40) = 15.75$$
$$V(X_b) = (.9 \cdot .9)(30 + 10) = 32.4$$

This example shows that we are unable to consider the value of items in relative isolation, but rather each item must be considered along with items that may not be represented in the sub-problem. This prevents us from using existing dynamic programming algorithms to solve the volatile versions of these problems.

2.6.1 Turnpike theorems for CPPAR

We have shown that the volatile knapsack problem (and by extension, CPPAR) does not exhibit optimal substructure, preventing us from using a turnpike theorem. The primary benefit of finding a turnpike in a knapsack problem is that it can be used to reduce the problem by subtracting the weight of the item from the capacity and solving for the remaining items. For the volatile knpasack, we can identify a single item that is of the highest value, and likely to be included in the optimal solution, but we are unable to use that selection to reduce the problem size. This is because collections of items must be considered as a bundle, and their value is a combination of their joint probability and the sum of their values.

Therefore, a similar turnpike method for the volatile knapsack would consider groups

of items, starting with the most valuable item first and storing the current probability and value for the next iteration. We then choose items that, in combination with previously chosen items, provide the largest gain in expected value. Eventually, none of the remaining items will provide a positive gain in expected value, indicating that no further items should be added. Unlike dynamic programming methods such as the original turnpike method, these steps are sequential and can not be performed in isolation.

While this process is effective for the volatile knapsack, it does not apply directly to CPPAR for two reasons. First, because tasks are located in a metric space, the order in which they are chosen affects the distance traveled and, therefore, the probability of survival. Second, when there are multiple agents to choose from it may be better to split a group of tasks among multiple agents rather than bundle them all together for one agent to collect. Both of these aspects of the problem lead to exponential growth in the state space, making a tree-based search computationally difficult.

These challenges led us to find a method for grouping tasks into multiple bundles, each of which maximized a local bundle value while also allowing for multiple agents. To help guide our search for a solution, we used brute force methods to solve a number of relatively small instances. Some observed properties of optimal solutions for the CPPAR 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.

The preference for tasks that are closer together is similar across most path-based planning problems, and clustering has long been a common method for helping to inform path

construction [73]. However, the introduction of risk motivates new solution concepts that consider all of these aspects, doing more than find shortest paths. Clustering methods seemed like a natural fit, but existing algorithms were not well suited to dealing with the risk model. Based on these challenges, we have devised a clustering algorithm that is responsive to the unique aspects of this problem and is capable of finding solutions for large numbers of tasks with variable agents. In the next chapter we will provide a description of our algorithm.

3
CLUSTERING FOR CPPAR

The collection planning problem with attrition risk (CPPAR), while having some unique properties, shares a number of features with other multi-agent planning problem. In general, tasks that are located spatially near each other are more likely to be connected by a single agent path. Additionally, when ample agents are available, a single path will often be contained on one side of the base. For example, with 2 agents with a base in the center, it's often the case that each agent will cover the tasks on one side of the base, with relatively no overlap.

The observation of 'clusters' among path-based planning has been widespread, and a number of algorithms use a form of clustering as a means of reducing the search space for finding solutions. The *cluster first-route second* method uses clustering to group together nearby tasks, then executes the routing procedure on those clusters [60]. In contrast, the *route first-cluster second* method generates a single route ignoring constraints, and then uses clustering to to split the route into feasible paths [9].

We present a solution method, referred to as Progressive Risk-aware Clustering (PRC), that uses hierarchical clustering to generate plans for groups of agents to complete a set of collection tasks with the risk of attrition. Tasks are clustered using a bottom-up approach, which starts with each task being in a singleton cluster and iteratively joining clusters in a

pairwise fashion, ending with all tasks in one cluster. This sequence yields a hierarchical data structure that can be processed to provide task groupings, with each group being visited by one agent. By exploiting spatial relationships and considering risk, PRC generates solutions with a high expected value. Additionally, the algorithm is effective across a wide range of problem instances without requiring user calibration.

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 [22] [96] [58] [19]. This is not the first work to use hierarchical clustering to find task groupings [81], but we do not require a user specify the number of agents to be used.

The value of a cluster in the CPPAR is dependent on a few key properties, as depicted in Figure 3.1. First, we must consider the total value of the tasks being collected as part of a cluster. Second, the inter-task distance is the length of the path that is necessary to visit all of the tasks in the cluster. Dense clusters would be expected to have a short inter-path distance, while loosely packed tasks will require additional distance to visit all of them. Finally, the distance of the cluster from the base has a significant effect on the overall value. In order to reach the cluster, the collecting agent must travel to and from the cluster, incurring additional risk.

The risk of losing assets and their associated tasks scheduled for collection introduces additional constraints that must be considered. If the value of the asset is high, then there may be tasks that are too far away, or not of sufficient value, to be collected. In this case, there are tasks that are not a member of any cluster. Additionally, as more tasks are added to a path, both the distance traveled and the value at risk is increased. Therefore, there is a 'critical mass' of tasks that is reached, at which point no more tasks can be added to a cluster without reducing the overall value of the path.

With these considerations in mind, we set out to investigate a number of methods for grouping tasks into clusters. Each method is evaluated based on the performance against a

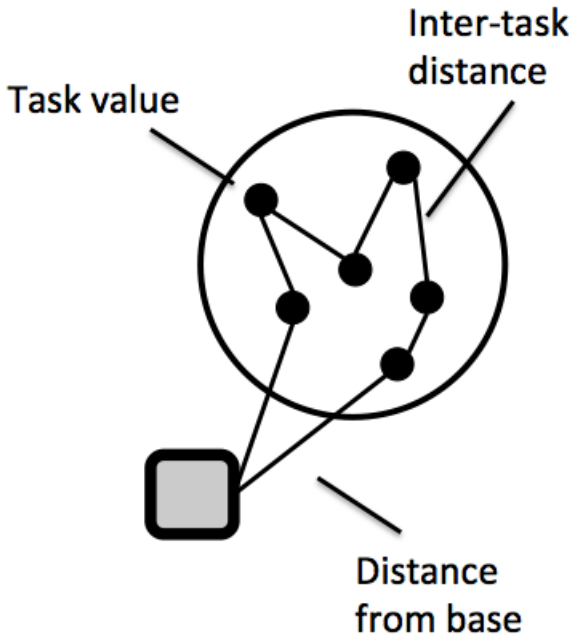


Fig. 3.1: The key properties of a cluster for CPPAR. The task value determines the joint value of the cluster, the inter-task distance is the length of the path visiting all tasks in the cluster, and the distance from the base determines how much value a cluster must have to be worth visiting.

common baseline, using a range of problem and parameter settings. We investigate the use of both community detection and clustering methods, which differ in their approach. Community detection looks for structure in a graph, requiring us to reduce the complete graph of inter-task edges. We test two methods for graph reduction, k-nearest neighbor and median cut, along with a community detection algorithm that seeks to maximize Modularity (Q-score) [17]. For clustering methods we show results for DBSCAN [33] and Heirarchical Clustering [27], neither of which require the number of clusters to be specified a priori. Based on these results, we formulate a method well-suited to CPPAR, which we call Progressive Risk-Aware Clustering (PRC).

3.1 Empirical Evaluation of Clustering Methods

Given the clear application of clustering methods to path-based planning problems, and the similarities of CPPAR to planning problems without risk, we investigated the effectiveness of existing clustering approaches. Applying these clustering methods to CPPAR turned out to be challenging for a few reasons:

- The number of paths to be used is not known a priori. When risk is low, many agents should be dispatched, when risk is high, few or no agents should be sent. Therefore, we should exclude any methods that require the number of clusters be specified, or find a automated way to select the number of clusters.

- Cluster value can decrease if clusters grow too large, which requires recomputing the distance measure between tasks based on the current size of the cluster. Many clustering algorithms assume the ‘distance’ between instances remains fixed throughout the clustering process.

- The value of a task (and its associated cluster) is in part determined by how far the agent must travel from the base to reach it. This yields cluster values that are a

function of both inter-task Euclidean distance and the relative distance from the base.

Uncertain of exactly which forms of clustering would be best suited to generate solutions for CPPAR, we implemented a number of approaches and compared their performance. We describe the most notable performing algorithms and discuss the implications of their results.

To evaluate the performance of these algorithms, we compare against the sequential greedy (SG) algorithm [20], which iteratively adds tasks to paths based on the gain in expected value. At each step, all tasks are evaluated based on insertion to all possible locations in existing paths, as well as the gain for starting a new path with only that task. The operation with the largest gain is performed and the task is removed from future consideration. When no tasks have a positive gain in expected value, the algorithm terminates and the resulting paths are used as the solution.

The following experiments were conducted on 100 randomly generated CPPAR problem instances with 100 tasks each. Tasks are placed in a 100km ÃŮ 100km 2D Euclidean space, according to a uniform random distribution. The probability of survival is ÏĹ$\psi = 0.99$ for each km traveled. Each approach is evaluated based on the mean expected value (EV) of the solution with respect to the mean performance of the SG baseline on the same instances.

For each set of experiments, we observe performance over a range of asset values (θ) in order to determine if the algorithm is effective for a range of problems. Recall from Chapter 2, Figure 2.6, that the optimal solutions vary significantly as asset value changes, with $\theta = 0$ assigning one agent to each task and high values of θ motivating paths that prefer closer tasks using less assets.

k-NN Cut + Modularity

In this approach, the initial complete graph is modified by keeping edges only between each vertex and the k nearest vertices in the space, where k is an input parameter to the

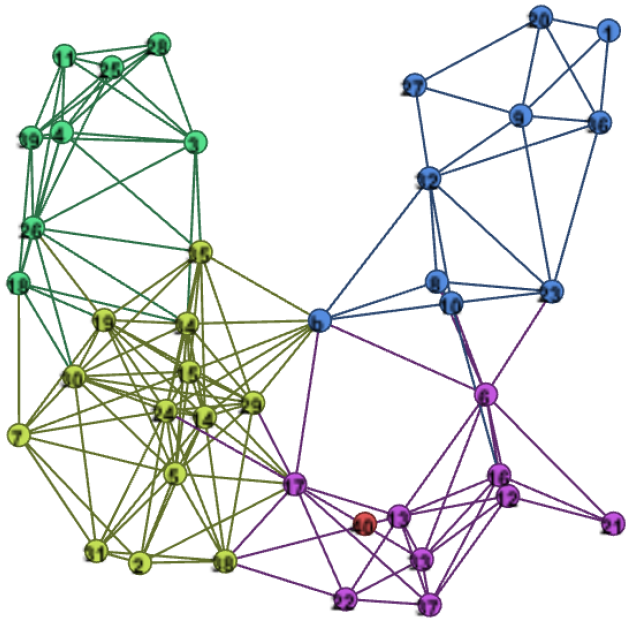


Fig. 3.2: Example of the application of the Modularity metric to a reduced graph. Different colors represents individual community, each of which is used to form a path for a single agent.

process. Because the community detection process is based on the existence of edges and does not consider distance, this reduction takes the complete graph and preserves links only between nearby tasks. We then use community detection method called Modularity [17], provided by the Gephi graph analysis library [8], to generate a set of groupings. Each group is converted to a path using the same sequential greedy method [24] we used for the baseline, except only on the subset of tasks identified to be in a single cluster.

The use of Modularity for path-based planning has been shown to be successful for Traveling Salesman problems [3]. However, it is highly sensitive to the topology of the reduced graph, motivating the exploration of different methods for generating the reduced graph.

Experiments were performed using an increasing number of nearest neighbors during the graph reduction, $k = \{1, 2, 3, 4, 5\}$. Increasing values of k preserve more edges connected to a single node, and result in graphs with more edges overall. The results of this approach on simulated instances is shown in Figure 3.3. Surprisingly, when $k = 1$, the

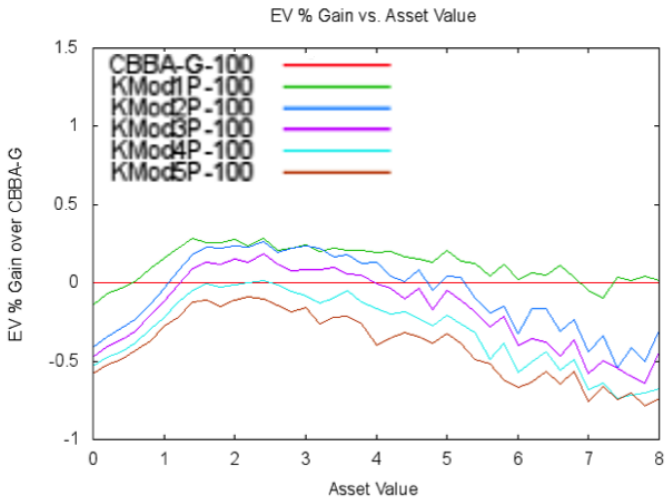


Fig. 3.3: Performance of k-NN graph reduction with Modularity for detecting groupings of tasks for verious values of k versus baseline (red line).

performance was the highest, with performance degrading as k in increased.

We were encouraged by the performance of this approach when $k = 1$, as it outperformed the baseline sequential greedy algorithm under certain conditions. This encouraged us to try other clustering methods to see if further performance gains could be found. However, it was still unclear if the k-NN process was causing poor performance, as opposed to the Modularity process.

Median Cut + Modularity

Instead of having an exact number of edges per vertex, as was the case with the k-NN graph reduction, we also tried reducing the graph by ordering the edges based on distance and removing those edges that fell beneath some percentage value. Originally referred to as a 'median cut' method, which eliminated any edges with a value less than the median, we also tried a number of percentile thresholds (0.1, 0.3, 0.5, 0.7, 0.9).

In general, this approach showed poor performance for all parameter settings tested, as shown in Figure 3.4. Based on these result, it seemed that more sophisticated graph reduction techniques would be needed to achieve the level of performance we would expect.

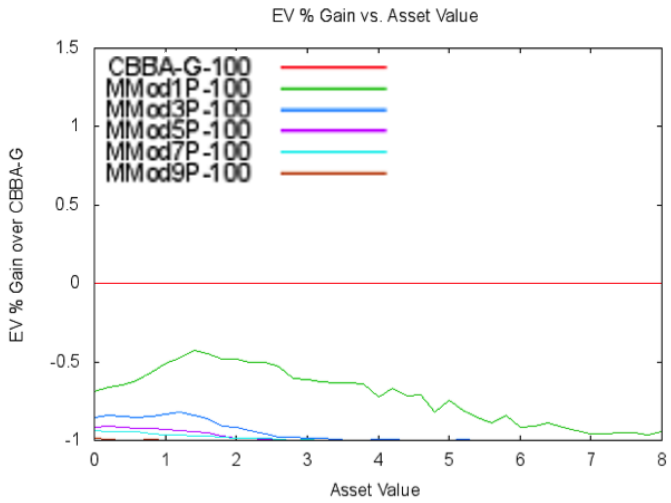


Fig. 3.4: Performance of 'median cut' reduction with Modularity for detecting groupings of tasks for various percentile cutoff values versus baseline (red line).

However, performing reduction based on pairwise computation of risk would be exponentially expensive. Therefore, we changed direction and looked at more traditional clustering methods.

DBSCAN

We first tried using clustering methods that were well-suited to 'visual' clustering of instances. These approaches have the benefit of not needing to have any specification of how many clusters that are expected. The DBSCAN [33] algorithm, contained in the WEKA library [45], requires the user to specify two parameters: *minPoints* specifies the minimum number of points nearby each other to be considered a cluster, and *epsilon* specifies how close together vertices should be to be considered part of a the same cluster. To coarsely explore the parameter space, we performed experiments using $minPoints = \{5, 10, 15\}$ and $epsilon = \{0.3, 0.5, 0.7\}$.

As shown in Figure 3.5, this approach performed better Median Cut + Modularity in many cases, but fell short of the promising performance seen with k-NN + Modularity. In addition, it also introduces the need to modify two parameters depending on the properties

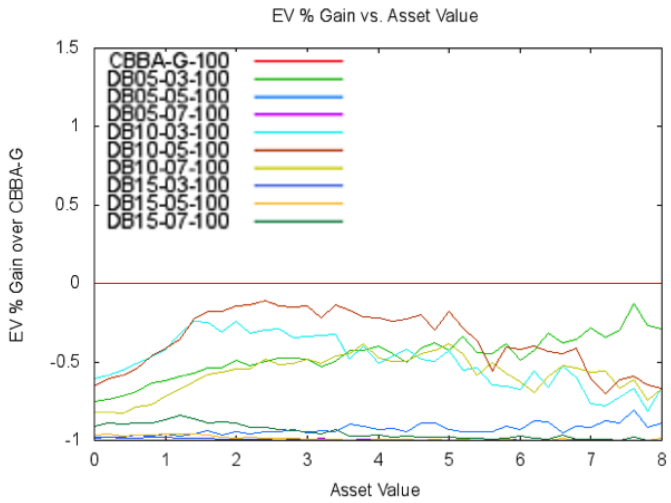


Fig. 3.5: Performance of DBSCAN for detecting groupings of tasks for various *minPoints* and *epsilon* values versus baseline (red line).

of varying problem instances. This seemed to introduce unecessary complexity if we were try to adapt it for CPPAR.

Hierarchical Clustering

Another clustering method that allows for clustering without explicitly specifying the number of clusters is Hierarchical Clustering [27], a method for grouping instances in a metric space. We use a 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, referred to as agglomerative clustering. Given the output of the process, a cutoff is chosen that induces a set of clusters. We used the WEKA library [45] for this implementation, which required a specification of the number of clusters to extract from the output, for which we select from $\{5, 10, 15, 20, 25, 30\}$.

In Figure 3.6 we show that hierarchical agglomerative clustering (HAC) proposed task assignments with the highest observed values across the clustering methods, in some cases showing 50% improvement across most asset values tested. This relatively stable performance led us to choose hierarchical clustering as our baseline approach and explore new

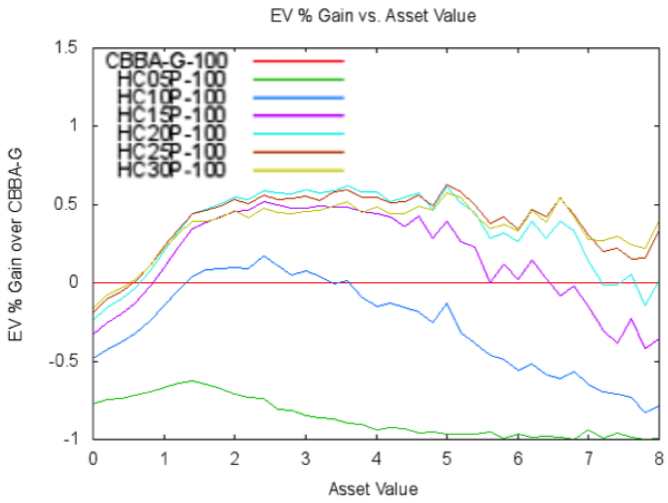


Fig. 3.6: Performance of Hierarchical Clustering for detecting groupings of tasks for various number of output clusters, versus baseline (red line).

ways to adapt it more specifically to this unique problem. The resulting algorithm is discussed in the next section.

3.2 Hierarchical Agglomerative Clustering for CPPAR

Based on the superior performance of HAC in our experiments, we conducted further investigation into the algorithmic process to identify improvements. The result of the HAC process can be visualized as a dendrogram, as shown in Figure 3.7. 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 [64] 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

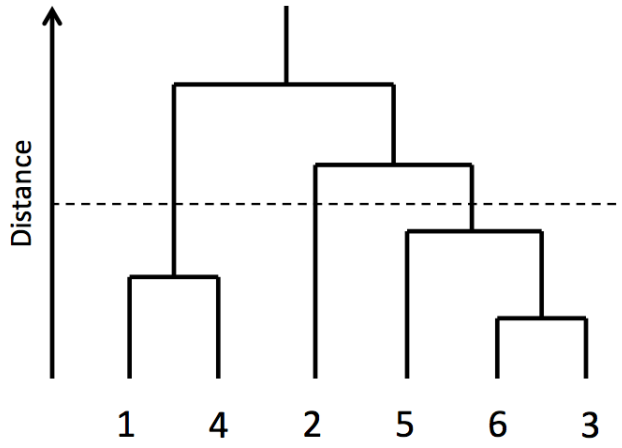


Fig. 3.7: An example dendrogram for six instances. Horizontal lines indicate a merge of two clusters, with the height along the the vertical axis being the distance between the clusters. The dotted line is an example of using a distance cutoff value to generate a set of clusters. In this case, the resulting clusters are $\{1, 4\}$, $\{2\}$, and $\{5, 6, 3\}$.

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 must consider not just the distance between clusters, but the gain in expected value when performing the merge. This requires considering all of the cluster features shown in Figure 3.1. Task value is fairly straightforward, as it is an additive property. The inter-task distance and base distance, however, require computing a new path for the combined set of tasks as part of the evaluation. This can be computationally expensive, motivating approximation methods that we will describe in detail.

Our Progressive Risk-aware Clustering (PRC) method 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", as the search process considers risk, exceeding the capabilities of existing algorithm to provide solutions for the CPPAR. We provide a formal

description of this algorithm in the next section.

3.2.1 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 methods with orders of magnitude less computation required. Unlike clustering methods commonly used for multi-agent tasking, we do not specify how many clusters are to be formed as part of the problem definition. Instead, we consider determining how many agents to send as part of the problem begin solved, allowing the details of the problem instance 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.

Recall from Chapter 2 that c_{ip} is the score for agent a_i completing the task t_p, and $p_s(e_{jk})$ is the probability of successfully traversing the edge between tasks t_j and t_k. Given a cluster $\mathcal{C}_i$ consisting of a set of tasks, 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:

$$EV^*(\mathcal{C}_i) = \prod_{e_{jk} \in \pi_i^*} p_s(e_{jk}) \sum_{t_p \in \mathcal{C}_i} c_{ip} - \theta_i \cdot (1 - \prod_{e_{jk} \in \pi_i^*} p_s(e_{jk})). \tag{3.1}$$

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

$$GAIN^*(\mathcal{C}_i, \mathcal{C}_j) = EV^*(\{\mathcal{C}_i \cup \mathcal{C}_j\}) - EV^*(\mathcal{C}_i) - EV^*(\mathcal{C}_j). \tag{3.2}$$

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-

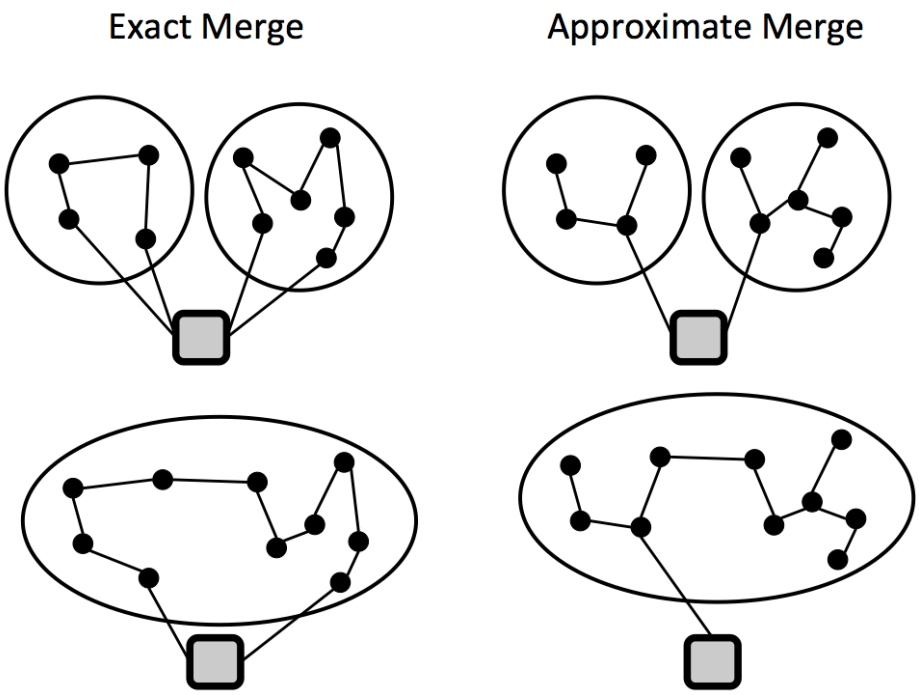


Fig. 3.8: An example showing the difference between an exact merge between clusters, which requires computing the optimal TSP route for the merged clusters, and an approximate merge, which utilizes the Minimum Spanning Tree and a single shortest distance path to compute the benefit of merging two clusters.

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. An example of this approximation method is shown in Figure 3.8.

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 MST has been shown to be a *2-approx* solution for a TSP[74], meaning that the value it returns is no more than 2 times the actual TSP distance. Using these components as an estimate of distance, we define the approximate path length for cluster $\mathcal{C}_i$ 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:

$$\tilde{V}(\mathcal{C}_i) = |\mathcal{C}_i| \cdot \psi^{\tilde{d}(\mathcal{C}_i)} - \theta_i \cdot (1 - \psi^{\tilde{d}(\mathcal{C}_i)}). \tag{3.3}$$

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

$$GAIN(\mathcal{C}_i, \mathcal{C}_j) = \tilde{V}(\{\mathcal{C}_i \cup \mathcal{C}_i\}) - \tilde{V}(\mathcal{C}_i) - \tilde{V}(\mathcal{C}_j). \tag{3.4}$$

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 0.7 [1, 85], 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. ELKI performs neighbor search using a database core that is indexed specifically for performing distance queries, yielding impressive speed for our clustering application.

3.3 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 once again compare against the sequential greedy (SG) algorithm specified in [24], 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,

Algorithm 1 Progressive Risk-aware Clustering

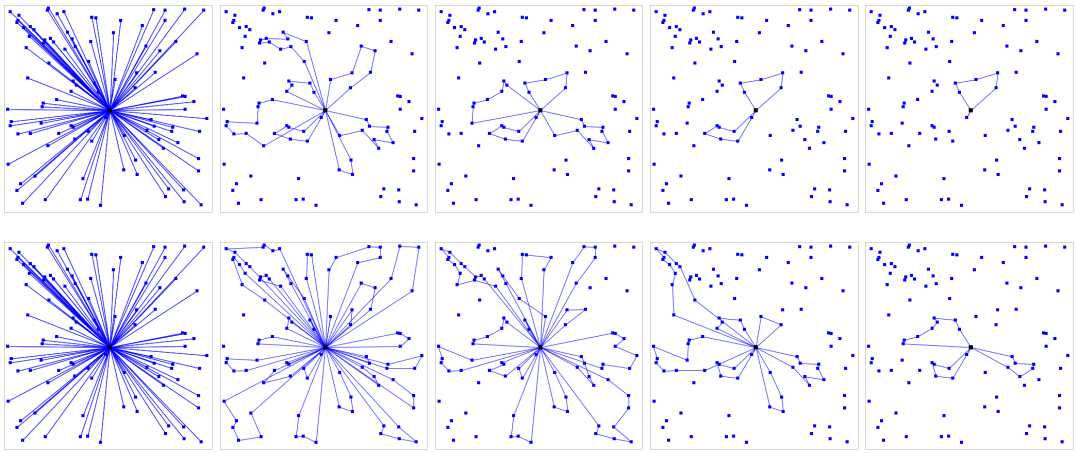
Input: Task set $T = \{t_1...t_n\}$, graph $G =< V, E >$, where $V = T \cup \{b\}$
Output: Set of clusters $\{\mathcal{C}_1, ..., \mathcal{C}_m\}$.

$\mathcal{A} \leftarrow \emptyset$
for $i \leftarrow 1...n$ **do**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\{t_i\}\}$
    $P[i] \leftarrow i$
    $H[i] \leftarrow \infty$
    $S[i] \leftarrow 1$
    $B[i] \leftarrow dist(b, t_i)$
    $M[i] \leftarrow 0$
    for $j = (i + 1)...n$ **do**
        dist[i][j] $\leftarrow dist(t_i, t_j)$
    end for
end for

while $|\mathcal{A}| > 1$ **do**
    $\mathcal{C}_1^*, \mathcal{C}_2^* \leftarrow \underset{\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{A}}{\arg\max}\ GAIN(\mathcal{C}_1, \mathcal{C}_2)$
    $P[\mathcal{C}_1^*] \leftarrow \mathcal{C}_2^*$
    $H[\mathcal{C}_1^*] \leftarrow GAIN(\mathcal{C}_1^*, \mathcal{C}_2^*)$
    $S[\mathcal{C}_2^*] \leftarrow S[\mathcal{C}_1^*] + S[\mathcal{C}_2^*]$
    $B[\mathcal{C}_2^*] \leftarrow min(B[\mathcal{C}_1^*], B[\mathcal{C}_2^*])$
    $M[\mathcal{C}_2^*] \leftarrow M[\mathcal{C}_1^*] + M[\mathcal{C}_2^*] + dist(\mathcal{C}_1^*, \mathcal{C}_2^*)$
    for all $C_i \in \mathcal{A} \setminus \mathcal{C}_2^*$ **do**
        $t_i^*, t_j^* \leftarrow \underset{x_i \in \mathcal{C}_i, x_j \in \mathcal{C}_2^*}{\arg\min}$
        $dist[\mathcal{C}_i][\mathcal{C}_2^*] \leftarrow dist(t_i^*, t_j^*)$
    end for
end while

ExtractClusters()

for all $\mathcal{C}_i \in \mathcal{A}$ **do**
    if $EV^*(\mathcal{C}_i) \leq 0$ **then**
        $\mathcal{A} \leftarrow \mathcal{A} \setminus \mathcal{C}_i$
    end if
end for



(f) $\theta = 0$ (g) $\theta = 1$ (h) $\theta = 2$ (i) $\theta = 3$ (j) $\theta = 4$

Fig. 3.9: Comparison of solutions provided by Sequential Greedy Gain [top] and PRC [bottom] 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.

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 [24] 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 3.11 and 3.12. 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.

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 3.9. 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

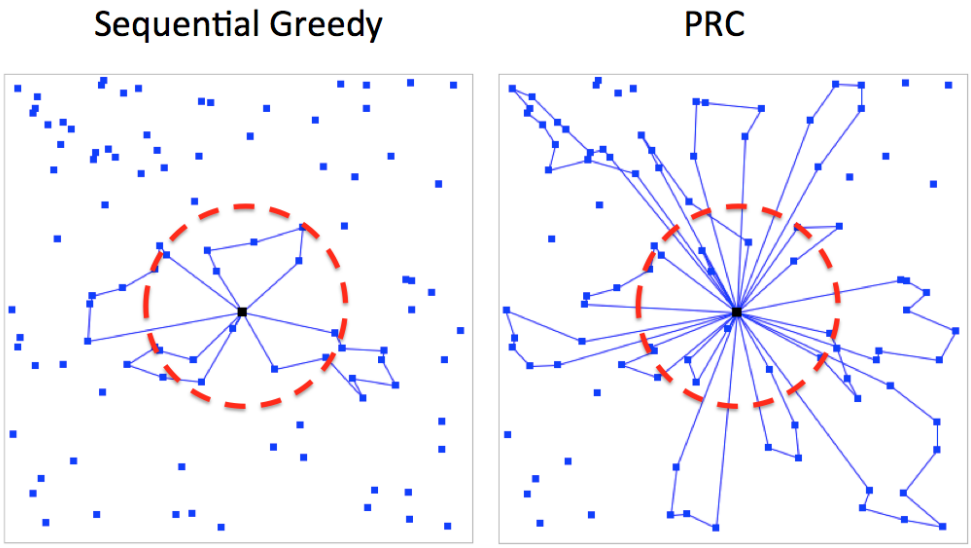


Fig. 3.10: A single pair of solutions from Figure 3.9, for $\theta = 2$, is shown. The red dashed circle is the threshold at which a single task would not have sufficient value to form a cluster with positive expected value. As a result, the Sequential Greedy method will form only paths that have a single task within that range, making it unable to recognize high-value clusters outside that range.

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). An example of this behavior is shown in Figure 3.10.

In Figure 3.11 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 3.12. We use this measure to show performance over a set problem instances with a variable number of tasks. The increased variance for high asset values is due to there being relatively fewer tasks being visited as part of a solution, in some cases no tasks are visited at all. Therefore, the performance of all algorithms is more

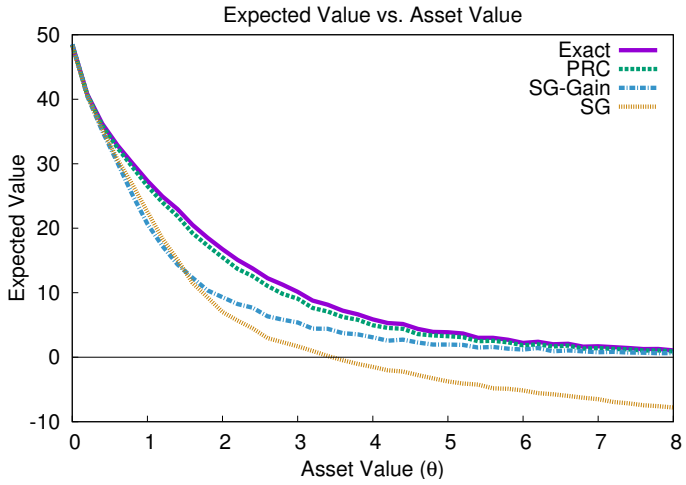


Fig. 3.11: The expected value of multi-agent plans generated by each algorithm on 100 tasks. PRC achieves performance near Exact Merge, with improvement over SG and SG-Gain.

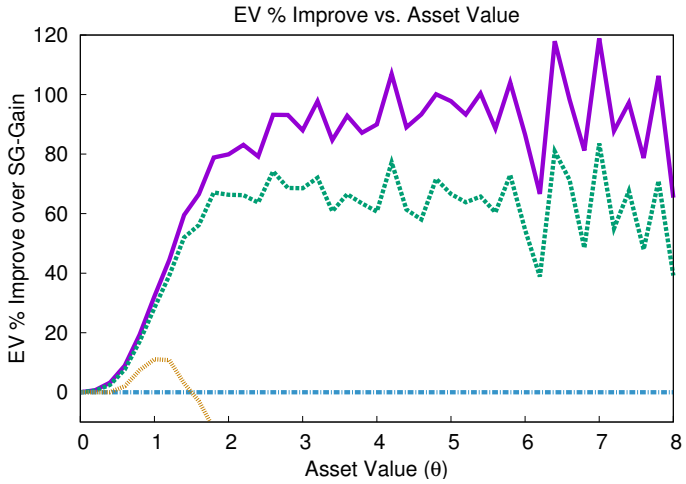


Fig. 3.12: The % improvement over SG-Gain for increasing asset value with 100 tasks. SG quickly degrades in performance as the asset value increases, Exact and PRC show improvement over SG-Gain except when $\theta = 0$.

dependent on the randomly generated problems, and how many tasks are located near the base. With respect to CPU time, PRC provides these performance results at a significant computational savings in almost all cases, as shown in Figure 3.13.

Algorithm 2 Exact Merge Algorithm

1: $\mathcal{A} \leftarrow \emptyset$
2: maxGain $\leftarrow -\infty$
3: **for** $i \leftarrow 1...N$ **do**
4: $\quad \mathcal{A} \leftarrow \mathcal{A} \cup \{\{x_i\}\}$
5: **end for**
6: **while** maxGain > 0 AND $|\mathcal{A}| > 1$ **do**
7: $\quad \mathcal{C}_1^*, \mathcal{C}_2^* \leftarrow \underset{\mathcal{C}_1, \mathcal{C}_2 \in \mathcal{A}}{\arg\max}\ GAIN(\mathcal{C}_1, \mathcal{C}_2)$
8: $\quad$ maxGain $\leftarrow GAIN(\mathcal{C}_1^*, \mathcal{C}_2^*)$
9: $\quad \mathcal{A} \leftarrow \mathcal{A} \setminus \{\{\mathcal{C}_1^*\}, \{\mathcal{C}_2^*\}\}$
10: $\quad \mathcal{A} \leftarrow \mathcal{A} \cup \{\mathcal{C}_1^* \cup \mathcal{C}_2^*\}$
11: **end while**

In Figure 3.14 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 3.15, the number of task sites visited by each planner is shown. The Sequential Greedy algorithm will always collect on all of the tasks, 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 3.11.

Finally, we show how many agents were dispatched by each algorithm in Figure 3.16. 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, moti-

vating sending an agent to collect and gain the associated value.

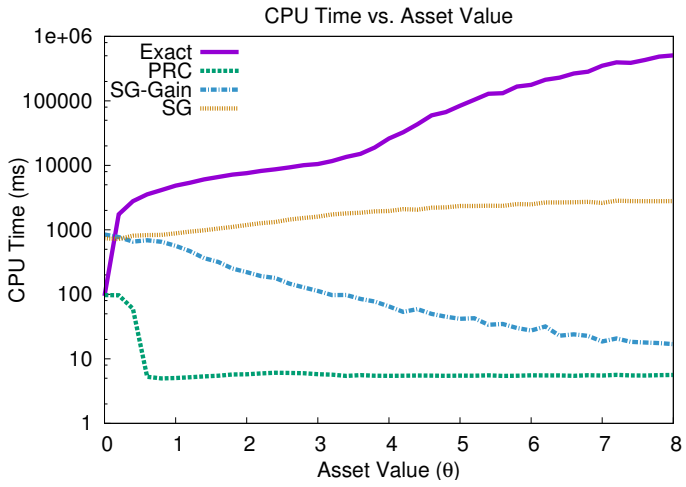


Fig. 3.13: CPU Time in milliseconds for each algorithm with respect to asset value for 100 tasks. Note the logarithm scale used for the vertical axis.

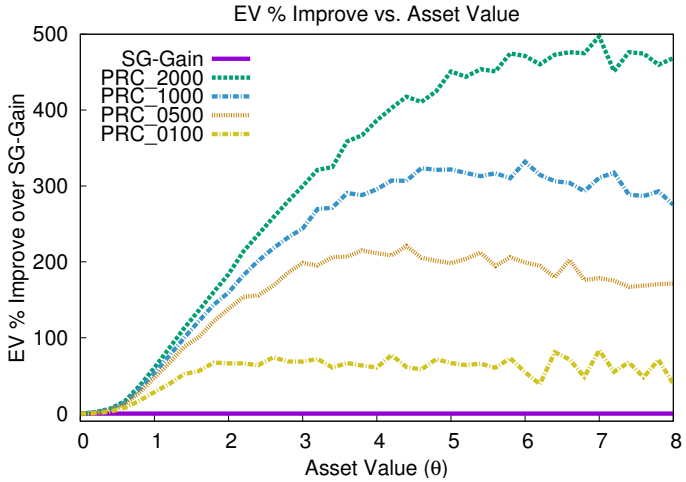


Fig. 3.14: The % improvement over SG-Gain grows as the problem size increases. Results are shown for PRC on problems with the 100, 500, 1000, and 2000 tasks.

The solutions generated by the PRC algorithm have been shown to be effective when a single agent visits each task, but there is additional value that can be gained by allowing for redundant collection of sensor data, increasing the probability of success. We will explore redundancy, and the limits on value that can be gained, in the next section.

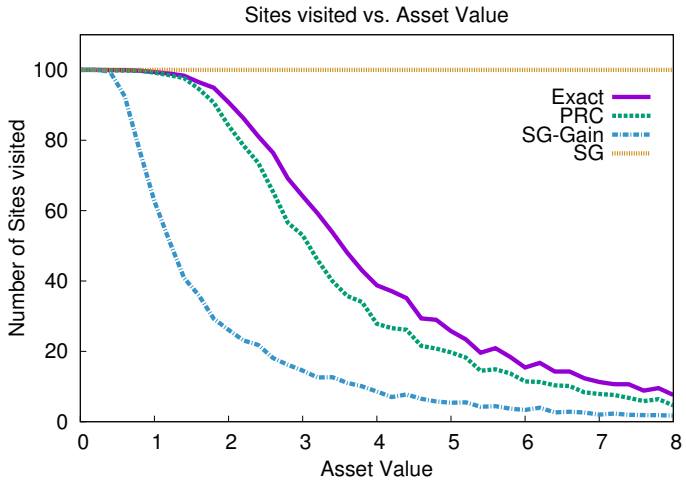


Fig. 3.15: The number of sites visited by all agents with respect to asset value θ, for 100 tasks. Sequential Greedy (SG) is designed to visit all of the sites under all conditions, the SG-Gain variant over corrects, visiting too few sites.

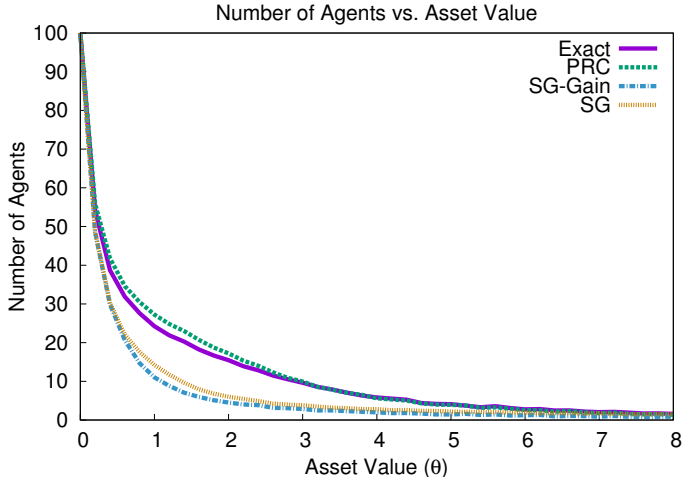


Fig. 3.16: Number of agents used by each algorithm with respect to asset value θ, for 100 tasks. The Exact Merge algorithms provides better performance using less agents than PRC, but at the cost of CPU Time, as shown in Figure 3.13.

3.4 Redundant Collection for CPPAR

Before this point, there has been an implicit assumption that tasks were collected once and returned to the base. However, it's not hard to imagine scenarios in which the task data can be collected by one or more agents, with credit being given for completion as long as at least one of them returns to the base with the data. For example, consider gathering imagery fixed locations, which can be collected at any time. If more than one agent is sent to collect the image then we can mitigate the risk of losing one of the agents, increasing the probability of completing that task.

Adding redundancy comes with a cost, however. Instead of one agent extending its path to complete a task, multiple agents must increase their risk to collect a single task. This requires that the value of the task, and the marginal gain in probability of successfully completing it, offset the additional cost of increasing the risk of losing another asset to collect it. We show that redundancy of collection has value only when the ratio of the asset cost to the task value is very low, on the order of 1:5. Additionally, each additional redundant agent has diminishing returns, as it can only marginally increase the probability of successfully completing the task.

In this section, we explore a method for including redundant collection in the solution process. We present an algorithm for computing an initial solution using the PRC algorithm and then iteratively search for redundant collections that maximize the expected value. Our results show a diminishing return on the increase in expected value from redundantly collecting a task. In cases where the asset value is high, there is no benefit to including redundancy.

3.4.1 Redundancy-Based Utility

To model redundancy, we leverage concepts from reliability engineering. A common method for handling faults is to form components in a series parallel system, where re-

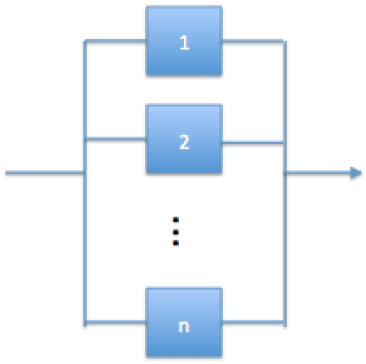


Fig. 3.17: Block diagram for a parallel system, with components 1 through n providing redundancy. Only one of the components needs to be successful for the overall process to be successful.

dundancy can provide protection against single component failures [88]. In the example shown in Figure 3.17, we have n parallel components, with each component i having a reliability value, R_i, indicating the probability of providing the desired function. The overall reliability R of the block diagram is,

$$R = 1 - (1 - R_1)(1 - R_2) \times \cdots \times (1 - R_n) \tag{3.5}$$

Similarly, we can think of agents as components that are acting in parallel to ensure a task gets completed. Each agent has a reliability value, which we refer to as *survivability*. The survivability S_i (see Equation 2.3) is the probability of the event that an agent a_i will return to the base with the collected task data via the specified path π_i.

Consider a single task t_i, that may be visited by one or more agents, where π_j is the path for agent a_j. By considering all of the agents that visit a node, we can compute the expected value for that task. Let Π_{t_i} be the set of agent paths that visit task t_i, with a task score c_i. The *expected task value* is,

$$V(t_i, \Pi_{t_i}) = \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right)\right] c_i \tag{3.6}$$

Adding a new agent path π_N to a task yields the following value

$$V(t_i, \Pi_{t_i} \cup \pi_N) = \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right)(1 - S_N)\right] c_i \tag{3.7}$$

The increase in value from adding an additional agent path has diminishing returns, as each successive path has reduced gain that it can provide to the probability of completing the task. For example, consider a set of paths $\Pi = \{\pi_1, .., \pi_n\}$, each with a fixed probability of success $S_1 = ... = S_n = 0.5$. STarting with a task with no visits, successively adding the task to each of these paths will yield the following sequence of success probabilities for the task, $0, 0.50, 0.75, 0.875, 0.9375, ...$, approaching 1.0 in the limit. Therefore, there will be a threshold at which it is no longer worth adding the task to an additional path, as the added value is less than the overall cost.

This diminishing returns is fairly straightforward if each path has a fixed probability of survival that is independent of what tasks are visited. However, in our model, adding a task to an agent path will affect the path that is traveled and cause the probability of survival to change accordingly. In our model, where the risk of attrition is a function of distance traveled, adding a task to a path will always increase the risk of the vehicle being lost, which will influence the probability of task completion for *all of the tasks* being visit by that path.

Path survivability

We now consider paths through metric space, where the distance between tasks locations is of importance. Let $\pi_i' = \pi_i \oplus v_j$ symbolize inserting task location v_j into path pi_i such that we minimize the the path length of π_i'. Using this definition, we can denote the change in path distance for inserting task t_j into path π_i is

$$\delta(\pi_i, v_j) = d(\pi_i \oplus v_j) - d(\pi_i) \tag{3.8}$$

It is worth noting that an empty path is considered to start and end at the base with no

task locations visited. Therefore, if π_i is an empty path, then $d(\pi_i) = 0$ and $\delta(\pi_i, v_j) = d(\pi_i \oplus v_j)$.

Traversing a path can be considered a series of events, with a probability of successfully reaching the end 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 a path π_i with distance $d(\pi_i)$, the probability of successfully traversing the entire path is

$$S_i = \psi^{d(\pi_i)} \tag{3.9}$$

such that $\lim_{d(\pi_i)\to\infty} S_i = 0$. 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.

Adding a task to a path will increase the distance traveled, simultaneously decreasing the survivability of the path. Let S_i' be the new survivability of a path after adding task location v_j, defined as,

$$S_i' = \psi^{(d(\pi_i)+\delta(\pi_i,v_j))} = S_i \cdot \psi^{\delta(\pi_i,v_j)} \tag{3.10}$$

Clearly, $S_i' < S_i$, which implies that adding any new task to an agent path will reduce the survival probability of that agent. This has implication that will be discussed in the next section.

$$S_i^{\Delta} = S_i - S_i \cdot \psi^{\delta(\pi_i,v_j)} \tag{3.11}$$

For additional clarity, consider the example plan in Figure 3.18, which has two paths π_1 and π_2. In this example, we consider altering path π_1 to include task t_4, providing redundancy. The survivability of the original paths is S_1 and S_2, and we denote the survivability of the updated π_1 as $S_1' = S_1 \cdot \psi^{d_2+d_3-d_1}$. This change will have 3 effects:

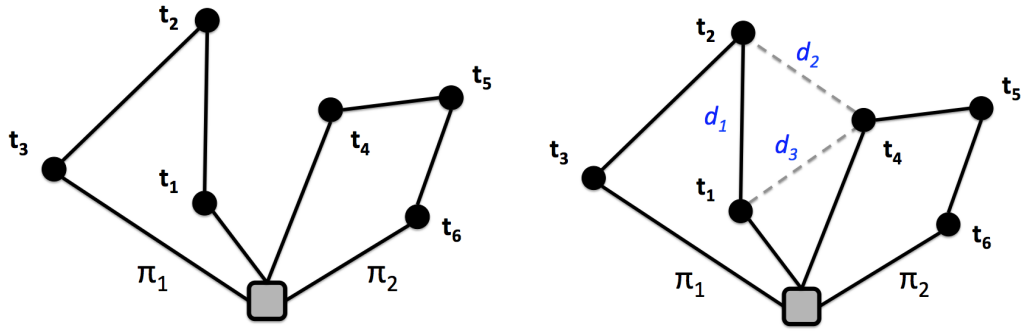


Fig. 3.18: Example plan with two paths (left) and a proposed extension of path π_1 to redundantly collect task t_4 (right).

- The probability of completing task t_4 will increase from S_2 to $1-(1-S_1')(1-S_2)$.
- The probability of completing the other tasks in π_1 will be reduced from S_1 to S_1'.
- The probability of losing asset a_1 will increase from $1-S_1$ to $1-S_1'$.

In order for the added redundancy to provide a gain in expected value, the gain in utility for task t_4 must be greater than the the reduction of value for the other tasks in the extended path, plus the additional cost incurred by the risk to the asset. Because of the diminishing returns on the increase in value for the task, a positive gain in expected value is usually observed when asset value is relatively low with respect to the task value. We provide evidence of this in Section 3.4.4.

3.4.2 CPPAR Utility with Redundancy

Having defined the effects of adding tasks locations to an agent path, we can now formulate the task expected value when an agent path π_N is altered to visit task t_i. This definition considers the full effects of both redundancy and the increased distance (and resulting risk) of the agent path.

To avoid having to compute the value of all tasks every time we check the value of adding a redundant visit, we define the updated task expected value in terms of the previous

expected value.

$$V(t_i, \Pi_{t_i} \cup \pi_N) = \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right)\left(1 - S_N \cdot \psi^{\delta(\pi_N, v_j)}\right)\right] c_i \tag{3.12}$$

$$= \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right) + \left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right) S_N \cdot \psi^{\delta(\pi_N, v_j)}\right] c_i \tag{3.13}$$

$$= V(t_i, \Pi_{t_i}) + \left[\left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right) S_N \cdot \psi^{\delta(\pi_N, v_j)}\right] c_i \tag{3.14}$$

$$= V(t_i, \Pi_{t_i}) + \left[\left(1 - \frac{V(t_i, \Pi_{t_i})}{c_i}\right) S_N \cdot \psi^{\delta(\pi_N, v_j)}\right] c_i \tag{3.15}$$

$$= V(t_i, \Pi_{t_i}) + \left[S_N \cdot \psi^{\delta(\pi_N, v_j)} - \frac{V(t_i, \Pi_{t_i})}{c_i} S_N \cdot \psi^{\delta(\pi_N, v_j)}\right] c_i \tag{3.16}$$

$$= V(t_i, \Pi_{t_i}) - V(t_i, \Pi_{t_i}) S_N \cdot \psi^{\delta(\pi_N, v_j)} + S_N \cdot \psi^{\delta(\pi_N, v_j)} \cdot c_i \tag{3.17}$$

$$= V(t_i, \Pi_{t_i})\left(1 - S_N \cdot \psi^{\delta(\pi_N, v_j)}\right) + c_i \cdot S_N \cdot \psi^{\delta(\pi_N, v_j)} \tag{3.18}$$

Using this result, we can now compute the net *benefit* of adding task t_i to agent path π_j,

$$B(t_i, \pi_j) = V(t_i, \Pi_{t_i} \cup \pi_j) - V(t_i, \Pi_{t_i}) \tag{3.19}$$

$$= V(t_i, \Pi_{t_i})\left(1 - S_j \cdot \psi^{\delta(\pi_j, v_i)}\right) + c_i \cdot S_j \cdot \psi^{\delta(\pi_j, v_i)} - V(t_i, \Pi_{t_i}) \tag{3.20}$$

$$= V(t_i, \Pi_{t_i})\left(1 - 1 - S_j \cdot \psi^{\delta(\pi_N, v_j)}\right) + c_i \cdot S_j \cdot \psi^{\delta(\pi_N, v_i)} \tag{3.21}$$

$$= c_i \cdot S_j \cdot \psi^{\delta(\pi_j, v_i)} - V(t_i, \Pi_{t_i}) S_j \cdot \psi^{\delta(\pi_j, v_i)} \tag{3.22}$$

$$= S_j \cdot \psi^{\delta(\pi_j, v_i)}\left(c_i - V(t_i, \Pi_{t_i})\right)) \tag{3.23}$$

In addition, we must consider the effects of reducing the survivability of the agent path π_N. This change not only effects the new task being visited, but also the set of all tasks currently being serviced by the agent. Let Π'_{t_i} be the set of agent paths that extends path

π_N such that $\{\pi_N \oplus t_j\}$,

$$\Pi'_{t_i} = (\Pi_{t_i} \setminus \{\pi_N\}) \cup \{\pi_N \oplus t_j\} \tag{3.24}$$

Once again, we wish to define the change in expected task value with respect to the previous value. We do so as follows:

$$V(t_i, \Pi'_{t_i}) = \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i} \setminus \pi_N} (1 - S_j)\right)(1 - S_N \cdot \psi^{\delta(\pi_N, v_j)})\right] c_i \tag{3.25}$$

$$= \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i} \setminus \pi_N} (1 - S_j)\right)(1 - (S_N - S_N^{\Delta}))\right] c_i \tag{3.26}$$

$$= \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i} \setminus \pi_N} (1 - S_j)\right)(1 - S_N + S_N^{\Delta})\right] c_i \tag{3.27}$$

$$= \left[1 - \left(\left(\prod_{\pi_j \in \Pi_{t_i} \setminus \pi_N} (1 - S_j)\right)(1 - S_N) + \left(\prod_{\pi_j \in \Pi_{t_i} \setminus \pi_N} (1 - S_j)\right) S_N^{\Delta}\right)\right] c_i \tag{3.28}$$

$$= \left[1 - \left(\prod_{\pi_j \in \Pi_{t_i}} (1 - S_j)\right) - \left(\prod_{\pi_j \in \Pi_{t_i \setminus \pi_N}} (1 - S_j)\right) S_N^{\Delta}\right] c_i \tag{3.29}$$

$$= V(t_i, \Pi_{t_i}) - \left[\left(\prod_{\pi_j \in \Pi_{t_i \setminus \pi_N}} (1 - S_j)\right) S_N^{\Delta}\right] c_i \tag{3.30}$$

Using this formulation, we can define the cost to t_i in agent path π_j when task t_k is added,

$$C(t_k, \pi_p, t_i) = V(t_k, \Pi_{t_k}) - V(t_i, (\Pi_{t_k} \setminus \{\pi_p\}) \cup \{\pi_p \oplus t_i\}) \tag{3.31}$$

$$= V(t_k, \Pi_{t_k}) - V(t_k, \Pi_{t_k}) + \left[\left(\prod_{\pi_j \in \Pi_{t_k} \setminus \pi_p} (1 - S_j) \right) S_p^{\Delta} \right] c_k \tag{3.32}$$

$$= \left[\left(\prod_{\pi_j \in \Pi_{t_k} \setminus \pi_p} (1 - S_j) \right) S_p^{\Delta} \right] c_k \tag{3.33}$$

By combining the benefit and costs of adding a redundant task, we can quantify the expected gain (or loss, if negative) of adding task t_i to agent path π_j.

$$G(t_i, \pi_j) = B(t_i, \pi_j) - \sum_{t_k \in T(\pi_j)} C(t_k, \pi_j, t_i) \tag{3.34}$$

where $T(\pi_i)$ be the set of tasks currently being visited by agent path π_i.

3.4.3 Algorithm

Having defined the equation for computing gain of adding a redundant collection, we now provide an algorithm that uses the gain to add redundant visits. We begin by computing an initial non-redundant solution using the PRC algorithm described in Chapter 3, and then iteratively search for redundant collections that maximize the expected value.

We impose a maximum redundancy value (M_R) to bound the algorithm. To understand the need for this parameter, consider the case when $\theta = 0$, meaning that the assets have no value. In this case, the optimal solution is one that has an infinite number of agents visit each task and return to the base, even though the gain in expected value for additional paths diminishes as redundancy increase.

We avoid adding redundancy to more than one site at a time. Once a single redundancy is added, it can change the value of adding redundancy to tasks within the corresponding

Table 3.1: Redundant Collection Algorithm for the Collection Planning Problem with Attrition Risk

Input: Task set $T = \{t_1, ..., t_n\}$, graph $G =< V, E >$ where $V = T \cup \{b\}$, maximum redundnacy M_R.
Output: Set of paths Π.

Phase 1. Risk-Aware Clustering

1. Execute Progressive Risk-aware Clustering algorithm as described in Chapter 3, generating approximate coalitions.
2. Store resulting task coalitions as $\mathcal{P}$.

Phase 2. Redundancy Insertion

1. **while** there exists a task-path pair t_i, π_j such that $G(t_i, \pi_j) > 0$ and $\Pi(t_i) < M_R$ **do**
   (a) All tasks $t_i \in T$ check for potential redundancies, choosing the path that maximizes its payoff.
   (b) The task-path pair with the highest expected gain in payoff t_m, π_m is chosen, adding t_m to path π_m.
2. Return the updated set of paths Π'.

paths. Therefore, we make sure that the gain in utility is recomputed before allowing for another redundancy to be added. In the next section, we provide the results of executing the algorithm on simulated instances.

3.4.4 Experimental Results

We now show the results of computational experiments using the Progressive Risk-Aware Clustering with Redundancy (PRC+R). 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 task y_i has a score $c_i = 1$, and we show the average over 100 randomly generated instances. We consider the effects of adding redundancy with respect to expected value, the gain in expected value over sequential greedy, and the computation time.

We denote our redundancy algorithm as $PRC + R\{M_R\}$, where M_R is the maximum number of paths that are permitted to visit a single tasks. This bound serves dual purposes. First, it limits the number of redundancies when $\theta = 0$, for which the optimal solution has an infinite number of paths visiting each task. Also, performing the checks to determine the value of adding a redundant visit is computationally expensive, and increasing values of M_R will perform an increasing number of checks. This bound allows us to explore the trade-off between allowing more redundancy and CPU time.

In Figure 3.19 we show the expected value for sequential greedy (SG), progressive risk-aware clustering (PRC), and increasing amounts of maximum redundancy ranging from 2 paths per site to 7 paths per site. These results show the diminishing returns of value for increasing redundancy, and also highlight the lack of gain for using redundancy as the relative value of the asset increases.

We provide a closer look at the difference between the different algorithms in Figure 3.20, showing the gain in expected value over the sequential greedy baseline. Even for very small θ values, the gain for increasing redundancy beyond 5 paths per task is minimal. It's

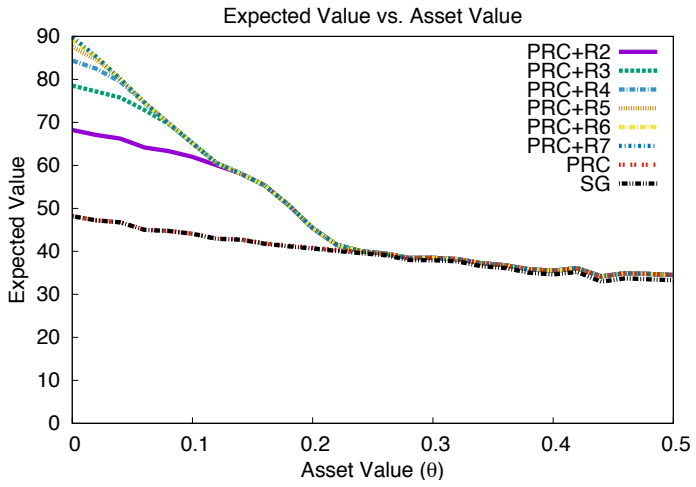


Fig. 3.19: Expected value for sequential greedy (SG), progressive risk-aware clustering (PRC), and increasing maximum number of paths per task (M_R) ranging from 2 paths per task to 7 paths per task. For high valued assets the cost of visiting a task redundantly is increased, therefore the gain in expected value is only observed when the relative asset value is low.

worth noting that all tasks have a score $c_i = 1$, and tasks with a larger value would provide increased value for redundancy.

The execution time for finding redundant collections is significant, as each task computes the value of insertion into every edge of each path. However, the computational cost can provide significant gains in value. Similarly to the trend seen for expected value, as the asset value increases there is little difference between the computation time for the increasing level of maximum redundancy.

These results provide encouraging evidence that redundancy can provide significant gain in solution value when the asset value is low with respect to the task value, on the order of 1:5. For problem instances in which assets have a higher value, no gain is expected and the increased computation should be avoided. Exploring the trade-off between solution quality and computational time with respect to the properties of the problem instance is an interesting direction we discuss further in Chapter 6.

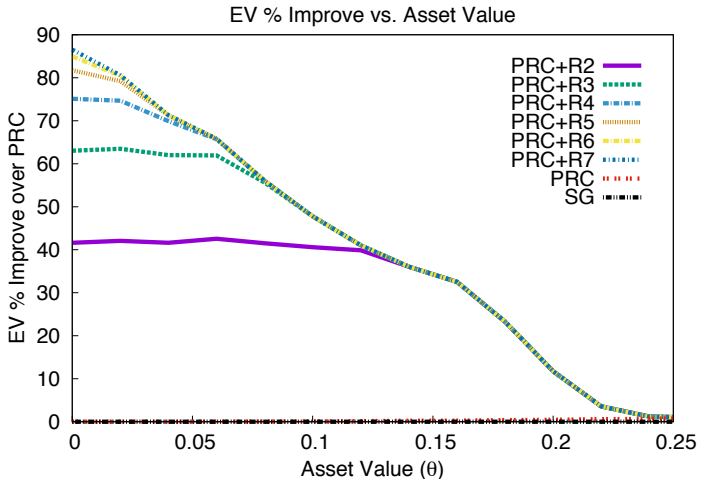


Fig. 3.20: Percent gain over sequential greedy (SG) with different degrees of maximum redundancy. As expected, we observe diminishing returns on adding additional paths to one site.

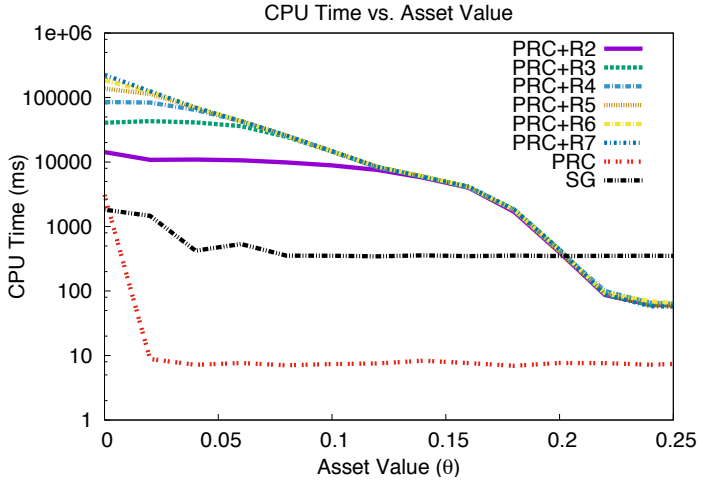


Fig. 3.21: Execution time for sequential greedy (SG), progressive risk-aware clustering (PRC), and increasing amounts of maximum redundancy ranging from 2 paths per site to 7 paths per site. Increasing redundancy requires additional computation.

Realistic Problem Instances

The previous results were drawn from problem instances with a uniform random distribution of tasks in the environment. In the real world, we would often expect that tasks will have some natural groupings with a less uniform distribution. For example, capturing images of events near an urban environment would likely have more tasks in the highly populated areas, and less where there are fewer people. While the uniform generation process inevitably yields some degree of groupings of tasks, we are also interested in how our algorithm performs on non-uniform problem instances with a higher degree of natural clustering.

Real problem instances were taken from the repository of National Traveling Salesman Problems found at http://www.math.uwaterloo.ca/tsp/world/countries.html, which are generated from cities in world countries and collected VLSI instances. The country data sets are relatively large in size, spanning over a large area with irregular density, while the VLSI instances are highly structured, with alternating regions of high and low density. In general, the VLSI instance have an overall higher density of tasks per unit distance. The results of using this data are found in Tables 3.2 and 3.3.

Across all input density distributions tested, both synthetic a real, the PRC algorithm performs better than the Sequential Greedy baseline for all asset values from 0 to 8. As with the synthetic instances, the performance gap increases as θ increases from 0, and narrows as the value of the assets becomes significantly larger than the task values. As with the synthetic instances, this improvement is due to the limitations of the sequential greedy algorithm to generate new paths with no tasks within the radius that a single task is worth visiting.

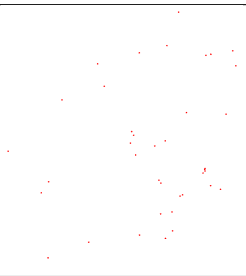
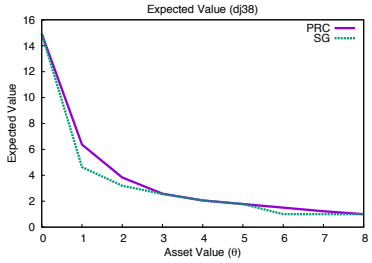
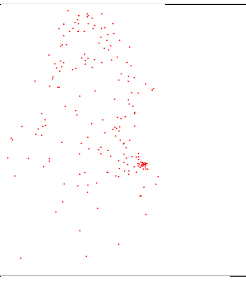
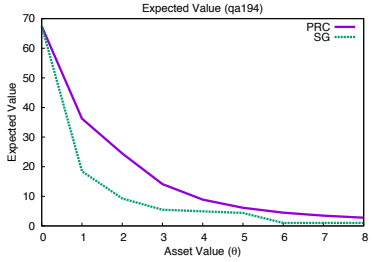
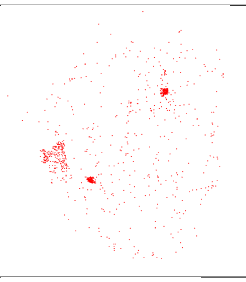
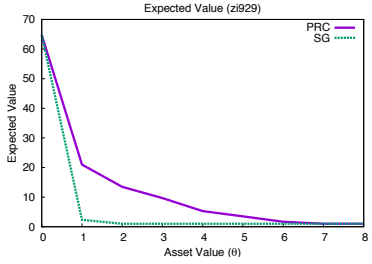
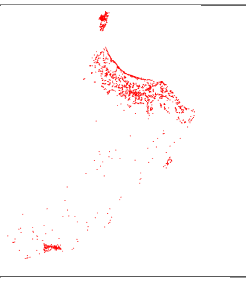
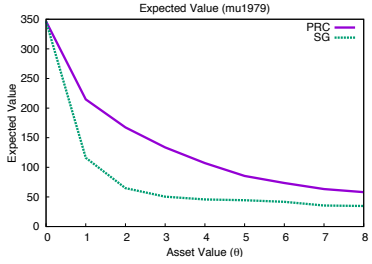
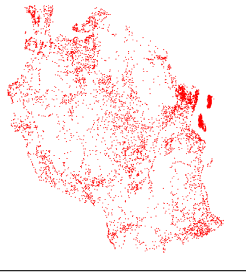
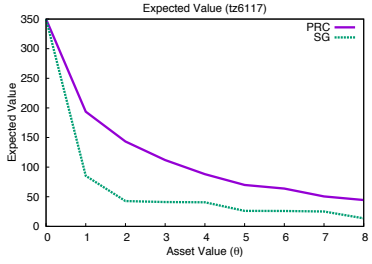
Instance	Point Set	Radius	Density	Results
dj38		1858	1.56E-5	
qa194		1459	1.63E-4	
zi929		7300	1.99E-5	
mu1979		10164	2.81E-5	
tz6117		13700	5.25E-5	

Table 3.2: Problem instances based on collections of cities in countries, displaying a non-uniform population density. Instances by country are Djibouti (dj38), Qatar (qa194), Zimbabwe (zi929), Oman (mu1979), and Tanzania (tz6117). The number in the instance name indicates the number of points. (Source: http://www.math.uwaterloo.ca/tsp/world/countries.html)

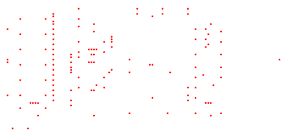
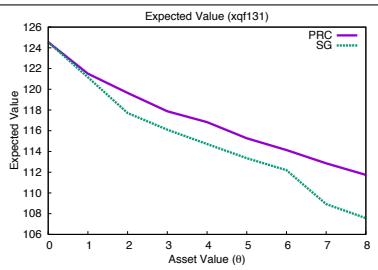
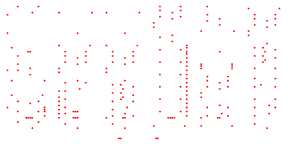
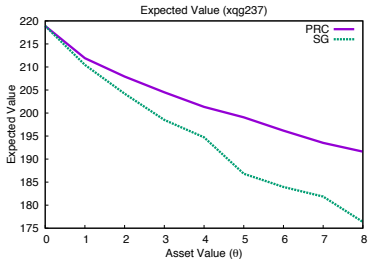
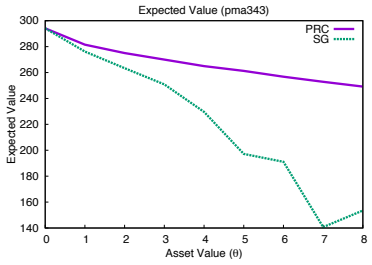
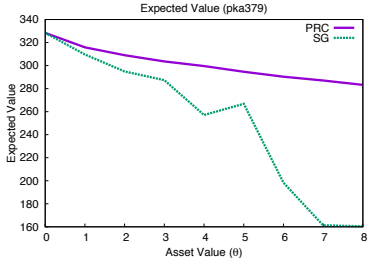
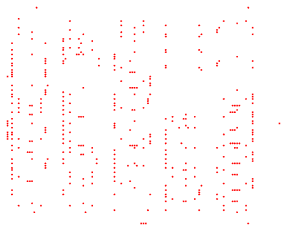
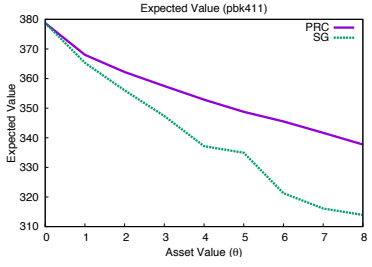
Instance	Point Set	Radius	Density	Results
xqf131		108.4	0.0262	
xqg237		140.5	0.0282	
pma343		300.0	0.0274	
pka379		297.8	0.0328	
pbk411		138.1	0.0348	

Table 3.3: Problem instances derived from VLSI data provided by the Bonn Institute, displaying highly structured clustering of points at relatively regular intervals. The number in the instance name indicates the number of points. (Source: http://www.math.uwaterloo.ca/tsp/vlsi/index.html)

3.5 Guided Random Search Algorithms

We have shown positive results using our Progressive Risk-aware Clustering algorithm to generate good solutions for CPPAR, as compared to an iterative search baseline. However, the Sequential Greedy algorithm has clear shortcomings, motivating comparison against other optimization/search methods. In this section we explore an alternate solution process, utilizing genetic algorithms. We implement a genetic algorithm (GA) that generates potential solutions for CPPAR. In order to compare against the PRC algorithm in a controlled manner, we utilize genetic search to generate the task groupings, and then perform sequential greedy path generation from the resulting groupings. We also implemented a genetic search method for path generation, but the performance was nearly identical to the sequential greedy approach, so no separate graphs are shown.

While there is a significant body of existing work using genetic algorithms for clustering, the majority of work reviewed makes assumptions that do not hold for our problem. In some cases it assumed that k, the number of clusters, is a known fixed value [14, 15], which violates our assumption that the number of agents (or paths) is not known a priori. Other work represents solutions as a set of fixed centroids [66, 53], which would not permit the path structure necessary for high value solutions. Finally, some representations rely on having explicit cluster labels [35, 36], while our task groupings are interchangeable and not assigned to a specific agent identity.

The CPPAR problem has another important distinction from existing work in clustering, as some tasks may not be chosen for any path, excluding them from being included in a cluster. We address this by introducing a *null cluster*, which always exists and can not be created or removed. Any tasks in the null cluster are not used when paths are generated, while still allowing clusters to be formed over all tasks in the problem. To represent solutions, we draw inspiration from existing methods for using genetic algorithms to group sets of objects [7, 25]. A solution, i.e., an individual chromosome, is represented as an array of |T| integers, with T begin the set of tasks. Each entry is the cluster number that

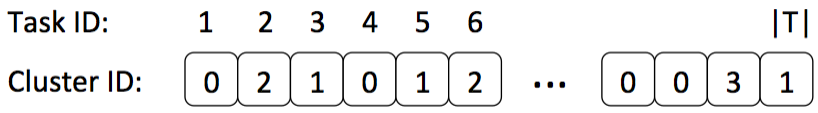
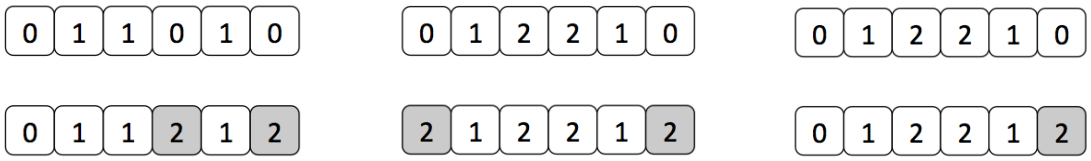


Fig. 3.22: The chromosome representation used for the genetic algorithm. Each entry represents a single task, with the value being the ID of the current cluster the task belongs to. Cluster 0 represents the null coalition, meaning the task is not part of any cluster that is visited by an agent.

the respective task belongs to, with 0 representing the null cluster, meaning the task is not selected for visit in the solution.

Cole [25] has shown limited effects of using traditional crossover operators, such as one point and two point crossovers, for clustering; He suggested the use of split, merge, and move operators for clustering with GAs. Our GA utilizes elitism, which keeps the most promising solutions in the population and applies genetic operators to search for better solutions. In order to construct the population for the next generation, we select the top 10% solutions to remain in population without modification. For each of the solutions in the top 10%, we make 5 copies (representing 50% of the new population) and apply a genetic operator, as per the selected distributions described in the results below. We then iteratively add the next highest ranking solutions to the population, applying a genetic operator to each solution according to the selected distribution. We terminate the process when the population has reached its original size minus one, as we reserve one spot in the next population for a randomly generated new solution to ensure continued diversity throughout the evolutionary process. This process is conducted until there is no improvement of expected value for the best solution after 100 generations. In this work, we utilize the following three genetic operators from Cole [25]:

- Split: The split operator chooses a random cluster and splits off a random number of members into a new cluster. The null coalition may also be chosen as the source cluster, but not as a new cluster.



(a) Split (b) Merge (c) Move

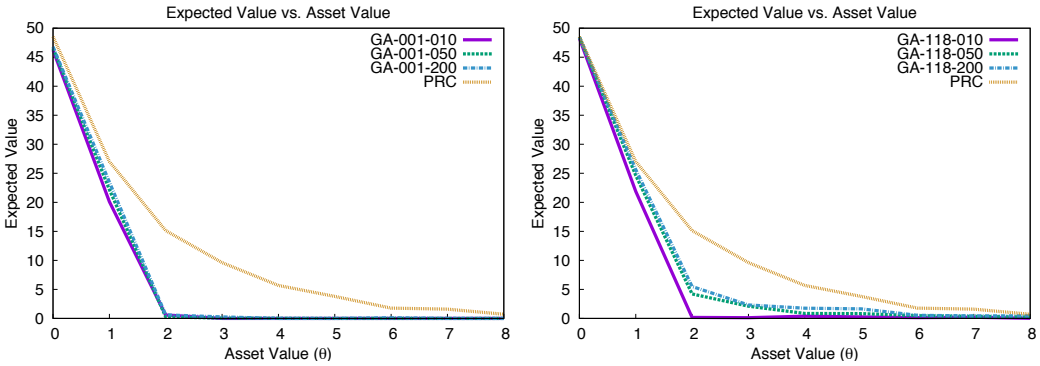
Fig. 3.23: Genetic operators used for clustering: (a) Split will randomly split an existing cluster to generate a new cluster. (b) Merge joins two existing clusters to form a single cluster from all members. (c) Move randomly swaps members of a single cluster to the cluster with nearest task.

- Merge: A merge chooses two random clusters and merges them into a single cluster. The null coalition may be one of these clusters chosen.

- Move: The move operator chooses a random cluster and randomly moves tasks to the cluster that contains the member with the nearest task.

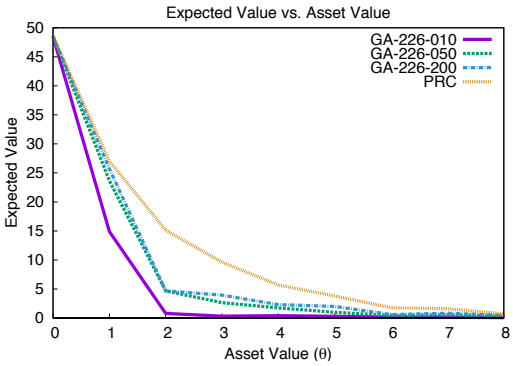
The combination of these operators allows the algorithm to explore the search space. Different probabilities applied to the genetic operators are explored, with split and merge having the same probability to prevent bias towards solutions with one large cluster or each task in its own cluster. In each generation, once a partition is performed by the GA, we use the same sequential greedy path generation process as the PRC algorithm in order to evaluate the effectiveness of the GA, making this a comparison between only the clustering methods, i.e., GA vs. PRC. We also conducted experiments with various population sizes, 10, 50, and 200.

In Figure 3.24, we observe that the genetic algorithm approach is outperformed by PRC across the spectrum of operator probabilities and different population sizes, more so when $\theta = 2$, where our PRC algorithm also showed the largest gains in expected value over the sequential greedy baseline. These results are also achieved at the expense of a significantly longer execution time, shown in Figure 3.25. The long execution times for the GA approach are caused by having to construct a solution in order to evaluate each solution in every generation of the population. However, it is worth mentioning that with long time

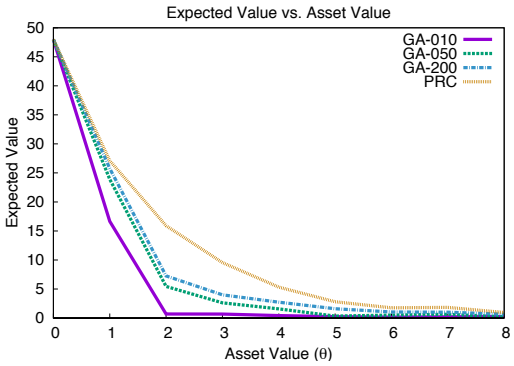


(a) Split: 0, Merge: 0, Move: 1

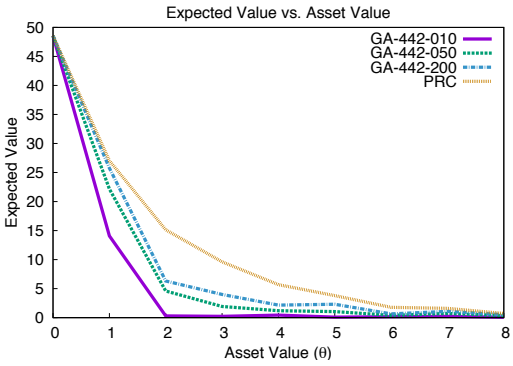
(b) Split: 0.1, Merge: 0.1, Move: 0.8



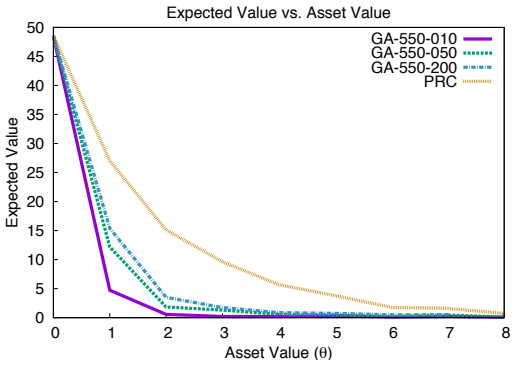
(c) Split: 0.2, Merge: 0.2, Move: 0.6



(d) Split: 0.3, Merge: 0.3, Move: 0.4



(e) Split: 0.4, Merge: 0.4, Move: 0.2



(f) Split: 0.5, Merge: 0.5, Move: 0

Fig. 3.24: The expected value of multi-agent plans generated by the genetic algorithm (GA) with different operator distributions, and with population size 10, 50 and 200, compared against our PRC algorithm on 100 task problem instances.

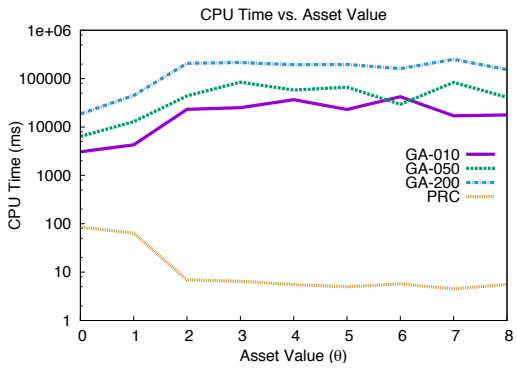


Fig. 3.25: The CPU time expended by the genetic algorithm (GA) with operator distribution split: 0.3, merge: 0.3, move: 0.4, and with population size 10, 50 and 200, and our PRC algorithm to find solutions for 100 task problem instances.

constraints, a large pool of computing resources, and a carefully designed set of genetic operators, we expect the genetic algorithm approach would perform better.

When θ is small (< 1), when a large number of agents should be sent to do relatively few tasks each, genetic search is an effective strategy. In this case, there is limited interactions between the paths and significant gain can be made performing the move operation, which swaps tasks between two agents. However, as θ increases, the complexity of the problem grows, requiring a longer series of operators to be executed in order to see an improvement in expected value. Additionally, there are additional local maxima that require a sequence of genetic operations to break out of, which may be lost as the value of the interim chromosomes is relatively low compared to other population member. As a result, we speculate that the genetic algorithm approach relies on a series of specific operators in order to find good solutions. Given these results, we conclude that further modifications need to be made to the genetic operators to further improve solution quality and reduce run time. One of the main reasons PRC's performance is that PRC was specifically designed for the multi-agent path finding problem, CPPAR, inherently containing some domain specific heuristics. We conclude that new genetic operators specific to CPPAR would improve GAs performance, but such a direction is beyond the scope of this dissertation, requiring major research efforts. We have also shown that search space for CPPAR solutions is often

complex, further motivating heuristics that exploit problem structure.

3.6 Summary of Contributions

Inspired by turnpike theorems, which focus on the largest gains in the problem space, we have developed a method for iteratively generating solution for the Collection Planning Problem with Attrition Risk using hierarchical clustering. We explore the the aspects of a cluster that must be considered during execution, providing insight into what contributes to cluster value. Based on these insights, we explore community detection and clustering methods that are flexible enough to be used without specifying the number of clusters as part of the problem definition.

Based on the promising results of experiments using hierarchical clustering, we implement 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 using an approximation methods without a significant loss of performance. We compare this method against an alternative randomized search procedure, with similar results. Finally, we show the effects of redundant collection of tasks, showing that it is most effective when asset values are relatively low, and with diminishing returns.

In the next chapter, we explore methods for distributing the cost of a path among members of the agent path, allowing us to distribute the cost to each task. We present a coalitional game theory formulation for CPPAR solutions, and prove that cost allocations are stable and fairly distributed among the member tasks for each path.

4
Hedonic Coalition Formation for CPPAR

We introduce a hedonic coalitional game formulation for the Collection Planning Problem with Attrition Risk that includes a risk of losing agents during execution as part of the planning process. To motivate this formulation, consider an *provider* that makes vehicles available to service tasks for *customers*, which could be individuals or other organizations. For each task serviced, the provider must determine how much to charge the customer in order to cover the cost the asset, θ, with respect to the risk of losing the asset when it is sent to collect. Because multiple customers may be serviced by a single asset, the provider must determine a fair allocation of cost to the customers for their respective tasks. If an allocation is not fair, the customers will desire to find another provider.

It may seem intuitive to allocate the cost of an asset equally among the tasks on the path being serviced, but we will show that this allocation is not fair for this problem and leads to undesirable instability. Consider two task requests for an image to be collected. One request is an aerial shot of an event taking place right next to the provider's base station, and the other is requesting a photo of a building located 3 miles away. A single vehicle is dispatched to collect on both tasks, requiring 6.1 miles of total travel at an estimated cost

of $40.

If we allocate the cost of the collection equally among the customers, each paying $20, the first customer will be charged far more than if their task alone was serviced by a vehicle. This is because the vast majority of the path distance, and the cost associated with the risk of losing the asset, is due to servicing the second task. Therefore, in order to keep customers, the provider should aim to allocate the cost such that each customer is happy with what they will pay for their task. In this example, if a fair allocation can not be found, the provider should consider dispatching an additional vehicle and charging each customer for a direct path to their task.

We provide a formulation for a hedonic coalitional formation game in which each customer seeks to minimize the cost paid for a task. Using this formulation, we devise a cost allocation method that yields fair allocations that are stable. This formulation allows us to distribute the cost of using an asset among the tasks being serviced in a manner that is fair and stable, making both the provider and customers happy.

We use the resulting cost allocation method to design an algorithm that outputs fair and stable cost allocations as paths for CPPAR. Du to the relative complexity of this algorithm, it is not well-suited for solving our problems from scratch. However, by executing the algorithm on the approximate solution from our PRC algorithm in Chapter 3, we show that this approach can generate solutions that are not only fair and stable allocations, but also have a higher expected value overall.

The application of cooperative game theory to path-based coalitions has received some attention. Previous work on Traveling Salesman Games has shown that stable cost allocations exist for $n \leq 5$ [57] and also shown that when $n = 6$ and the distances are Euclidean there may be an empty core [34]. Discussion on a wide range of combinatorial optimization problems and the application of cooperative game theory can be found in [30].

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. We then provide a hedonic coalition formation game for CPPAR, with a cost allocation method that ensures key properties that ensure stability. Finally, we show the improvement this method can provide in simulated problem instances.

4.1 Coalitional Game Theory for CPPAR

Coalitional games, also referred to as cooperative games, require that a set of players come to an agreement on organizing themselves into one or more coalitions. Each coalition is a collection of players, and has an associated utility that is distributed as a payoff to the members of that coalition. This paradigm is beneficial, as it informs a distributed implementation of a process for finding path assignments with high expected value, which is key for ensuring resiliency and scalability in realistic operating environments where a central planner is unavailable or undesirable.

In this work, the tasks (and their respective customer) are the players and vehicle paths are the coalitions with tasks as members. Each path has an associated utility that is divided among the tasks. Each task seeks a coalition that maximizes its own utility, and a coalition is considered stable if no task can achieve a higher utility by leaving the coalition. We assume that payoffs are a transferable currency, which allows players to exchange utility freely as incentive to other players. We utilize the definitions provided in [20, 80] to formalize the descriptions of coalitions, with some notable modifications to account for tasks that are not visited. Having previously defined the process for ordering tasks on a path, we denote the coalition containing the tasks in a path π_j as P_j.

A coalition structure is defined as the set $\mathcal{P} = \{P_1, ..., P_p\}$, which assigns each member of the set of visited tasks T to a coalition $P_j \in \mathcal{P}$. These coalitions are equivalent to the paths used in the previous chapter. Any task not included in P is referred to as being in the

null coalition $P_\emptyset$. A task in the null coalition is not visited by any agent path and generates no cost or utility. Given a coalition structure $\mathcal{P}$, the coalition to which task t_i belongs is denoted as $S_\mathcal{P}(t_i) \in \mathcal{P} \cup P_\emptyset$.

4.1.1 Utility of Coalitions and Tasks

In the previous chapter, we provided methods to maximize the overall expected utility, which we denote here as the sum over the utility of all coalitions,

$$U(\mathcal{P}) = \sum_{P_j \in \mathcal{P}} u(P_j) \tag{4.1}$$

Utility is defined as the the value of completing one or more tasks minus the cost of servicing the tasks. A single vehicle represents a coalition of tasks, having a value that is the sum of the task values, and a total cost of exposing the vehicle to risk. A customer is assigned the value for its own completion, and the cost that is allocated to it by the provider for use of the vehicle.

Coalition Utility

In basic terms, we can compute the overall utility of a coailition P_j as the value of that coalition minus the cost of the coalition,

$$u(P_j) = v(P_j) - c(P_j). \tag{4.2}$$

The utility of the null coalition is $u(P_\emptyset) = 0$, as these tasks are not visited by any vehicle. Otherwise, the value of the coalition is defined as, derived from Equation 2.6,

$$v(P_j) = \sum_{t_i \in P_j} v_i(P_j) = \sum_{t_i \in P_j} S_j g_i \tag{4.3}$$

where S_j is the survivability of the path formed from coalition P_j (see Equation 2.3),

and g_i is the gain for completing task t_i. The notation is changed from from the original formulation, from score c_i to gain g_i. This reflects the use of a monetary exchange, as well as more clearly differentiating this value from the cost function $c_i(\cdot)$. The total cost of the coalition is,

$$c(P_j) = (1 - S_j)\theta. \tag{4.4}$$

The goal is to provide an allocation of this coalition cost to each of the tasks that results in no task preferring to leave a coalition that is part of the globally maximum solution.

Task Utility

The utility of a path consists of both the value of the tasks being completed and the cost of risking the asset to perform collection. The utility of a single task t_i in coalition π_j is,

$$u_i(P_j) = v_i(P_j) - c_i(P_j). \tag{4.5}$$

The individual task value is also based on the survivability of the entire path, denoted as,

$$v_i(P_j) = S_j g_i. \tag{4.6}$$

The cost incurred by task t_i in coalition P_j is denoted as $c_i(P_j)$, and methods for determining this value will be discussed in the next section. For problems where order of visits is important, there is literature that describes more appropriate cost allocation schemes [39]. In this work, we provide an allocation based on the relative path distance, regardless of the direction the agent travels on the path.

For each task $t_i \in T$ a preference relation $\succeq_i$ is a complete, reflexive, and transitive binary relation defined over the set $\{P_k \subseteq T | t_i \in P_k\} \cup P_\emptyset$. If a task t_i is given two coalitions $P_1 \subseteq T$ and $P_2 \subseteq T$, where $t_i \in P_1$ and $t_i \in P_2$, $P_1 \succeq_i P_2$ indicates that task t_i

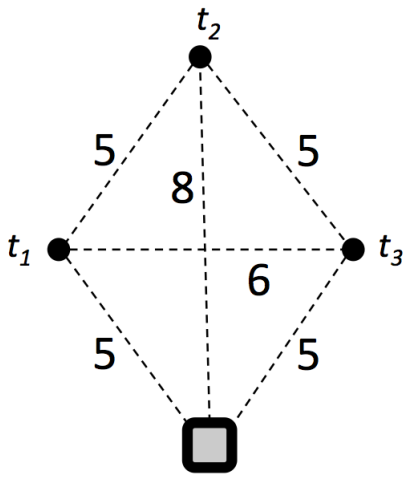


Fig. 4.1: Simple example problem with a base (bottom) and three tasks t_1, t_2, and t_3, with all pairwise distances denoted. In this example, the task values are $g_1 = g_2 = g_3 = 1$, the cost per asset is $\theta = 1$, and the survival probability per unit traveled is $\psi = 0.97$.

prefers to be a member of coalition P_1 over being a member of coalition P_2, or prefers the coalitions equally. A stronger preference is indicated by $P_1 \succ_i P_2$, in which task t_i strictly prefers coalition P_1 over P_2.

The preferences of a task is defined based on the utility that the agent receives from a coalition, such that,

$$P_1 \succeq_i P_2 \iff u_i(P_1) \geq u_i(P_2) \tag{4.7}$$

where $u_i : 2^{\mathcal{P}} \to \mathbb{R}$ is a function defined for each task. Put simply, each task prefers the coalition that yields the maximal utility for that task.

Consider the example shown in Figure 4.1 with 3 tasks, with task values $c_1 = c_2 = c_3 = 1$, cost per asset of $\theta = 1$, and the survival probability per unit traveled is $\psi = 0.97$. The survivability of a path is ψ raised to the power of the distance of a path. For example, the path traveling to task t_1 and back to the base with distance 10 has a survivability of $\psi^1 0$. The utility for each of the six possible coalitions is as follows:

$$u(\{t_1\}) = \psi^{10} g_1 - (1 - \psi^{10})\theta = .4748$$

$$u(\{t_2\}) = \psi^{16} g_2 - (1 - \psi^{16})\theta = .2286$$

$$u(\{t_3\}) = \psi^{10} g_3 - (1 - \psi^{10})\theta = .4748$$

$$u(\{t_1, t_2\}) = \psi^{18}(g_1 + g_2) - (1 - \psi^{18})\theta = .7339$$

$$u(\{t_1, t_3\}) = \psi^{16}(g_1 + g_3) - (1 - \psi^{16})\theta = .8439$$

$$u(\{t_2, t_3\}) = \psi^{18}(g_2 + g_3) - (1 - \psi^{18})\theta = .7339$$

$$u(\{t_1, t_2, t_3\}) = \psi^{20}(g_1 + g_2 + g_3) - (1 - \psi^{20})\theta = 1.175$$

The maximum overall value is achieved by either of two symmetric solutions, $U(\{\{t_1, t_2\}, \{t_3\}\}) = 1.2087$ and $U(\{\{t_1\}, \{t_2, t_3\}\}) = 1.2087$. For this example, we will discuss the solution $\{\{t_1, t_2\}, \{t_3\}\}$, focusing the coalition formed by tasks t_1 and t_2. Recall that the utility for an agent is composed of a value and a cost function. The value earned by each agent is derived from the survivability of the path visiting the tasks, and is not transferable between tasks. Therefore, we seek to allocate the cost of the collector among the tasks being serviced.

4.1.2 Cost Allocation

Consider a task joining an existing coalition, which incurs a longer distance to be traveled by the collector, consequently reducing the survivability of the collector. As a result, not only does the cost of the risk to the collector increase, but the value provides to all members of the coalition is correspondingly decreased. Therefore, in order to provide a fair allocation of the cost, the joining task must consider both the cost and value that are offset by its entry into the coalition.

We assume that the value of the task is non-transferable, meaning that one task can not

offer the value gained to another task as compensation. Instead, we must determine how the cost will be allocated among the members of a coalition. In Figure 4.2, we provide an example of coalitions that would form among tasks, with each task having a local value and a cost that must be shared among the member tasks.

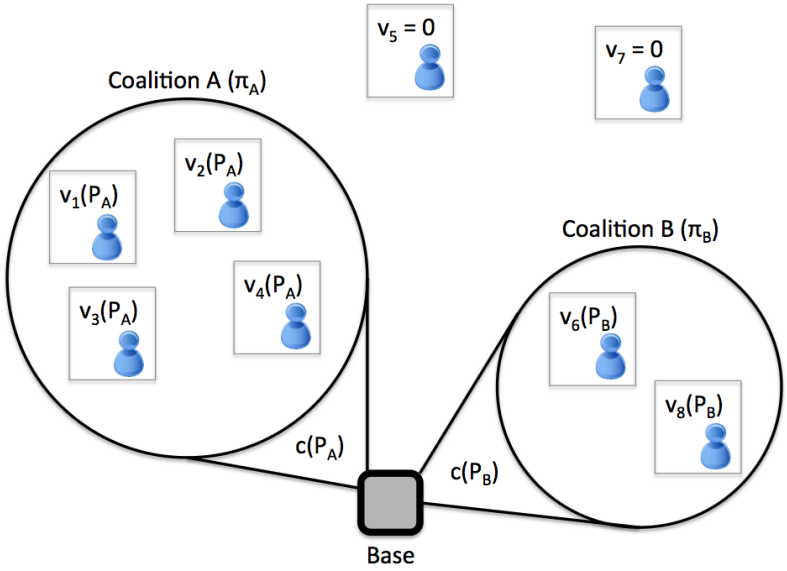


Fig. 4.2: An example of 2 coalitions that would be formed for tasks in a problem instance. Each task has a value that is non-transferable, and each coalition must allocate the cost of the asset to be sent. Tasks not in a coalition are not visited and provide no value or cost.

We desire a cost allocation method that ensures stability and fairness. Potters and Curiels [77] propose three conditions that should be satisfied:

1. **Efficiency**: The sum of the cost for each task sums up to the total cost for the coalition.

2. **Individual rationality**: Each task is willing to accept, at most, the cost paid for a direct path from the base to the task and back.

3. **Minimal Obligation**: Each tasks pays no less than its marginal cost.

We will show that equal allocation of cost violates these conditions, and introduce a cost allocation method, along with a proof that these conditions are met for a solution that maximized global utility.

Equal allocation

The most straightforward cost allocation method is to split the cost of the collecting asset equally among the tasks being collected. Let $c_i(P_j)$ be the cost paid by the owner of task t_i being collected by agent a_j, with S_j being the survivability (Equation 2.3) of coalition P_j. Each task pays an equal share of the total cost,

$$c_i(P_j) = \frac{(1 - S_j)\theta_j}{|P_j|}. \tag{4.8}$$

We show that this is not a valid cost allocation by counterexample. Using once again the example in Figure 4.1, the total cost of the coalition $\{t_1, t_2\}$ is defined as $c(\{t_1, t_2\}) = (1 - \psi^{18})\theta = .422$. This yields the following utility for the individual tasks,

$$u_1(\{t_1, t_2\}) = v_1(\{t_1, t_2\}) - c_1(\{t_1, t_2\}) = .578 - .422/2 = 0.367$$
$$u_2(\{t_1, t_2\}) = v_2(\{t_1, t_2\}) - c_2(\{t_1, t_2\}) = .578 - .422/2 = 0.367$$

With this cost allocation, $u_1(\{1\}) > u_1(\{1, 2\})$ (.4748 > 0.367), which means that task t_1 would prefer to be in a singleton coalition rather than be in a coalition with task t_2. This violates our requirement for individual rationality, therefore an equal cost allocation is not desirable.

Cost-value proportional allocation

We propose a cost allocation method for CPPAR that considers the the relative increase in risk that each task brings to the coalition. This allocation is a modified version of the *alternative costs avoided* method described in [13]. The total cost for the coalition can be split into separable and non-separable costs. The separable cost, also referred to as the marginal cost, represent the cost share attributable to a single task. The non-separable costs are shared by the tasks, and must be allocated further.

The marginal cost is for task t_i in coalition P_j is denoted as m_{i,P_j}, and takes into account both the increased risk of losing the collector and the decreased value incurred by all other tasks in the coalition. Recall that $v_k(P_j)$ is the individual value for task $t_k \in P_j$ as part of the path formed by coalition P_j.

$$m_{i,P_j} = \left(\sum_{t_k \in P_j \setminus \{i\}} v_k(P_j \setminus \{i\}) - v_k(P_j) \right) + \left(c(P_j) - c(P_j \setminus \{i\}) \right) \tag{4.9}$$

If the marginal cost exceeds the total cost of the collector, $m_{i,P_j} > (1 - S_j)\theta$, then task t_i is better off not joining the coalition, as it could gain equal or greater value for, at most, a cost of θ.

Continuing to use the example in Figure 4.1, we compute the marginal cost for both t_1 and t_2 in coalition $\{1, 2\}$,

$$m_{1,\{1,2\}} = (.6143 - .578) + (.422 - .3857) = .0726$$
$$m_{2,\{1,2\}} = (.7374 - .578) + (.422 - .2626) = .3188$$

These values reflect the decrease in overall utility that each member of the coalition is responsible for. The non-separable residual cost is the remaining cost that can not be directly attributed to a single task. The non-separable cost of coalition P_j is the remaining cost after subtracting the separable costs, referred to as the *residual cost* and denoted as,

$$c^R(P_j) = c(P_j) - \sum_{k \in P_j} m_{k,P_j} \tag{4.10}$$

The non-separable residual cost is the remaining cost that can not be directly attributed to a single task, and any remaining cost is shared equally among the members of the coalition. This cost is split equally among the agents, with the individual cost allocation of task t_i denoted as,

$$c_i^R(P_j) = m_{i,P_j} + \frac{w_i}{\sum_{k \in P_j} w_k} c^R(P_j). \tag{4.11}$$

For the residual allocation, w_i is the weight for task t_i, based on the savings for t_i when joining P_j if it had to only pay its separable cost,

$$w_i = c(\{t_i\}) - (v(\{t_i\}) - v_i(P_j)) - m_{i,P_j}. \tag{4.12}$$

The utility equation for a task t_i in coalition P_j is,

$$u_i(P_j) = v_i(P_j) - c_i^R(P_j). \tag{4.13}$$

We now show the derived utility for the coalition $\{t_1, t_2\}$ in the example in Figure 4.1, using the cost-value gap method of allocation. We first compute the residual of the coalition $c^R(\{t_1, t_2\}) = c(\{t_1, t_2\}) - \sum_{t_i \in P_j} m_{i,P_j} = .422 - (.0726 + .3188) = .0306$. This value is split using the weights for each task, $w_1 = .2626 - (.7374 - .578) - .0726 = .0306$ and $w_2 = .3857 - (.6143 - .578) - .3188 = 0.0306$. We can now compute the cost allocation for each task, $c_1^R = .0726 + \frac{1}{2}.0306 = .0879$ and $c_2^R = .3188 + \frac{1}{2}.0306 = .3341$.

$$u_1(\{t_1, t_2\}) = .578 - .0879 = .4901$$
$$u_2(\{t_1, t_2\}) = .578 - .3341 = .2439$$

In this example, these values satisfy the efficiency requirement, such that $u(\{t_1, t_2\}) = u_1(\{t_1, t_2\}) + u_2(\{t_1, t_2\})$, meaning that all of the utility provided to the coalition has been accounted for. They account for individual rationality, with $u_1(\{t_1, t_2\}) > u_1(\{t_1\})$ and $u_2(\{t_1, t_2\}) > u_2(\{t_2\})$, so that neither task is motivated to leave the coalition to form a singleton coalition. Finally, both agents fulfill their minimal obligation, such that $c_1(\{t_1, t_2\}) \geq c_1^R(\{t_1, t_2\})$ and $c_2(\{t_1, t_2\}) \geq c_2^R(\{t_1, t_2\})$. We now show that these prop-

erties hold in the general case.

Proposition 1. The cost-value proportional allocation has the efficiency property when a coalition maximizes expected value.

Proof. We must show that the utility of the coalition is distributed exactly among the members. For members of the null coalition $c(P_\emptyset) = c_i(P_\emptyset) = 0$, for all $t_i \in P_\emptyset$. For a coalition with a single task, $c(P_j) = c_i(P_j)$ if $P_j = \{t_i\}$. For the general case, we must show that $u(P_j) = \sum_{t_i \in P_J} u_i(P_j)$.

$$
\begin{aligned}
u(P_j) &= \sum_{t_i \in P_J} u_i(P_j) \\
v(P_j) - c(P_j) &= \sum_{t_i \in P_J} v_i(P_j) - \sum_{t_i \in P_J} c_i^R(P_j) && \text{By Equations 4.4,4.13} \\
\sum_{t_i \in P_j} S_j g_i - c(P_j) &= \sum_{t_i \in P_J} S_j g_i - \sum_{t_i \in P_J} c_i^R(P_j) && \text{By Equations 4.3,4.6} \\
c(P_j) &= \sum_{t_i \in P_J} c_i^R(P_j) \\
c(P_j) &= \sum_{t_i \in P_J} \left(m_{i,P_j} + \frac{w_i}{\sum_{k \in P_j} w_k} c^R(P_j) \right) && \text{By Equation 4.11} \\
c(P_j) &= \sum_{t_i \in P_J} m_{i,P_j} + \sum_{t_i \in P_J} \frac{w_i}{\sum_{k \in P_j} w_k} c^R(P_j) \\
c(P_j) &= \sum_{t_i \in P_J} m_{i,P_j} + c^R(P_j) \\
c(P_j) &= c(P_j) && \text{By Equation 4.10}
\end{aligned}
$$

Proposition 2. The cost-value proportional allocation accounts for individual rationality when a coalition maximizes expected value.

Proof. We must show that if there is a coalition P_j such that the global utility is maximized, $\sum_{t_k \in P_j} u(\{t_k\}) \leq u(P_j)$, $t_i \in P_j$ then, $u(\{t_i\}) \leq u_i(P_j)$, ensuring no member tasks would be better off if they opted to be in a singleton coalition. We can show this via

equivalence.

$$u(\{t_i\}) \leq u_i(P_j)$$

$$v(\{t_i\}) - c(\{t_i\}) \leq v_i(P_j) - c_i^R(P_j) \qquad \text{By Eq. 4.2,4.13}$$

$$v(\{t_i\}) - c(\{t_i\}) \leq v_i(P_j) - \left(m_{i,P_j} + \frac{w_i}{\sum\limits_{k \in P_j} w_k} c^R(P_j) \right) \qquad \text{By Eq. 4.11}$$

$$v(\{t_i\}) - c(\{t_i\}) \leq v_i(P_j) - (c(\{t_i\}) - v(\{t_i\}) + v_i(P_j) - w_i) - \frac{w_i}{\sum\limits_{k \in P_j} w_k} c^R(P_j) \qquad \text{By Eq. 4.12}$$

$$0 \leq w_i - \frac{w_i}{\sum\limits_{k \in P_j} w_k} c^R(P_j)$$

$$\frac{w_i}{\sum\limits_{k \in P_j} w_k} c^R(P_j) \leq w_i$$

$$c^R(P_j) \leq \sum_{k \in P_j} w_k$$

$$c(P_j) - \sum_{k \in P_j} m_{k,P_j} \leq \sum_{k \in P_j} (c(\{t_k\}) - (v(\{t_k\}) - v_k(P_j)) - m_{k,P_j}) \qquad \text{By Eq. 4.10,4.12}$$

$$c(P_j) - \sum_{k \in P_j} m_{k,P_j} \leq \sum_{k \in P_j} c(\{t_k\}) - \sum_{k \in P_j} v(\{t_k\}) + \sum_{k \in P_j} v_k(P_j) - \sum_{k \in P_j} m_{k,P_j}$$

$$c(P_j) \leq \sum_{k \in P_j} c(\{t_k\}) - \sum_{k \in P_j} v(\{t_k\}) + \sum_{k \in P_j} v_k(P_j)$$

$$\sum_{k \in P_j} (v(\{t_k\}) - c(\{t_k\})) \leq \sum_{k \in P_j} v_k(P_j) - c(P_j) \qquad \text{By Eq. 4.13,4.2}$$

$$\sum_{k \in P_j} u(\{t_k\}) \leq u(P_j)$$

Therefore, $u(\{t_i\}) \leq u_i(P_j)$ if and only if $\sum_{k \in P_j} u(\{t_k\}) \leq u(P_j)$, ensuring no agent would receive a cost allocation that would give it incentive it to leave the coalition and form a singleton. $\square$

Proposition 3. The cost-value proportional allocation enforces minimal obligation when a coalition maximizes expected value.

Proof. By Equation 4.11, the marginal cost for task t_i in coalition P_j is $c_i^R(P_j) = m_{i,P_j} + \frac{w_i}{\sum_{k \in P_j} w_k} c^R(P_j)$. It remains to show that $c^R(P_j) \geq 0$. Recall from Equation 4.10, $c(P_j) = \sum_{t_i \in P_j} m_{i,P_j} + c^R(P_j)$. If $c^R(P_j) < 0$, the there must be some task $t_i \in P_j$ with a marginal that exceeds $u(\{t_i\})$, which would require that the coalition does not maximize expected value. $\square$.

4.2 Hedonic Coalition Formation Algorithm for CP-PAR

A *hedonic coalitional game* imposes two additional constraints; the payoff of a player is dependent only on the members of the coalition that the player belongs to, and coalitions are formed based on the preferences of players over the set of possible coalitions. Therefore, each player must be able to compare coalitions that is can be a member of, and indicate an ordering to indicate which coalition is preferable. We adopt a similar set of definitions to existing work in hedonic coalition formation games [20, 80].

Definition 1. (Switch Rule). Given a coalition structure $\mathcal{P} = \{P_1, ..., P_p\}$ and a set of tasks $T = \{t_1, ..., t_m\}$, a task t_i will choose to leave its current coalition $S_{\mathcal{P}}(t_i)$ and join another coalition $P_k \in \mathcal{P} \cup P_\emptyset$, $P_k \neq S_{\mathcal{P}}(t_i)$, if and only if $P_k \cup \{t_i\} \succ_i S_{\mathcal{P}}(t_i)$.

The switch rule provides the condition under which a t_i will choose to perform a switch operation, leaving its current coalition for another. This represents a self-interested decision, as it does not rely on any notion of maximizing the global utility, nor does it consider the effect on the utility of other agents. This evaluation also allows an agent to consider

changing the coalition structure based only on a subset of problem information, which can be desirable for distributing the computational workload of the algorithm.

In order to ensure stability of the proposed algorithm, we introduce a coalition history for each task t_i, denoted as $h(t_i)$. This records all coalitions that t_i has previously been a member of, allowing us to prevent the task from rejoining a previous coalition and entering a cycle. The modified utility function for task t_i in coalition P_j is,

$$u_i(P_j) = \begin{cases} 0, & \text{if } P_j \in h(t_i) \\ v_i(P_j) - c_i^R(P_j), & \text{otherwise} \end{cases} \tag{4.14}$$

When no agent desires to leave its current coalition, the solution is considered to be stable. This also indicates that the algorithm has converged, and no further operations will be performed. Formally, a set of coalitions $P = \{P_1, ..., P_p\}$ is Nash-stable if $\forall t_i \in T$, $S_{\mathcal{P}}(t_i) \succeq_i P_k \cup \{t_i\}$, for all $P_k \in \mathcal{P} \cup P_\emptyset$.

Proposition 4. A coalition structure $\mathcal{P}$ resulting from the hedonic coalition formation algorithm in Table 4.1 is Nash-stable.

Proof. Assume we are given as a converged output of the algorithm a coalition structure $\mathcal{P}$ or which no task has incentive to leave its current coalition using the utility function in Equation 4.14. If $\mathcal{P}$ is not Nash-stable, then there exists a task t_i and a coalition $P_j \in \mathcal{P}$ such that $P_k \cup \{t_i\} \succ_i S_{\mathcal{P}}(t_i)$. Therefore, task t_i could perform a switch operation, which contradicts $\mathcal{P}$ being the result of convergence of the algorithm. Therefore, any coalition structure $\mathcal{P}$ resulting from HCFA is Nash-stable.

This algorithm uses the approximate solution provided by PRC as a starting point, then allowing for individual tasks to bid on better paths as a method of increasing utility. No task will join an existing path that yields a loss greater than the gain realized by making the switch, which ensures that any switch will increase the global utility, as well. Therefore, we expect that this 2 phase process will yield solutions of higher expected value than using PRC alone. In the next section, we confirm this expectation.

Table 4.1: Hedonic Coalition Formation Algorithm (HCFA) for the Collection Planning Problem with Attrition Risk

Input: Task set $T = \{t_1, ..., t_n\}$, graph $G =< V, E >$ where $V = T \cup \{b\}$.
Output: Coalition structure $\mathcal{P}_{\text{out}} = \{P_1, ..., P_p\}$.

Phase 1. Approximate Clustering

1. Execute Progressive Risk-aware Clustering algorithm as described in Chapter 3 [51], generating approximate coalitions.
2. Store resulting task coalitions as $\mathcal{P}$.

Phase 2. Hedonic Coalition Formation

1. **while** there exists a task t_i such that $P_k \cup \{t_i\} \succ_i S_{\mathcal{P}}(t_i)$ **do**
   (a) All tasks $t_i \in T$ checks the potential switch operations, choosing the operation that maximizes its payoff.
   (b) The task with the highest expected gain in payoff t_m is chosen to execute the switch operation:
      i. The coalition history for the task $h(t_m)$ is updated, adding $S_{\mathcal{P}}(t_i)$.
      ii. Task t_m is removed from its current coalition $S_{\mathcal{P}}(t_i)$.
      iii. Task t_m is added to the coalition that maximizes its payoff.
2. Return the converged coalition structure $\mathcal{P}$ as $\mathcal{P}_{\text{out}}$.

4.3 Performance and Evaluation

We evaluate the performance of the Hedonic Coalition Formation Algorithm (HCFA) on 100 randomly generated problem instances for each parameter setting. 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.

Due to the nature of the problem space, executing Phase 2 of the algorithm from an empty set of coalitions would have limited success. Expanding paths in an incremental process, as is the case with the sequential greedy algorithm used in [51], has been shown to have relatively poor performance. Instead, clustering-based approaches allow for solutions with a higher expected value, as shown by our previous work developing the Progressive Risk-aware Clustering (PRC) algorithm [51]. Because PRC is an approximate algorithm, we expect that there is room for improvement, a performance gap that can be closed using our hedonic coalition formation process.

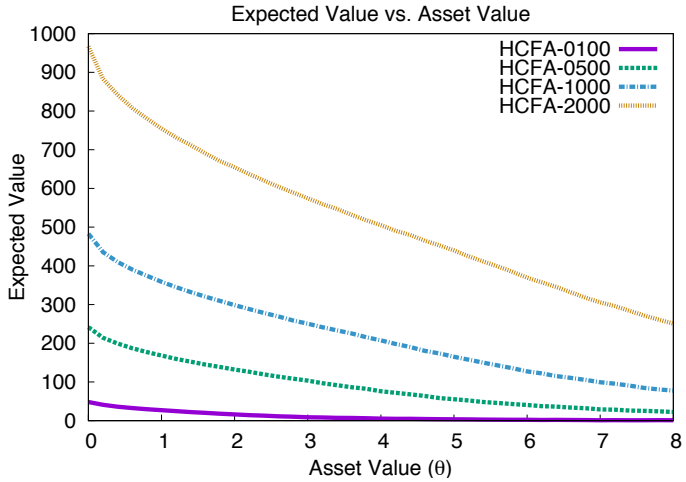


Fig. 4.3: The expected value of multi-agent plans generated by HCFA on 100, 500, 1000, and 2000 tasks.

In Figure 4.3, we show the resulting performance of HCFA, which outputs expected values that are consistent with previous results on the CPPAR [51]. Additionally, in Figure 4.4 we see an increase in performance over the base PRC algorithm by applying the hedonic coalition formation process. For large problems with 2000 tasks, we observe an

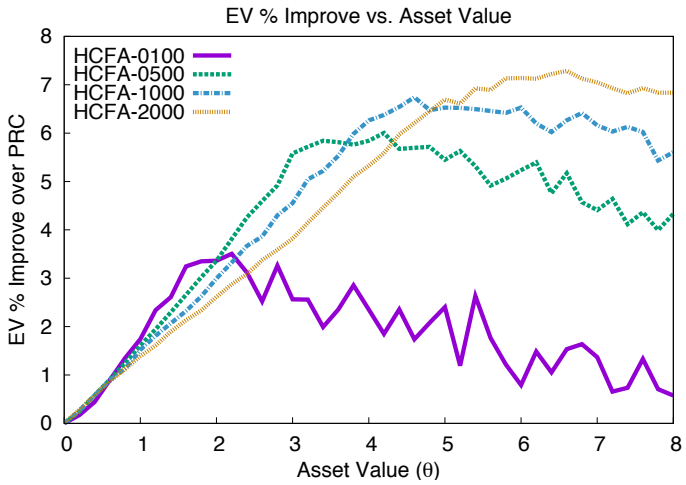


Fig. 4.4: The % gain in expected value of using the hedonic coalition formation over using PRC alone on 100, 500, 1000, and 2000 tasks.

approximate gain of 7%, for problems with a high asset value.

We have provided a hedonic coalition formation game for the collection planning problem with attrition risk (CPPAR) and applied it to improve the expected value of approximate solutions on a range of problem instances. We have also provided a cost allocation method that is efficient, individually rational, and fulfills the minimal obligation for each task.

This formalism provides a mechanism for allocating cost to customers who are requesting tasks be fulfilled by a servicing organization that will charge for risk incurred by their vehicles. We see further potential to integrate the cost allocation scheme with a procedural clustering method in a distributed framework, providing stable solutions without requiring the two phases of path formation and improvement. Additionally, further study is needed on the effects of varying task values, heterogeneous asset capabilities and values, communication of collected data through an ad hoc network, and the allowing for redundancy of collection. In the next chapter, we discuss an alternative form of risk based on uncertainty of costs of collection at a task location.

5
Resource-Constrained Risk for Collection Planning

Having provided methods for dealing with attrition risk, we now discuss another form of probabilistic risk that alters how we find solutions for path-based optimization problems. We address the challenge of collecting data that can be located at multiple sites, but with an uncertain cost of collection. We may wish to dispatch vehicles to collect data from locations that have uncertain conditions, such as weather, and may require a variable amount of effort. For example, if collecting imagery, foggy conditions will require the platform move closer to the target before collecting, requiring additional energy expenditure. Therefore, there is uncertainty surrounding the cost of collection that can only be dispelled by sending a sensor-laden vehicle to the site.

This problem seeks solutions that minimize cost and minimize the probability of failure, objectives that are in conflict. Additionally, conducting these missions requires human oversight to assure objectives are met. The decision maker may have a preference prior to search, allowing the search to focus on a single objective. But commonly, the decision maker may wish to evaluate a set of alternatives after analyzing a set of alternatives against their own preference [71]. In the latter case, a *solution set* needs to be generated that spans

the objective space.

If the decision maker is able to provide additional budget or probability of failure constraints, the goal is to find a path that meets one of two objective functions, both of which have been shown to be NP-complete [55] on general graphs:

- **Minimize budget**: given a probability of failure p^*_{fail}, minimize the budget necessary to ensure an active site is located with certainty of at least $1 - p^*_{fail}$. This answers the question, "How much budget will I need to ensure the risk of failure is below acceptable levels?"

- **Minimize probability of failure**: given a fixed starting budget B^*, minimize the probability of failing to find an active site, while ensuring the sum of travel and purchase costs do not exceed B^*. This addresses, "What risk of failure can I expect given this limited budget?"

A user may want to be presented with a range of options representing the trade-off between budget and risk. For example, the mission may be a part of a larger operation and is one of many competing objectives, which requires evaluating the overall efficiency of the search process to determine if it should be carried out at all, or if additional resources are needed. If a decision maker has a requirement for a maximum travel cost, or minimum risk, we can optimize the solution for the other objective. Otherwise, we must present the user with a set of alternatives that they can choose from. While similar formulations exist [55, 46], to our knowledge this is the first work to provide solutions for this multiobjective optimization problem.

This problem is similar to a Traveling Salesman Problem and its variations (see [44] for a comprehensive review), but varies in signficant ways. In this work, each site has a distinct probability of failure that differentiates it from other sites, imposing a preference order. More recent work extended the TSP to include features that assign value to visited locations [6, 86], but these models assume that costs are fixed and known. The Orienteering

Problem with Stochastic Profits (OPSP) introduces a distribution over the profits at each site and seeks to meet a profit threshold within a given time limit [90].

Another key difference of this domain from prior work is the minimization of multiple competing objectives. Multi-objective optimization for path planning has been approached using evolutionary algorithms [54], but has often been limited to simple cycles on graphs. The Traveling Purchaser Problem with Stochastic Prices [55] introduces the model used in this work, but generates solutions by generalizing the cost distribution at each site as the expected cost, which we compare against in our evaluation in Section 5.4. More recent work has provided solutions for maximizing probability with a fixed budget or minimizing budget with a fixed probability [46, 21], but not both simultaneously.

5.1 Stochastic Physical Search Problem (SPSP)

We define the problem as a stochastic physical search problem (SPSP), where we are given an undirected network $G(S^+, E)$ with a set of sites $S^+ = S \cup \{o, d\}$ where $S = \{s_1, ..., s_m\}$ is the set of m sites that may be active, o and d are the origin and destination locations. We are also given a set of edges $E = \{(i, j)] : i, j \in S^+\}$. Each (i, j) has the cost of travel t_{ij} that is deterministic and known. An agent must start at origin node o, visit a subset of sites in S to find an active site, and then end at the destination point d.

The cost of collecting the data at each site $s_i \in S$ is an independent random variable C_i with an associated probability mass function $P_i(c)$, which gives the probability that determining if a site is active will cost c at site s_i. A solution is a path π, a list of length n where $\pi(i)$ and $\pi(i+1)$ are consecutive locations along a path, with $\pi(1) = o$ and $\pi(n) = d$. We define $\pi(i, j)$ as the the sub-path of π containing consecutive path locations $\pi(i)$ through $\pi(j)$. The objective is to provide a path that minimizes the cost c and probability of failure p_{fail}.

This cost at site may be infinity, which indicates the site is not active and no information

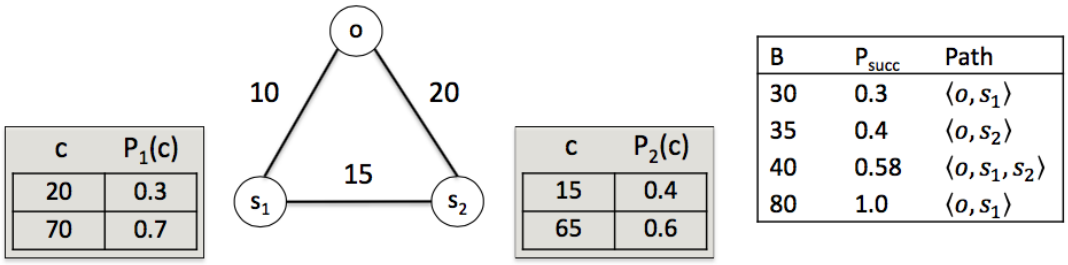


Fig. 5.1: A simple instance of a 2 site SPSP with the agent starting at location s. For each site s_i, there is a probability $P_i(c)$ of being able to collect the data at cost c. The table (right) indicates the budget thresholds at which the agent can achieve the corresponding probability of success, $P_{succ} = 1 = p_{fail}$, via the path.

can be gained, or 0 to indicate there is no cost. We refer to this specialization as a *Binary SPSP*. A site s_i is active with some probability p_i, or inactive with probability $1 - p_i$. In Figure 5.2 we show a small instance of a Binary SPSP with 5 sites and the associated pareto set of path solutions. The longest solution path, the shortest path that visits all sites, is also the solution to the Traveling Salesman Path Problem [44].

5.2 Multiobjective optimization of SPSP

A multi-objective optimization problems (MOP) represents a trade-off between competing objectives. Efficient exploration of the solution space is difficult in the case where MOP criteria share dependencies on a common set of variables, such as budget and probability of failure, in this case. The following definitions are derived from multiobjective optimization literature [97], with adaptations for the specifics of the B-SPSP.

In general, a multi-objective problem is characterized by a vector of r decision variables, $x = (x_1, x_2, ..., x_r) \in X$, and k criteria. There is an evaluation function $F : X \rightarrow A$, where $A = (a_1, a_2, ...a_k)$ represents the k attributes of interest. We represent the fitness of vector x as $F(x) = (f_1(x), f_2(x), ...f_k(x))$, where $f_i(x)$ is the mapping from the decision variable vector to a single attribute a_i.

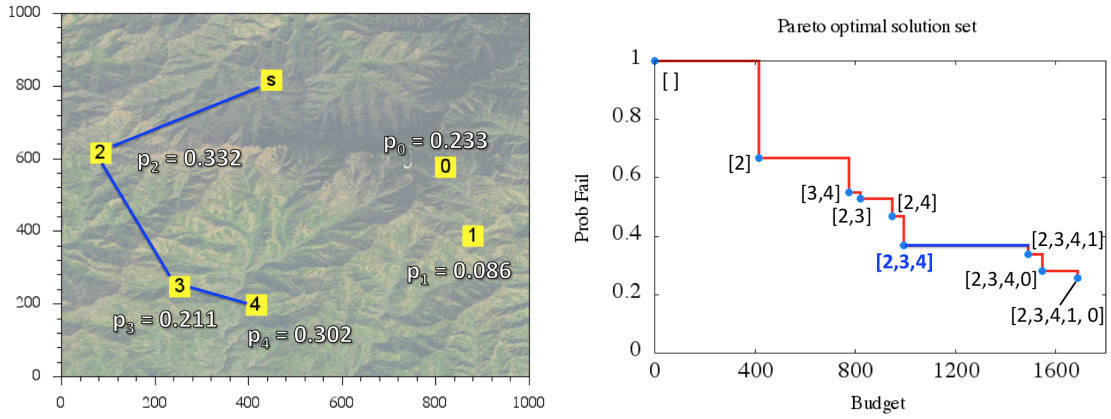


Fig. 5.2: A CPPRR instance with starting location s and 5 sites. Below each site is the probability that the site is active. The set of solutions contains a collection of nondominated paths and their associated cost and probability. Each point is labeled with the path, which is the order of site visits starting from s. A single path (blue) is shown on both graphs, for reference.

Definition 1.1 (Multiobjective Optimization for SPSP) We define a set of decision variables x denoting a path, a set of 2 objective functions for cost, $f_1 : X \to \mathbb{R}$, and probability of failure, $f_2 : X \to [0, 1]$, and a set of m constraints that require a decision vector be a valid path. We use probability of failure for clarity of definition, allowing us to minimize both objectives. The optimization goal is to

$$\begin{aligned} &\text{minimize} && F(x) = (f_1(x), f_2(x)) \\ &\text{subject to} && e(x) = (e_1(x), e_2(x), ..., e_m(x)) \leq 0 \\ &\text{where} && x = (x_1, x_2, ..., x_n) \in X \end{aligned} \tag{5.1}$$

The *feasible set* X_f is defined as the set of decision vectors in X that form a valid path, with each decision vector x satisfying the path constraints $e(x)$, and such that $X_f = \{x \in X | e(x) \leq 0\}$. While we would prefer solutions that provide the minimum probability of failure at the minimum cost without violating the constraints, the objectives do not have optima that correspond to the same solution.

Definition 1.2. For a pair of objective vectors u and v,

$$\begin{aligned} u = v \quad &\text{iff} \quad \forall i \in \{1, 2, ..., k\} : u_i = v_i \\ u \leq v \quad &\text{iff} \quad \forall i \in \{1, 2, ..., k\} : u_i \leq v_i \\ u < v \quad &\text{iff} \quad u \leq v \wedge u \neq v \end{aligned} \tag{5.2}$$

The relations $\geq$ and $>$ are defined similarly.

These definitions impose a partial order on solutions. When none of these relations hold between u and v, we can only state that $u \not\succeq v$ and $u \not\preceq v$, meaning neither is superior. A solution that provides higher probability and higher cost than another solution is just as strong and may prove more effective if the additional budget is available to spend.

Definition 1.3 (Pareto Dominance) For any decision vectors b and c,

$$\begin{aligned} b \succ c \text{ (b dominates c)} \quad &\text{iff} \quad F(b) < F(c) \\ b \succeq c \text{ (b weakly dominates c)} \quad &\text{iff} \quad F(b) \leq F(c) \\ b \backsim c \text{ (b is indifferent to c)} \quad &\text{iff} \quad F(b) \not\leq F(c) \wedge F(b) \not\geq F(c) \end{aligned} \tag{5.3}$$

The definitions for "dominated by" ($\prec, \preceq, \backsim$) are analogous.

An optimal solution is one that can not be improved by any other solution in the feasible set. This does not include solutions that are indifferent, as they are not comparable within the partial order imposed by pareto dominance. A decision vector $x \in X_f$ is nondominated with respect to a set $D \subseteq X_f$ iff $\nexists a \in D : a \succ x$. We refer to a solution x as Pareto optimal iff x is nondominated by X_f. Because the set of all solutions may be a partial order, there can be two or more Pareto optimal solutions for a given problem instance. The collection of all nondominated solutions with respect to the set of all solutions is referred to as the **nondominated solution set**.

Let $D \subseteq X_f$. The function $n(D) = \{d \in D : d \text{ is nondominated regarding } D\}$ outputs

the set of nondominated decision vectors in D. The objective vectors in $f(n(D))$ is the *nondominated front* regarding D. The set $X_n = n(X_f)$ is referred to as the Pareto-optimal set (Pareto set). We use the term **solution set** to denote both nondominated decision set and their associated objective values. Solution sets represent the output of a solver that can be used to evaluate performance on problem instances.

5.2.1 Comparing solution sets

We desire a measure to compare solution sets, so that we can determine the trade-offs between using different solution techniques. Because the budget values are not normalized, and will vary greatly with respect to the number of sites, it is important that any measure be scale-independent. We use a measure similar to the $\mathcal{S}$ metric [97] on a two-dimensional space.

Definition 1.4 (Size of solution set) Let $D = (d_1, d_2, ..., d_m) \subseteq X$ be a set of decision vectors, ordered by f_1. The function $\mathcal{S}(D)$ returns the union of the area enclosed by the objective values for each vector d_i. The area enclosed by a single decision vector d_i is a rectangle defined by the points ($f_1(d_{i+1})$, 0) and ($f_1(d_i)$, $f_2(d_i)$). If $i = m$, then let $f_1(d_{i+1}) = f_1(d_i)$.

Using the $\mathcal{S}$ metric, we can derive a measure of error with respect to the optimal solution.

Definition 1.5 (Solution set error) Let $B^* = (x_1^*, x_2^*, ..., x_m^*) \subseteq X_f$ be the optimal set of decision vectors. Given a decision vector $D = (x_1, x_2, ..., x_m) \subseteq X$ we define the error of C as

$$\mathcal{E}(D) = \left(\frac{\mathcal{S}(D)}{\mathcal{S}(B^*)} \right) - 1 \tag{5.4}$$

A solution set should seek to minimize the $\mathcal{E}$ metric, with $\mathcal{E}(D) = 0$ indicating solution set D is optimal.

5.3 Algorithm

The Iterative Domination Solver (IDS) starts by generating all paths containing one market and adding them to the active set. Each iteration of of the algorithm consists of two phases: the *search phase* and the *domination phase*. In the search phase, remaining sites are added to all paths in the active set. This generates a new set of candidate paths that are added to the active set. In the domination phase, the solver evaluates new candidate paths and determines if they are dominated or dominate any other solutions. Depending on the filter depth parameter, dominated solution are removed from the active set, preventing any further search using that path. This procedure allows us to focus search on high value sub-paths without expending significant effort on paths that are low value and unlikely to be found on a non-dominated path.

5.3.1 Search Phase

A site is added to the previous path immediately after the start site or previous to the destination site, which ever adds lower total travel cost. This operation prevents insertions that break up edges from the last iteration, preserving inter-site edges that exist in sub-paths. An example of this operation is shown in Figure 5.3. This process continues, iteratively adding new markets to the candidate paths, until we have the path that visits all sites in the problem, at which point the process terminates and the non-dominated set of intervals is returned.

We can reduce the search space via a *filter depth* (fd) parameter, which is path length

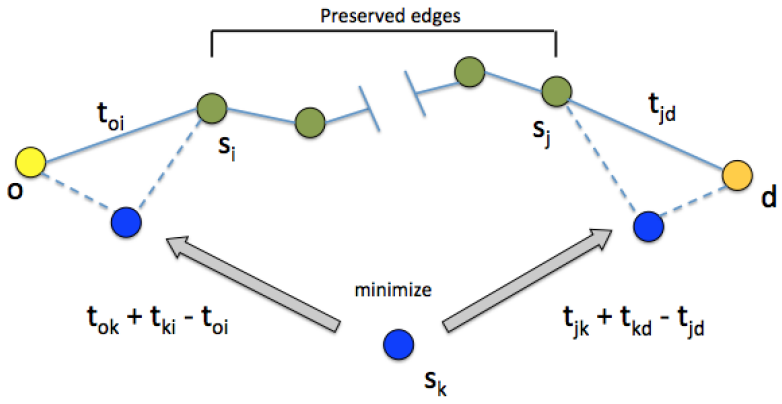


Fig. 5.3: Illustration of the insertion step for the IDS algorithm. A new node s_k is added to the search path after the origin site or before the destination site, whichever has the lower added path cost. This preserves the edges from the previous path in the search process, exploiting nondominated substructure.

at which we start pruning out dominated solutions. For example, if we set $fd = 1$, only a single path consisting of (o, s_i, d) will remain after initial path generation, and all paths of length 2 must contain s_i. Then, if the only nondominated size 2 path contains s_i and s_j, all resulting paths from the next iteration must contain that ordered pair, and so on. Using filter depth is an effective pruning strategy because it prefers edges with shorter path lengths and there is a diminishing return on probability as the problem size increases.

5.3.2 Domination phase

A quad tree [38] is a rooted data structure where a node may have up to 4 children. For our purposes, each child represents a quadrant in the 2D plane of budget and probability of failure. This is defined recursively, making it an extension of a binary tree to 2 dimensions. The quad tree data structure allows us to store the value pairs (cost, probability) and their associated paths, while also detecting when a new solution is dominated or dominates another solution.

Because the number of possible paths is exponential with respect to the number of sites [79], storing all solutions is unreasonable. When a domination relation is detected, we use the structure to search for entries that should be removed or preserved and restructure the

tree as necessary. This adds additional overhead during storage of individual solutions, but allows us to limit the space complexity of storing candidate solutions. The efficiency of searching this structure is sensitive to proper selection of the initial root node, an optimization still being investigated. In this work we choose to seed the tree with a random path using half of the markets, but better methods are certain to exist.

5.4 Experimental Results

We evaluate our approach using randomly generated problem instances and comparing performance against a number of intuitive approaches to solving the B-SPSP. Site networks are formed by placing sites randomly in a 1000×1000, with travel cost t_{ij} being the Euclidean distance between sites s_i and s_j. The probability of failure at each sites is drawn from a Gaussian distribution with varying mean and variance. All results are based on 100 instances, with error bars indicating a 95% confidence interval. We use a naming convention based on the chosen filter depth. If $fd = k$ then we refer to the algorithm as IDS-k.

In order to determine the relative performance of IDA, we compare against a solver for these problem types that minimizes Expected Cost (EC), adapted from the approach outlined in [55]. Changes to the algorithm are the acquisition of only a single 'item', and the removal of item cost, which is 0 in this domain.

At each step, EC will provide a path the minimizes the overall cost of the complete path, without consideration for budget or some fixed probability of success. In order to generate a solution set from this single path, we include all sub-paths of the solution beginning at the starting site. This gives us a step-wise set of intervals that still meet the criterion of minimizing expected cost. We also include a greedy solution that generates a set of solutions by selecting the lowest cost edge available.

Using the $\mathcal{E}$ metric defined in Section 5.2.1, we can compute the error of each approach

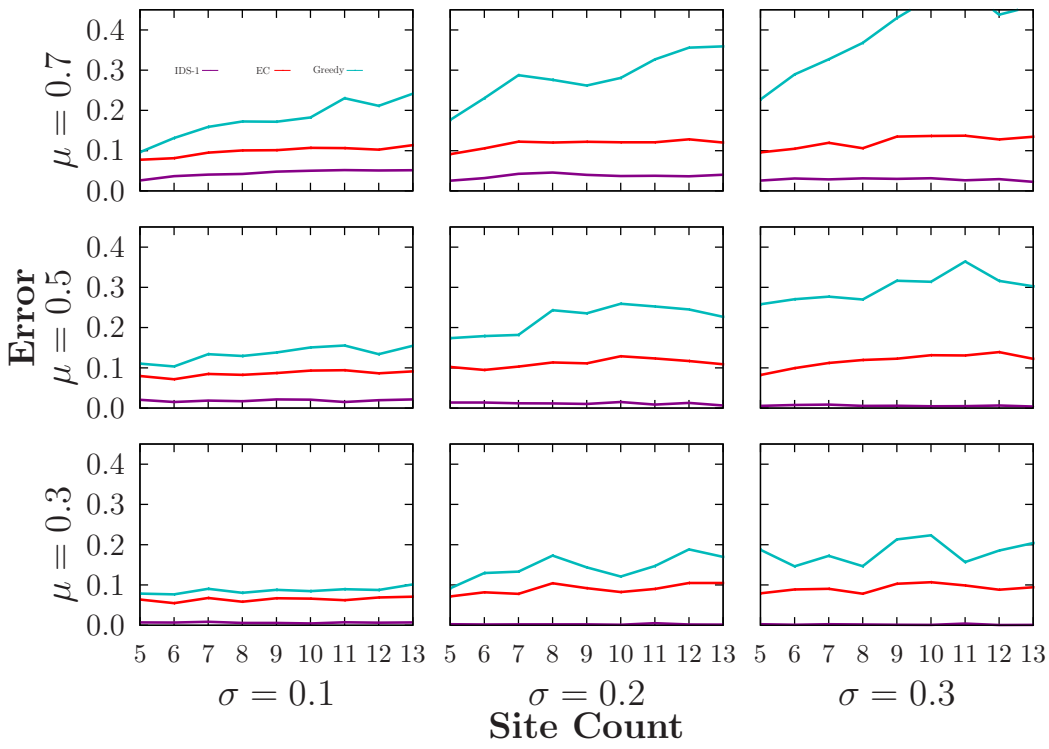


Fig. 5.4: Comparison of error on random instances with respect to mean and variance of probability of failure on sites

with respect to optimal. In Figure 5.4, we show the effectiveness of IDS with respect to both changes in the mean and variance of the probability of failure. As the mean probability of failure increases, all 3 algorithms perform closer to optimal. This is due mostly to each site individually moving closer to 0% failure, leaving less room for error, in general.

As the variance on the probability is increased, expected cost seems relatively unaffected, the greedy solver performs significantly worse, and IDS sees significant improvement. Increased variance distinguishes the site probabilities, making higher cost edges feasible options for paths that may be non-dominated. This makes the single path with low average cost less likely to cover the space of non-dominated solutions.

In Figure 5.5 we show the execution time of IDS with expected cost, optimal and a random solver that generates 30,000 randomly selected paths. IDS completes search orders of magnitude faster than optimal and EC, mainly due to being able to terminate search before expanding the search tree on paths that are not of good quality. In Figure 5.6 we

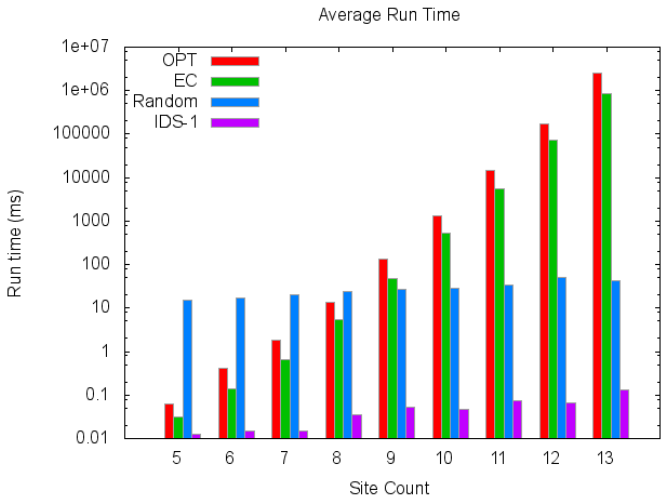


Fig. 5.5: Comparison of average execution time among solvers.

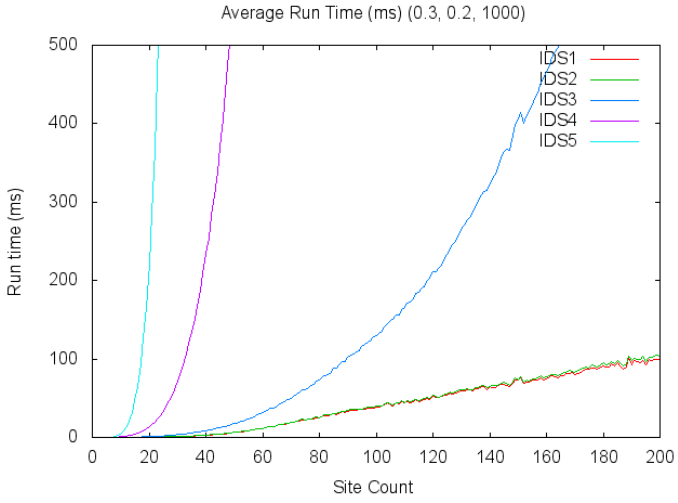


Fig. 5.6: Execution time for various IDS filter depths.

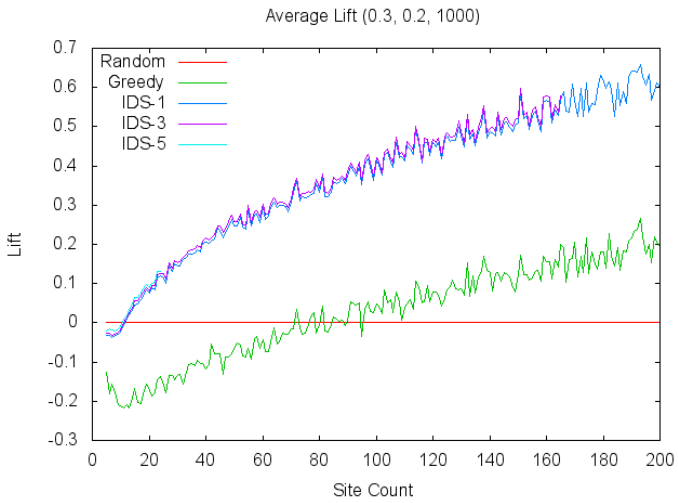


Fig. 5.7: Lift of IDS and greedy versus random sampling

show execution time of IDS with respect to varied filter depths.

Because the optimal solution requires exponential time to solve as the problem size increases, it is difficult to compare performance against optimal for large instances. To show that the performance of IDS is maintained for large problem instances, we compare against a random solver, which generates 30,000 random paths and generates a solution set from all enumerated sub-paths. We define the lift of set D as $\left(\frac{\mathcal{S}(D)}{\mathcal{S}(R)}\right) - 1$, where R is the nondominated solution set generated by the random solver. The results of this analysis are shown in Figure 5.7. While increasing the filter depth provides minor gains in performance, the increased computational demand is significant, especially for large instances.

We have presented B-SPSP as a method for solving CPPRR, providing analysis of the problem space and a characterization of nondominated solution sets. We have also presented the Iterative Domination Solver as an approximate method for generating solution sets. We have shown that IDS performs better than existing approaches across a wide range of problem parameters with orders of magnitude less execution time on randomly generated instances. While the filter depth can be increased for minor performance gains on instances of the CPPRR, in most cases IDS-1 will prove sufficient. In the next chapter, we discuss the many opportunities for future work in risk-aware planning.

6
Conclusions & Future Work

In this thesis, we have introduced a new type of path-based planning problem with a risk of losing assets during collection of sensor data, referred to as the Collection Planning Problem with Attrition Risk (CPPAR). This problem is motivated by the emergence of small, low-cost air vehicles, which have the potential to become a invaluable tool for performing collection of data for both military and civilian applications.

We consider a planning problem in which there is a risk of losing agents during execution, and that risk must be dealt with as part of the planning process prior to execution. Additionally, the number of platforms used is not specified by the user, but is rather informed by the magnitude and nature of the risk in the environment. As a result, we can provide solutions that are robust to a wide range of operating conditions.

We have provided a clustering method for determining the allocation and path ordering for agents, referred to as Progressive Risk-Aware Cluster (PRC), showing that it outperforms a sequential greedy solutions in both solution quality and execution time. To do so we have modified a hierarchical clustering process to consider the unique aspects of this problem, including task value, inter-task distance, and the distance of the cluster from the base. For digital goods that can be collected by more than one agent, we have introduced a problem formulation that included redundancy of collection. In this case, more than one

agent can perform collection, and the task is completed if any one of the agents returns to the base with the collected data. We show improvement in expected value when the asset value is low relative to the value of the tasks, and observe that there are diminishing returns on redundancy that limit gains as we add more agents to a single task.

Using coalitional game theory, we have specified a mechanism for distributed computation of task clusters, which includes a method for fair allocation of the costs for each platform among the visited tasks. This mechanism can be used by a servicing organization to charge customers for completing tasks in a manner that is fair and stable. We show that our cost allocation method meets the criterion for generating stable allocations, by meeting expectations of efficiency, individual rationality, and minimal obligation. We use this method to further improve our clustering results, while simultaneously measuring the gap between the clustering output and a stable solution.

Finally, we explored an alternative representation of risk, based on limited resources being used to perform tasks with a probability distribution over the cost of collection. We provide an algorithm for computing a non-dominated front of solutions that seek to maximize both cost and the probability of success, providing a user with a set of alternatives. Our approach outperforms monte carlo methods, providing in most cases better performance with a significant reduction in CPU time.

6.1 Future Research

While we have laid the groundwork in this thesis, there is a vast space of new research opportunities that remain to be explored. Some of these opportunities are fundamental questions that still remain, while other arise from more specific applications. Each of these can provide a valuable contribution to the research community.

6.1.1 Communication

One can imagine that if each collecting agent were outfitted with communication hardware, then collected data could be transmitted back to the base before the agent has returned, mitigating some of the risk. While full communication among all agents and the base is preferable, it's practical to assume that distance and signal limitation will result in scenarios in which only a subset of agents can communicate at any given time. This introduces a number of challenges, each of which requiring significant attention.

With communication available, there is value in transmitting each task's data to any other agent that will listen, and have them transmit it to the base at the earliest moment possible. However, storage on the platforms is likely to be limited, and one platform may not be able to store all of the data for every single task. As a result, each agent may want to be selective about what ti transmits and when. If the data for one task has been replicated to many other agents already, it may be more valuable to transmit a different task's data to the next agent in range. Computational methods for computing the value of transmitting a certain task to another agent, taking into account storage limitations, the history of previous transmission, and the paths of agents through the environment, could begin to solve this challenge.

Constraints in the communication itself may also provide restrictions that an agent should consider when communicating. Bandwidth may be limited, preventing too many agents from communicating at once. The data begin transmitted may be large enough to require a time window to transmit, so agents should consider their movement vectors to determine if they have enough time to complete the replication process. Finally, for mobile agents in ad hoc networks, there is the challenge of maintaining routing tables for multi-hop transmission, which may be unreliable at high velocities.

The introduction of dynamically appearing tasks also provide challenges due to communication. Even if re-planning is available, an agent will not be able to schedule collection of a task it does not know about. Information about what tasks were detected needs to be

propagated to the agents before they can make plans for collection, and this relies on the communications infrastructure available. If resources are scarce, the agents may wish to avoid saturating the network with new task requests, especially to agents that are not within feasible range to perform collection.

All of these challenges may not only be a decision to be made by the agent during execution, but could included as part of a planning process. An agent may choose to be a transmission node, providing a bridge between other agents and the base. This could be part of the global expected value function, motivating an additional layer of planning for the communication infrastructure. Additionally, agents may want to structure their paths to maximize the time they spend within communication range, increasing the likelihood of successful replication of their respective task data.

6.1.2 Heterogeneous Platforms and tasks

In this work, we have assumed that each platform has the same relative value and capabilities for collection. We may have a collection of different platforms, each with its own value, in which case we may want to allocate high-valued assets to lower risk collections, and send the lower valued assets out on longer collection paths with increased risk. Alternatively, each platform may vary in the value gained by performing specific tasks. For example, a platform with a higher resolution camera may provide an image that increases the value of a task, if collected.

Another consideration is platforms that have different modalities of operation, allowing for collection of only a subset of the tasks. For example, we may have tasks that require infrared imagery, or capturing a signal at a specific frequency, and no single platform is of sufficient size or power to collect everything on its own. This requires new methods of planning for paths, and may also motivate a problem where each platforms is outfitted with a subset of sensors as part of the planning process.

6.1.3 Varied Threat Models

We have modeled threats as a uniform probability of the asset being destroyed per unit moved. This can represent passive threats, such as weather, or active threats with an unknown location within the environment. However, if more information is available about specific threats, this information could be provided to the agent as part of the planning process.

Some examples of threat information could be the location of threats with a radius of influence, or dynamic threat models that increase the threat as the agent is detected by the adversary. Alternatively, there may be information about what regions of the area of operation are considered more dangerous, allowing the agent to compute the risk on each edge based on how it intersects those regions.

6.1.4 Risk-aware Planning with Dynamic Replanning

We do not address replanning during execution in this work, but rather focus on how we can anticipate risk as part of the planning process. While dynamic replanning can be used with plans that do not consider risk, we expect the combination of the two will have additive benefits. For example, because our risk-aware planning models do not assume how many agents will be used a priori, the number of agents dispatched will effect how effective the dynamic replanning may be.

If tasks are time-sensitive or appear dynamically, some form of replanning will be necessary to ensure they are completed if they appear after execution has started. Alternatively, we may want to plan for ‘waves’ of agents that are dispatched in response to he evolving environment.

6.1.5 Tipping and Cueing

Until now, we have treated task data as if it was something only to be delivered, with the contents to be interpreted by a human analyst. With sufficient computational resources, we could consider scenarios in which a platform can process the collected data and use it to inform further collection activities. For example, an agent may collect data that contains a list of coordinates for other tasks of interest. By processing this list, the agent can then generate task requests and an associated value.

As discussed above in the communication section, the methodology for sending this new task information through the network may introduce new challenges. Consider a new task list with hundreds of additional tasks to be collected. If the current mission is relatively short and fuel limited, it may be better to not send any tasks, but rather deliver the data for use in a follow-on collection. On the other hand, if the set of platforms are conducting persistent operations over a long period of time, the agent will need a way to distribute the task information in a way that doesn't negatively effect communications that could be used for task data.

6.1.6 Quality vs. Computation Trade-off with Redundancy

In Chapter 3 we explored the effects of allowing for redundant collection of tasks by multiple assets. Under certain conditions, the benefits can be significant, nearly doubling the solution quality. In other cases, redundancy provided no additional value, making the increased computation without benefit. This raise a number of research problems to be addressed:

- Methods of pre-processing problem instances to determine if redundancy will provide sufficient gain to warrant the computation.
- The effects of varying task value, both independently and groupings of tasks nearby each other.

- Determining if we can alter the initial paths to provide a greater benefit from redundancy, rather than relying on our clustering technique.

These considerations can also be explored in combination with the other challenges presented in this chapter, providing a rich space of challenging problems.

6.1.7 Coalitional Formation Game with Redundancy

Traditionally, coalitional game theory is concerned with partitions, in which each task would be a member of a single coalition. We have explored this case Chapter 4, providing the foundation for additional extensions to overlapping coalitions. Extending this definition to redundant collection, as discussed in Section 3.4, remains an open problem. There is recent work in overlapping coalitional games [23] that could be leveraged to formulate this relatively complex mechanism. One must consider not only the benefit to a single task for having additional collectors visit, but also find ways to fairly allocate the task's contribution to each member coalition and provide a fair allocation of cost from each coalition that reflect the demand on collection resources.

6.1.8 Additional Opportunities

There are a number of concerns about security and privacy that are sure to become at the forefront of policy and regulation as unmanned air vehicles increase in accessibility. At the time of this dissertation, we have already witnessed near misses at airports and security incidents at federal government buildings. This has prompted calls for increased control over these vehicles.

As we enable these platforms with autonomous capabilities, we must also be sure that adversaries or other malicious actors are prevented from taking control of the aircraft and using them for unintended purposes. Additionally, the information stored during collection may be sensitive or proprietary, motivating new methods for securing data as it is collected

and during transport.

We are thankful to have been given the opportunity to contribute to this field at a time when unmanned vehicles are an emerging and exciting trend. We hope that our work will inform others and feed the imagination of our peers. We offer gratitude to the countless researchers before us, who enabled progress through their hard work and persistence.

REFERENCES

[1] E. Achtert and A. Z. Hans-peter Kriegel, "ELKI: A Software System for Evaluation of Subspace Clustering Algorithms," no. Ssdbm, pp. 580–585, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.3263

[2] S. M. Adams and C. J. Friedland, *A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management*. publisher not identified, 2011.

[3] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, and L. Liberti, "Column generation algorithms for exact modularity maximization in networks," *Physical Review E*, vol. 82, no. 4, p. 046112, 2010.

[4] H.-C. An, R. Kleinberg, and D. B. Shmoys, "Improving christofides' algorithm for the st path tsp," *Journal of the ACM (JACM)*, vol. 62, no. 5, p. 34, 2015.

[5] R. Andonov, V. Poirriez, and S. Rajopadhye, "Unbounded knapsack problem: Dynamic programming revisited," *European Journal of Operational Research*, vol. 123, no. 2, pp. 394–407, 2000.

[6] S. Arora and G. Karakostas, "A 2+ ε approximation algorithm for the k-mst problem," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2000, pp. 754–759.

[7] A. Banerjee, "An improved genetic algorithm for robust fuzzy clustering with unknown number of clusters," in *Fuzzy Information Processing Society (NAFIPS), 2010 Annual Meeting of the North American*. IEEE, 2010, pp. 1–6.

[8] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[9] J. E. Beasley, "Route firstâĂŤcluster second methods for vehicle routing," *Omega*, vol. 11, no. 4, pp. 403–408, 1983.

[10] T. Bektas, "The multiple traveling salesman problem : an overview of formulations and solution procedures," vol. 34, pp. 209–219, 2006.

[11] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton University Press, 2009.

[12] A. A. Bertossi, "The edge hamiltonian path problem is np-complete," *Information Processing Letters*, vol. 13, no. 4, pp. 157–159, 1981.

[13] N. Besozzi, L. Ruschetti, C. Rossignoli, and F. Strozzi, "The traveling salesman game for cost allocation: The case study of the bus service in castellanza," *Game Theory*, vol. 2014, 2014.

[14] J. C. Bezdek, S. Boggavarapu, L. O. Hall, and A. Bensaid, "Genetic algorithm guided clustering," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*. IEEE, 1994, pp. 34–39.

[15] J. N. Bhuyan, V. V. Raghavan, and V. K. Elayavalli, "Genetic algorithm for clustering with an ordered representation." in *ICGA*, 1991, pp. 408–415.

[16] N. Bianchessi, R. Mansini, and M. Speranza, "The distance constrained multiple vehicle traveling purchaser problem," *European Journal of Operational Research*, vol. 235, no. 1, pp. 73–87, May 2014. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0377221713008412

[17] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[18] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, "On the minimum latency problem," p. 9, 1994. [Online]. Available: http://arxiv.org/abs/math/9409223

[19] F. F. Boctor, G. Laporte, and J. Renaud, "Heuristics for the traveling purchaser problem," *Computers & Operations Research*, vol. 30, no. 4, pp. 491–504, Apr. 2003. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0305054802000205

[20] A. Bogomolnaia and M. O. Jackson, "The stability of hedonic coalition structures," *Games and Economic Behavior*, vol. 38, no. 2, pp. 201–230, 2002.

[21] S. Brown, Daniel, J. Hudack, and B. Banerjee, "Algorithms for stochastic physical search on general graphs," *AAAI Workshop on Planning, Search, and Optimization*, 2015.

[22] a. M. Campbell, D. Vandenbussche, and W. Hermann, "Routing for Relief Efforts," *Transportation Science*, vol. 42, no. 2, pp. 127–145, 2008.

[23] G. Chalkiadakis, E. Elkind, E. Markakis, M. Polukarov, and N. R. Jennings, "Cooperative games with overlapping coalitions," *Journal of Artificial Intelligence Research*, vol. 39, no. 1, pp. 179–216, 2010.

[24] H. L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[25] R. M. Cole, *Clustering with genetic algorithms*. Masters Thesis, University of Western Australia, 1998.

[26] M. Corcoran, "Drone journalism: Newsgathering applications of unmanned aerial vehicles (uavs) in covering conflict, civil unrest and disaster." 2015 [Online] Available: https://assets.documentcloud.org/documents/1034066/final-drone-journalism-during-conflict-civil.pdf.

[27] R. M. Cormack, "A review of classification," *Journal of the Royal Statistical Society. Series A (General)*, pp. 321–367, 1971.

[28] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[29] K. B. Culver, "From battlefield to newsroom: ethical implications of drone technology in journalism," *Journal of Mass Media Ethics*, vol. 29, no. 1, pp. 52–64, 2014.

[30] I. Curiel, *Cooperative game theory and applications: cooperative games arising from combinatorial optimization problems*. Springer Science & Business Media, 2013, vol. 16.

[31] M. Dror, "Cost allocation: the traveling salesman, binpacking, and the knapsack," *Applied Mathematics and Computation*, vol. 35, no. 2, pp. 191–207, 1990.

[32] E. Erkut and V. Verter, "Modeling of Transport Risk for Hazardous Materials," pp. 625–642, 1998.

[33] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[34] U. Faigle, W. Kern, and J. Kuipers, "Note computing the nucleolus of min-cost spanning tree games is np-hard," *International Journal of Game Theory*, vol. 27, no. 3, pp. 443–450, 1998.

[35] E. Falkenauer, "The grouping genetic algorithms-widening the scope of the gas," *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 33, no. 1, p. 2, 1992.

[36] ——, "A new representation and operators for genetic algorithms applied to grouping problems," *Evolutionary computation*, vol. 2, no. 2, pp. 123–144, 1994.

[37] D. Feillet, P. Dejax, and M. Gendreau, "Traveling Salesman Problems with Profits," *Transportation Science*, vol. 39, no. 2, pp. 188–205, May 2005. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/trsc.1030.0079

[38] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[39] M. Frisk, M. Göthe-Lundgren, K. Jörnsten, and M. Rönnqvist, "Cost allocation in collaborative forest transportation," *European Journal of Operational Research*, vol. 205, no. 2, pp. 448–458, 2010.

[40] M. R. Garey, D. S. Johnson, and R. E. Tarjan, "The planar hamiltonian circuit problem is np-complete," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 704–714, 1976.

[41] M. Gendreau, "Stochastic vehicle routing," vol. 7, no. 1992, 1996.

[42] B. P. Gerkey, "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[43] B. P. Gerkey and M. J. Mataric, "Multi-robot task allocation: Analyzing the complexity and optimality of key architectures," in *Robotics and Automation, 2003. Pro-*

ceedings. ICRA'03. IEEE International Conference on, vol. 3. IEEE, 2003, pp. 3862–3868.

[44] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations*. Springer Science & Business Media, 2006, vol. 12.

[45] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[46] N. Hazon, Y. Aumann, S. Kraus, and D. Sarne, "Physical search problems with probabilistic knowledge," *Artificial Intelligence*, vol. 196, pp. 26–52, Mar. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370213000027

[47] B. E. Helvik, "Dependable computing systems and communication networks; design and evaluation," *Department of Telematics, NTNU*, 2009.

[48] S. Herwitz, L. Johnson, S. Dunagan, R. Higgins, D. Sullivan, J. Zheng, B. Lobitz, J. Leung, B. Gallmeyer, M. Aoyagi *et al.*, "Imaging from an unmanned aerial vehicle: agricultural surveillance and decision support," *Computers and electronics in agriculture*, vol. 44, no. 1, pp. 49–61, 2004.

[49] G. A. Hollinger and J. A. Djugash, "Efficient Multi-robot Search for a Moving Target Efficient Multi-Robot Search for a Moving Target," pp. 201–219, 2009.

[50] P. H. Huang and T. L. Morin, "New turnpike theorems for the unbounded knapsack problem," *Technical Report, Purdue University*.

[51] J. Hudack and J. C. Oh, "Multi-agent sensor data collection with attrition risk," *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 2016.

[52] N. Hyafil and F. Bacchus, "Conformant probabilistic planning via csps." in *ICAPS*, vol. 98, 2003, pp. 205–214.

[53] C. Z. Janikow and Z. Michalewicz, "An experimental comparison of binary and floating point representations in genetic algorithms." in *ICGA*, 1991, pp. 31–36.

[54] N. Jozefowiez, F. Glover, and M. Laguna, "Multi-objective Meta-heuristics for the Traveling Salesman Problem with Profits," *Journal of Mathematical Modelling and Algorithms*, vol. 7, no. 2, pp. 177–195, Feb. 2008. [Online]. Available: http://link.springer.com/10.1007/s10852-008-9080-2

[55] S. Kang and Y. Ouyang, "The traveling purchaser problem with stochastic prices: Exact and approximate algorithms," *European Journal of Operational Research*, vol. 209, no. 3, pp. 265–272, Mar. 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0377221710006089

[56] J. Katrašnik, F. Pernuš, and B. Likar, "A survey of mobile robots for distribution power line inspection," *Power Delivery, IEEE Transactions on*, vol. 25, no. 1, pp. 485–493, 2010.

[57] J. Kuipers, "On the core of information graph games," *International Journal of Game Theory*, vol. 21, no. 4, pp. 339–350, 1993.

[58] M. G. Lagoudakis, S. Koenigt, and A. J. Kleywegt, "Simple Auctions with Performance Guarantees for Multi-Robot Task Allocation," 2004.

[59] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, pp. 345–358, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/037722179290192C

[60] ——, “Fifty years of vehicle routing,” *Transportation Science*, vol. 43, no. 4, pp. 408–416, 2009.

[61] J. LeBoeuf, “Practical applications of remote sensing technologyâĂŤan industry perspective,” *HortTechnology*, vol. 10, no. 3, pp. 475–480, 2000.

[62] C.-S. Liao and Y. Huang, “Generalized Canadian traveller problems,” *Journal of Combinatorial Optimization*, Apr. 2013. [Online]. Available: http://link.springer.com/10.1007/s10878-013-9614-z

[63] I. Little, S. Thiebaux *et al.*, “Probabilistic planning vs. replanning,” in *ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[64] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press, 2008, vol. 1.

[65] S. Martello, D. Pisinger, and P. Toth, “New trends in exact algorithms for the 0–1 knapsack problem,” *European Journal of Operational Research*, vol. 123, no. 2, pp. 325–332, 2000.

[66] U. Maulik and S. Bandyopadhyay, “Genetic algorithm-based clustering technique,” *Pattern recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.

[67] H. Min and N. Papanikolopoulos, “The multi-robot coverage problem for optimal coordinated search with an unknown number of robots,” *Robotics and Automation (ICRA), ...*, pp. 2866–2871, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5979995

[68] M. Mirik, R. J. Ansley, K. Steddom, D. C. Jones, C. M. Rush, G. J. Michels, and N. C. Elliott, “Remote distinction of a noxious weed (musk thistle: Carduusnutans) using airborne hyperspectral imagery and the support vector machine classifier,” *Remote Sensing*, vol. 5, no. 2, pp. 612–630, 2013.

[69] T. W. Neller and C. G. Presser, "Optimal play of the dice game pig," *The UMAP Journal*, vol. 25, no. 1, 2004.

[70] S. U. Ngueveu, C. Prins, and R. Wolfler Calvo, "An effective memetic algorithm for the cumulative capacitated vehicle routing problem," *Computers and Operations Research*, vol. 37, no. 11, pp. 1877–1885, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2009.06.014

[71] T. A. Nguyen, M. Do, A. E. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati, "Generating diverse plans to handle unknown and partially known user preferences," *Artificial Intelligence*, vol. 190, pp. 1–31, Oct. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370212000707

[72] E. Nikolova, M. Brand, and D. Karger, "Optimal Route Planning under Uncertainty." *ICAPS*, pp. 131–140, 2006. [Online]. Available: http://www.aaai.org/Papers/ICAPS/2006/ICAPS06-014.pdf

[73] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of operations research*, vol. 41, no. 4, pp. 421–451, 1993.

[74] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[75] A. G. Percus and O. C. Martin, "The stochastic traveling salesman problem: Finite size scaling and the cavity prediction," *Journal of Statistical Physics*, vol. 94, no. 5-6, pp. 739–758, 1999.

[76] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, Feb. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0377221712006388

[77] J. A. Potters, I. J. Curiel, and S. H. Tijs, "Traveling salesman games," *Mathematical Programming*, vol. 53, no. 1-3, pp. 199–211, 1992.

[78] J.-Y. Potvin, "State-of-the-art surveyâĂŤthe traveling salesman problem: A neural network perspective," *ORSA Journal on Computing*, vol. 5, no. 4, pp. 328–348, 1993.

[79] B. Roberts and D. P. Kroese, "Estimating the Number of s - t Paths in a Graph Counting by Estimation," vol. 11, no. 1, pp. 195–214, 2007.

[80] W. Saad, Z. Han, T. Başar, M. Debbah, and A. Hjørungnes, "Hedonic coalition formation for distributed task allocation among wireless agents," *Mobile Computing, IEEE Transactions on*, vol. 10, no. 9, pp. 1327–1344, 2011.

[81] C. Sabo, D. Kingston, and K. Cohen, "A formulation and heuristic approach to task allocation and routing of uavs under limited communication," *Unmanned Systems*, vol. 2, no. 01, pp. 1–17, 2014.

[82] S. Sahni and T. Gonzales, "P-complete problems and approximate solutions," in *Switching and Automata Theory, 1974., IEEE Conference Record of 15th Annual Symposium on*. IEEE, 1974, pp. 28–32.

[83] M. W. Savelsbergh, "Local search in routing problems with time windows," *Annals of Operations research*, vol. 4, no. 1, pp. 285–305, 1985.

[84] P. Scerri, B. Kannan, P. Velagapudi, K. Macarthur, P. Stone, M. Taylor, J. Dolan, A. Farinelli, A. Chapman, B. Dias *et al.*, "Flood disaster mitigation: A real-world challenge problem for multi-agent unmanned surface vehicles," in *Advanced Agent Technology*. Springer, 2011, pp. 252–269.

[85] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek, "A framework for clustering uncertain data," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1976–1979, 2015.

[86] L. V. Snyder and M. S. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 174, no. 1, pp. 38–53, 2006.

[87] B. Stout, "Smart moves: Intelligent pathfinding," *Game developer magazine*, vol. 10, pp. 28–35, 1996.

[88] B. Suo, Y.-s. Cheng, C. Zeng, and J. Li, "Calculation of Failure Probability of Series and Parallel Systems for Imprecise Probability," *International Journal of Engineering and Manufacturing*, vol. 2, no. 2, pp. 79–85, 2012. [Online]. Available: http://www.mecs-press.org/ijem/ijem-v2-n2/v2n2-12.html

[89] L. Talarico, K. Sörensen, and J. Springael, "The risk constrained cash-in-transit vehicle routing problem with time windows," vol. 32, no. 0, 2013.

[90] H. Tang and E. Miller-Hooks, "A tabu search heuristic for the team orienteering problem," *Computers & Operations Research*, vol. 32, no. 6, pp. 1379–1407, 2005.

[91] L. Tang and X. Wang, "Iterated local search algorithm based on very large-scale neighborhood for prize-collecting vehicle routing problem," *The International Journal of Advanced Manufacturing Technology*, vol. 29, no. 11-12, pp. 1246–1258, 2006.

[92] K. P. Valavanis, *Advances in unmanned aerial vehicles: state of the art and the road to autonomy*. Springer Science & Business Media, 2008, vol. 33.

[93] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[94] S. Voß, "Dynamic tabu search strategies for the traveling purchaser problem," *Annals of Operations Research*, vol. 63, no. 2, pp. 253–275, 1996.

[95] K.-H. C. Wang and A. Botea, "Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees," *Journal of Artificial Intelligence Research*, vol. 42, pp. 55–90, 2011.

[96] K. A. I. Zhang, A. Florida, and E. G. Collins, "Centralized and Distributed Task Allocation in Multi-Robot Teams via a Stochastic Clustering Auction," vol. 2, no. 3, 2010.

[97] E. Zitzler, "Evolutionary Algorithms for Multiobjective Optimization : Methods and Applications," no. 30, Doctoral thesis ETH NO. 13398, Zurich: Swiss Federal Institute of Technology (ETH), Aachen, Germany: Shaker Verlag. 1999.

Jeffrey Hudack

Research Interests

Multi-agent task allocation, risk-aware planning, game theory, machine learning, information fusion

Education

Syracuse University, Syracuse, NY

Ph.D., Computer & Information Science & Engineering, *Expected:* December 2016

- Thesis Topic: *Risk-aware Planning for Sensor Data Collection*
- Advisor: Jae C. Oh, Ph.D

M.S., Computer Science, May 2010

State University of New York at Utica-Rome, Marcy, NY

B.S., Computer Science , May 2006

Research Experience

Computer Scientist — January 2010 to present
Information Directorate,
Air Force Research Laboratory
Rome, NY

Associate Computer Scientist — May 2006 to January 2010
Information Directorate,
Air Force Research Laboratory
Rome, NY

Mathematician's Aide — June 2005 to May 2006
Information Directorate,
Air Force Research Laboratory
Rome, NY

Refereed Conference Publications

1. **Hudack, J.**, Oh, J.C. "Multi-agent sensor data collection with attrition risk" *Twenty-Sixth International Conference on Automated Planning and Scheduling.* 2016.

2. **Hudack, J.**, Gemelli, N., Brown, D.S., Loscalzo, S., Oh, J.C. "Multiobjective optimization for the stochastic physical search problem", In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (pp. 212-221). Springer International Publishing, 2015.

3. Brown, D.S., **Hudack, J.**, Banerjee, B. "Algorithms for stochastic physical search on general graphs." *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

4. Gemelli, N., **Hudack, J.**, Oh, J.C., "Virtual Structure Reduction on Distributed K-Coloring Problems" In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02*, pp. 46-52. IEEE Computer Society, 2013.

5. Gemelli, N., **Hudack, J.**, Oh, J.C., "Virtual Structure Reduction for Distributed Constraint Problem Solving." *AAAI (Late-Breaking Developments)*, 2013.

6. **Hudack, J.**, Gemelli, N., Scalzo, M. "Local Utility Estimation in Model-Free, Multi-Agent Environments" No. AFRL-RI-RS-TP-2011-42. AIR FORCE RESEARCH LAB ROME NY INFORMATION DIRECTORATE, 2010.

7. Wright, R., **Hudack, J.**, Gemelli, N., Loscalzo, S., Lue, T.K. "Agents technology research" No. AFRL-RI-RS-TR-2010-057. AIR FORCE RESEARCH LAB ROME NY INFORMATION DIRECTORATE, 2010.

8. Staskevich, G.R., **Hudack, J.**, Lawton, J., Carozzoni, J.A. "Semantic Interoperability in Distributed Planning." AIR FORCE RESEARCH LAB ROME NY INFORMATION DIRECTORATE, 2008.

9. Carozzoni, J., Lawton, J., DeStefano, C., Ford, A., **Hudack,. J.**, "Distributed Episodic Exploratory Planning (DEEP)" No. AFRL-RI-RS-TR-2008-279. AIR FORCE RESEARCH LAB ROME NY INFORMATION DIRECTORATE, 2008.

Refereed Journal Publications

1. Brown, D.S., **Hudack, J.**, Gemelli, N., Banerjee, B., "Exact and Heuristic Algorithms for RiskAware Stochastic Physical Search" *Computational Intelligence* (2016)

Presentations

Air Force Presentations

- Vehicle Swarms for Intelligence, Surveillance, and Reconnaissance (ISR), Rome, NY Mar 2016
- Risk-Aware Planning for ISR, Rome, NY Aug 2015
- ISR Swarms, AFRL Autonomy Workshop, Dayton, OH Aug 2014
- Detecting Multi-Agent Emergent Behavior using Evolutionary Game Theory, Rome, NY Nov 2013