NORTH ATLANTIC TREATY ORGANIZATION

SCIENCE AND TECHNOLOGY ORGANIZATION

AC/323(SAS-107)TP/865

www.sto.nato.int

STO TECHNICAL REPORT **TR-SAS-107**

Factoring Communications and Situational Awareness in Operational Models of Dismounted Combat

(Intégration des communications et de la connaissance de la situation dans les modèles opérationnels de combat débarqué)

Final Report of SAS-107.

Published March 2020

Distribution and Availability on Back Cover

NORTH ATLANTIC TREATY ORGANIZATION

SCIENCE AND TECHNOLOGY ORGANIZATION

AC/323(SAS-107)TP/865

www.sto.nato.int

STO TECHNICAL REPORT **TR-SAS-107**

Factoring Communications and Situational Awareness in Operational Models of Dismounted Combat

(Intégration des communications et de la connaissance de la situation dans les modèles opérationnels de combat débarqué)

Final Report of SAS-107.

The NATO Science and Technology Organization

Science & Technology (S&T) in the NATO context is defined as the selective and rigorous generation and application of state-of-the-art, validated knowledge for defence and security purposes. S&T activities embrace scientific research, technology development, transition, application and field-testing, experimentation and a range of related scientific activities that include systems engineering, operational research and analysis, synthesis, integration and validation of knowledge derived through the scientific method.

In NATO, S&T is addressed using different business models, namely a collaborative business model where NATO provides a forum where NATO Nations and partner Nations elect to use their national resources to define, conduct and promote cooperative research and information exchange, and secondly an in-house delivery business model where S&T activities are conducted in a NATO dedicated executive body, having its own personnel, capabilities and infrastructure.

The mission of the NATO Science & Technology Organization (STO) is to help position the Nations' and NATO's S&T investments as a strategic enabler of the knowledge and technology advantage for the defence and security posture of NATO Nations and partner Nations, by conducting and promoting S&T activities that augment and leverage the capabilities and programmes of the Alliance, of the NATO Nations and the partner Nations, in support of NATO's objectives, and contributing to NATO's ability to enable and influence security and defence related capability development and threat mitigation in NATO Nations and partner Nations, in accordance with NATO policies.

The total spectrum of this collaborative effort is addressed by six Technical Panels who manage a wide range of scientific research activities, a Group specialising in modelling and simulation, plus a Committee dedicated to supporting the information management needs of the organization.

- AVT Applied Vehicle Technology Panel
- HFM Human Factors and Medicine Panel
- IST Information Systems Technology Panel
- NMSG NATO Modelling and Simulation Group
- SAS System Analysis and Studies Panel
- SCI Systems Concepts and Integration Panel
- SET Sensors and Electronics Technology Panel

These Panels and Group are the power-house of the collaborative model and are made up of national representatives as well as recognised world-class scientists, engineers and information specialists. In addition to providing critical technical oversight, they also provide a communication link to military users and other NATO bodies.

The scientific and technological work is carried out by Technical Teams, created under one or more of these eight bodies, for specific research activities which have a defined duration. These research activities can take a variety of forms, including Task Groups, Workshops, Symposia, Specialists' Meetings, Lecture Series and Technical Courses.

The content of this publication has been reproduced directly from material supplied by STO or the authors.

Published March 2020

ISBN 978-92-837-2189-5

Table of Contents

List of Figures and Tables

List of Acronyms

AAPL	Approximate POMDP Planning
BBN	Bayesian Belief Network
BN	Bayesian Network
C4I	Command, Control, Communications, Computers, and Intelligence
CTMC	Continuous Time Markov Chain
DTMC	Discrete Time Markov Chain
MC	Markov Chain
MDP	Markov Decision Process
MRP	Markov Reward Process
OM	Observation Model
POMDP	Partially Observable Markov Decision Process
PPE	Personal Protective Equipment
RTG	Research Task Group
SA	Situational Awareness
SARSOP	Successive Approximations of the Reachable Space under Optimal Policies
SLE	Stochastic Lanchester Equation
STO	Science and Technology Organization
TTP	Techniques, Tactics and Procedures
UAV	Unmanned Aerial Vehicle

Acknowledgements

Thanks to Mr. David Shaw (CAN) for initially suggesting the POMDP approach, and for his support along the way.

SAS-107 Membership List

CO-CHAIRS

Mr. Roy BENDA
Netherlands Organisation for Applied Scientific Research (TNO)
NETHERLANDS
Email: roy.benda@tno.nl

Dr. Jérôme LEVESQUE
DRDC-CORA
CANADA
Email: jerome.levesque@forces.gc.ca

MEMBERS

Dr. Andrew COUTTS
Defence Science and Technology Group (DSTG)
AUSTRALIA
Email: andrew.coutts@dst.defence.gov.au

Mr. Christopher NICHOLLS
Dstl
UNITED KINGDOM
Email: cjnicholls@dstl.gov.uk

Mr. K. Tara SMITH
HFE Solutions Ltd.
UNITED KINGDOM
Email: tara@hfesolutions.co.uk

Dr. Nicholas STANBRIDGE
Dstl
UNITED KINGDOM
Email: NHSTANBRIDGE@mail.dstl.gov.uk

Dr. Maurice VANBEURDEN
Netherlands Organisation for Applied Scientific Research (TNO)
NETHERLANDS
Email: maurice.vanbeurden@tno.nl

Factoring Communications and Situational Awareness in Operational Models of Dismounted Combat

(STO-TR-SAS-107)

Executive Summary

Introduction

Defence funds dedicated to dismounted soldier systems are finite, and must be divided among multiple components. Deciding on the right mix can be difficult – some technologies improve lethality and protection, others improve SA. At the same time, these technologies might increase cognitive and physical load. In this report we present a way to perform comparisons across this apparent divide, and find the optimal mix of technologies. We present a mathematical combat model that considers the joint effects of situational awareness, lethality, and protection equipment in terms of expected lives saved. The model can therefore be used to design an optimal dismounted soldier system, one that will save the most lives.

Model

Our approach relies on representing the decision maker as an optimal one, at all times. That decision maker however must make decisions under uncertainty, and time constraints. As cognitive burden increases, several changes can occur in the model: the time between decisions might increase, the amount of information considered in each decision can decrease, or the planning horizon might be shortened, resulting in more myopic decisions. Each of these levers in the model gives the flexibility to represent a degradation of decision-making, and SA, while still assuming that the commander is making the best decision possible, but under difficult constraints. Technically, our model is based on two pillars. First, combat is modelled as a Continuous-Time Markov Chain (CTMC). Second, the commander is modelled as a decision-maker in a Partially-Observed Markov Decision Process (POMDP). POMDPs are sequential decision problems that are solved by dynamic programming. They are difficult to solve because, contrary to fully observable Markov Decision Process (MDPs), some of the state variables are hidden. Fortunately, advanced computational methods have been developed to solve them.

Results

We implemented a proof of concept, based on a dismounted combat scenario in which a section of 12 soldiers must secure the entrance of a tunnel. At any time the commander can alter the route, or abort the operation, based on the information available at that time. In the scenario we also include an area sensor, which could be an Unmanned Aerial Vehicle (UAV), for example. We show how to find the optimal trade-off between increasing the soldiers' sensing capacity, and increasing the capacity of the UAV. We also show how to find the optimal trade-off between increasing the soldiers' sensing capacity, and increasing their lethality and personal protection equipment. Finally we demonstrate how, by increasing the time interval between decisions in the model, we can simulate an increase in cognitive burden, which increases the expected lives at risk.

Conclusion

Our model has exploitation potential in the sectors of procurement, capability development, defence S&T, and academia. This wide-ranging potential is a tribute to the flexibility of POMDPs, which can be made as abstract, or detailed, as wanted. We suggest several avenues for expanding our implementation of the model: integrating Bayesian Belief Networks (either in the CTMC and/or the observation model), combining a sequence of scenario stages, and exploring other ways to represent cognitive burden.

Intégration des communications et de la connaissance de la situation dans les modèles opérationnels de combat débarqué

(STO-TR-SAS-107)

Synthèse

Introduction

Les budgets de la défense consacrés aux systèmes de combattants débarqués sont limités et doivent être partagés entre plusieurs composants. L'élaboration d'une combinaison optimale demeure difficile : certaines technologies améliorent la létalité et la protection, d'autres améliorent la connaissance de l'environnement. Dans le même temps, ces technologies peuvent entraîner une augmentation de la charge cognitive et physique. Dans ce rapport, nous présentons une méthode pour effectuer des comparaisons portant sur ces deux volets en apparence séparés, et obtenir la combinaison optimale de technologies. Nous présentons un modèle de combat mathématique qui prend en compte les effets conjoints de la connaissance de la situation d'une part, et de la létalité et de l'équipement de protection d'autre part, en termes de vies épargnées attendues. Le modèle peut donc être utilisé pour concevoir un système de combattants débarqués optimal, capable d'épargner le plus grand nombre de vies.

Modèle

Notre approche consiste à représenter le décideur comme étant le meilleur possible, à tout moment. Ce décideur doit toutefois prendre des décisions dans des conditions d'incertitude et est soumis à des contraintes de temps. A mesure que la charge cognitive augmente, plusieurs modifications peuvent se produire dans le modèle : l'intervalle entre les décisions peut s'accroître, la quantité d'informations prise en compte dans chaque décision peut diminuer, ou l'horizon de planification peut être raccourci, ce qui entraîne davantage de décisions myopes. Chacun de ces leviers du modèle offre la possibilité de représenter une dégradation de la prise de décision, tout en supposant que le commandant prend la meilleure décision possible, mais sous des contraintes sévères. Techniquement, notre modèle repose sur deux piliers. Le premier, le combat est modélisé comme une chaîne de Markov à temps continu (CTMC). Le second, le commandant est modélisé en tant que décideur dans un processus de décision markovien partiellement observé (POMDP). Les POMDP sont des problèmes séquentiels appelant une décision qui sont résolus par une programmation dynamique. Ils sont difficiles à résoudre car, contrairement au processus de décision markovien (MDP) parfaitement observable, certaines des variables d'état sont masquées. Toutefois, des méthodes de calcul avancées ont été développées pour les résoudre.

Résultats

Nous avons mis en place une validation de principe, basée sur un scénario de combat à pied dans lequel une section de 12 soldats doit sécuriser l'entrée d'un tunnel. A tout moment, le commandant peut modifier l'itinéraire ou interrompre l'opération en fonction des informations disponibles à ce moment-là. Dans le scénario, nous incluons également un capteur de zone, tel un véhicule aérien sans pilote (UAV), par exemple. Nous montrons comment trouver le compromis optimal entre l'augmentation de la capacité de détection des combattants et celle de l'UAV. Nous montrons également comment trouver le compromis

optimal entre l'augmentation de la capacité de détection des combattants, et l'élévation de leur létalité et le renforcement de leurs équipements de protection individuelle. Enfin, nous montrons comment, en augmentant l'intervalle de temps entre les décisions dans le modèle, nous pouvons simuler une augmentation du fardeau cognitif, laquelle augmente le nombre attendu de vies en danger.

Conclusion

Notre modèle présente un potentiel d'exploitation dans les secteurs de l'approvisionnement, du développement des capacités, des sciences et technologies de la défense et des universités. Ce vaste potentiel rend hommage à la flexibilité des POMDP, qui peuvent être abrégés ou détaillés à volonté. Nous proposons plusieurs pistes pour élargir la mise en œuvre du modèle : intégrer les réseaux de croyances bayésiennes (dans le modèle CTMC et/ou le modèle d'observation), combiner une séquence de scénarios et explorer d'autres moyens de représenter le fardeau cognitif.

Chapter 1 – INTRODUCTION

When defence departments invest in dismounted soldier systems, they face the difficult problem of finding a combination of individual equipment that maximizes a team's combat effectiveness. They must consider kinetic aspects of combat, such as lethality, protection, or mobility, but also informational aspects such as communications and Situational Awareness (SA). An optimal solution must conciliate these seemingly disparate objectives. While weapons and Personal Protective Equipment (PPE) are well represented in existing combat models, the representation of Command, Control, Communications, Computers, and Intelligence (C4I) technology is less mature. With the increased range and importance of these devices, their proper representation in combat models is indispensable.

We present a combat model that combines the effectiveness of weapons and the quality of information in a common objective function: how they reduce loss of life on the friendly side. Our proof of concept shows how to formulate the time-dependent attrition rates, the commander's observations, and her decision making under stress and uncertainty.

1.1 OBJECTIVES

The NATO SAS-107 Research Task Group (RTG) had the following objectives [17]:

1) Determine the key operational factors describing the effects of changing situational awareness in dismounted operations e.g. due to the enhancement in soldier technology:
   a) Link operational factors to human performance parameters related to situational awareness; and
   b) Describe the possible impacts of the new technologies in improving situational awareness of the dismounted combatant.
2) Define a methodology for better integrating these operational factors in operational analysis, modelling and simulation of dismounted combatant operations:
   a) Determine how different combat regimes affect the requirement for enhanced situational awareness; and
   b) Identify output parameters and algorithms which quantify the timing and quality of decisions resulting from variations in situational awareness.
3) As a proof-of-concept of the new methodology, generate a pilot study demonstrating the change in operational effectiveness due to variation in situational awareness of the dismounted combatant.

To address Objective 1 we reviewed the concepts of SA and cognitive workload. This review is presented in Chapter 2, with additional details in Annex A. For Objective 2, we identified Partially Observable Markov Decision Process (POMDPs) as the best way to integrate the SA idea in a combat model. This method is presented in Chapter 3, with a toy application (defence of a combat outpost) developed in detail in Annex C. Finally, we addressed Objective 3 by building a complete application based on a dismounted operation. This application is described in Chapter 4, with additional details on the scenario in Annex B.

Our application leverages the excellent Approximate POMDP Planning (AAPL) Toolkit created by the groups of Lee Wee Sun and David Hsu at the Advanced Robotics Center (National University of Singapore)[1]. This toolkit

[1] AAPL webpage: http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/; GitHub repository: https://github.com/AdaCompNUS/sarsop.

is based on the SARSOP algorithm developed there by Hanna Kurniawati et al. [14]. The AAPL Toolkit provides a very performant, yet generic POMDP solver. The details of each specific problem must be detailed by the user in a separate input file. Creating these input files can be a non-trivial task if the problem is relatively large. For our application, each input corresponds to a text file of several megabytes that depends on extensive calculations. For this we developed a set of Python programs that are found in Annex D, along with a Python script to do batch runs and the R script we used to generate the figures found in Chapter 4.

1.2 BACKGROUND

Since Endsley's seminal papers on SA [7], [8] there have been multiple studies and experiments on the role of SA in military operations, including past activities under the NATO Science and Technology Organization (STO) [1]-[4]. While our objectives included a characterization of SA, our aim was to understand the state of the art so that we could properly integrate this concept in the development of a mathematical model of combat.

Endsley bases the concept of SA on the dynamic nature decision making, and the decision maker's ability to anticipate future states of the world. This task – making optimal sequences of decisions in a familiar but uncertain world – is also behind the idea of *dynamic programming*, the "mathematical theory of multistage decision processes" [5]. Nowadays "programming" is associated with coding instructions for computers. In the 1940's however, when dynamic programming was coined, "[t]he word programming was used by the military to mean scheduling" [15], and dynamic programming was developed as an efficient method for optimizing sequences of decisions, which could be applied to decision-making under uncertainty. The method quickly outgrew its first military applications, and got extended to many domains of science, engineering, operational research, and economics. Dixit and Pindyck suggest applications in social science, to questions of marriage and suicide, and in law, to study legal reform and constitutions [6].

Chapter 2 – SITUATIONAL AWARENESS IN DISMOUNTED COMBAT

In an ideal world, a soldier's observation of reality would be perfect; his information would be complete and certain which, of course, is not the case in the real world. Within a real world battlespace, a soldier must make discrete decisions, based on incomplete and uncertain information about the state of the battlespace. This situation is further complicated by the moderating effects of the various task, environmental and individual factors, mentioned in Figure 2-1. Because of this uncertainty – the "fog of war" in other words – it is impossible to know the exact state of the battlespace.

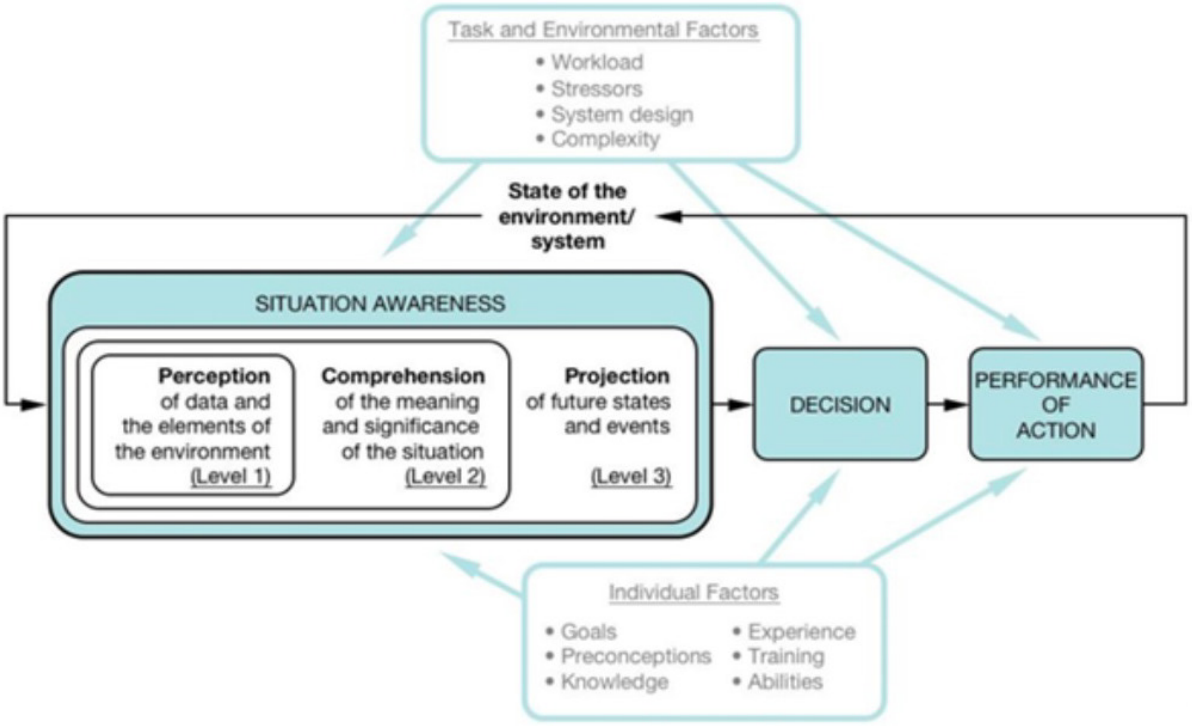


Figure 2-1: Conceptual Overview of Manifestation of SA During Combat.

An important objective of this study is to determine how SA manifests itself in dismounted combat. The output of this effort serves as a theoretical/conceptual base for the method we propose in the next section. The definition we adopted for SA closely follows Endsley[1]:

> ***Situational Awareness:*** *The level of individual and shared perception of elements in the battlespace of interest within specific intervals in time and space, the comprehension of their meaning and the projection of their status in the near future in order to make appropriately informed and timely decisions that facilitate the accomplishment of the mission.*

[1] Endsley's definition [8]: "Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future."

As one can derive from the definition, the three core constructs in understanding SA are perception, comprehension and projection. Meaning that a soldier must be able to perceive the observed data and elements within its environment, comprehend the meaning and significance of the situation the soldier is in and subsequently project possible future states of the situation (including the assessment of the likelihood of these possible future states) in order to make a decision and act accordingly. The action's effect alters the state of the environment which, on its turn, is observed by the soldier (and the cycle restarts). This iterative process continues until the goal (or objective) is reached or the soldier stops functioning (e.g. the soldier is ordered to stop or is not able to continue). Note that this process is moderated by various task and environmental factors (e.g. workload, task complexity, stress), as well as individual factors (e.g. preconceptions, training, experience). The previous rationale is captured in the conceptual overview, depicted in Figure 2-1.

A model for representing SA must include a functionality which is able to produce and subsequently feed the decision model a 'noisy' rendition of reality, with due regard for the various task, environmental and individual factors. In response, the workgroup constructed a strawman model of a Bayesian Belief Network (BBN) which could fulfill this requirement (see Figure 2-2). This BBN is largely based on the conceptual overview as depicted in Figure 2-1.

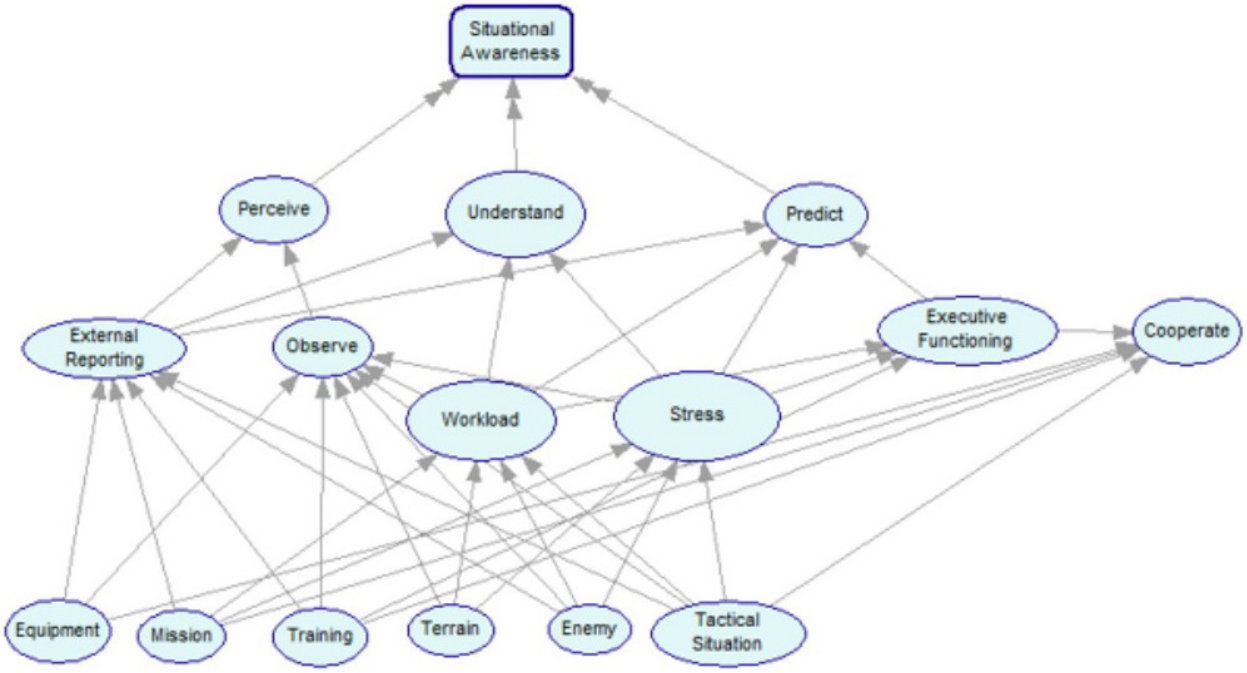


Figure 2-2: Strawman Model of a BBN for SA in Combat.

2.1 INTERACTION BETWEEN DECISION MAKING AND COGNITION

The impact of task, workload and other factors on SA is driven through the likelihood (likelihood of the success of the observation) and work rate (any resultant time delay due to the workload affecting the propagation from perception to comprehension to projection) of the sub-states of an observation: i.e. the SA parameters. In other words we are equating the results of an observation to SA. An expanded discussion of this can be seen in Annex A.

There are three main elements that contribute to the cognition process:

1) The task profile;
2) The workload profile; and
3) External factors that interact with workload.

There is a complex and iterative relationship between the decision model and the observation model. The parameters that drive the task at hand and the experienced workload can be derived from the decision model; these parameters are then propagated through the observation model to give the likelihood of an observation happening. The elements of the cognitive process are expanded and presented below.

2.1.1 Task Profile

The task profile describes the characteristics of the task at hand, and will determine the external demands that a soldiers experiences. There are three components to the task profile which are more extensively described in Ref. [9]:

- Level of information processing: these can be categorized by the Rasmussen task categories of information processing:
  - Skill;
  - Rule (TTP); and
  - Knowledge.
- Time occupied: i.e. the percentage of time that an individual can apply to the observation task:
  - Time.
- Task Switching: the process of moving between one context or task to another:
  - Number of tasks (would impact on work rate);
  - Number of contexts of task (would impact on work rate); and
  - Separation of information (would impact on likelihood, work rate).

2.1.2 Workload Profile

The workload profile could be presented at different levels of detail. As we are working in a limited domain (close combat) we could reduce the level of detail needed to encapsulate the workload to a number of pre-calculated levels, based on our understanding of the close combat environment.

Table 2-1: Two Approaches to Representing Workload Profiles for Close Combat.

Option 1 Full Profile	Option 2 Simple Profile
• Input Modality – Visual – Auditory – Haptic – Data Processing Stages – Perception – Processing – Action • Reasoning Modality – Subconscious – Symbolic – Linguistic	• Visual based activity – Level 1 – Level 2 – Level 3 • Auditory based activity – Level 1 – Level 2 – Level 3

2.1.3 External Factors that Interact with Workload

In a group model building session we identified factors from three scientific domains – cognitive, psychosocial and physical – that impact cognitive workload [10]. Examples of factors that that interact with cognitive workload:

- Stress / Perceived risk (would impact on likelihood, work rate);
- Fatigue (would impact on work rate);
- Physical workload (would impact on likelihood, work rate);
- Self efficacy (would impact on work rate);
- Working memory (would impact on likelihood, work rate); and
- Abilities/training/experience (would impact on likelihood, work rate).

Not in the model, but factors that also impact workload are:

- Expectations (construct of decision making method "baseline belief");
- Goals and objectives (linked to expectations in decision making method);
- Personality type / IQ (would impact on work rate); and
- Usability (would impact on likelihood, work rate).

Note: not all of these factors have an equal impact, either to each other or in different situations.

There are two different ways that these factors could be represented within the model. The simple one would be to allow them to apply a modifying factor to the overall likelihood of success of observation, however they could

also be represented as modifying the fundamental workload and task parameters directly. This second method would be a much more realistic way of applying them, but would lead to a much more complex and sophisticated model.

2.2 MEASURING SITUATIONAL AWARENESS

The workgroup's review of current methods and models resulted in the following two general findings:

1) One of the most difficult aspects about methods for assessing SA revolves around measuring SA without impacting it. Most available methods do not meet this requirement or postpone assessing the amount of SA until after an experiment (or real-life case for that matter). The latter assessment approach is fraught with hindsight bias and other unwanted memory related influences and therefore not recommended; and
2) Current methods for assessing SA usually require a lab or field experiment type of setting in order to collect data and are often heavily reliant on human test subjects. This type of experiments can easily become expensive and time consuming to set up.

Hence, a cost and time efficient method for assessing SA, without impacting it, is not available.

Chapter 3 – METHOD

In this section we summarize the model, and the theory behind it. It is merely an overview: POMDPs are at the forefront of current research in many fields. Their theory is underpinned by decades of developments in stochastic modelling and dynamic programming. References such as [13], and in particular [14], have in-depth treatment of the formalism. For an overview of how POMDPs relate to other, simpler decision models see the example developed in Annex C.

This section interprets how we use the basic components of POMDPs for the specific task of modelling decision-making in dismounted combat.

3.1 INFINITE HORIZON POMDPS

A POMDP can be formulated over a finite horizon, with a pre-determined ending, or an infinite one where costs are discounted every period by a factor $\rho \in [0, 1)$, so that the cost function converges. In the proof of concept that follows (Chapter 4) we formulate the problem as an infinite horizon one. Formally, such a problem can be specified by these seven components[1]. We provide them here in the same order as in Ref. [14], each with its description in the context of our application:

1) **State Space –** The set of all possible combat states: three integers representing the number of friendly combatants remaining and the number of enemy combatants on each of two approach routes.
2) **Action Space –** A set of tactics available to the commander throughout the mission, three in our proof of concept: move to (or stay on) the left route, move to (or stay on) the right route, or stop the mission.
3) **Observation Space –** The commander's observations. We formulated them in the same space as the combat states: the commander knows her team's status perfectly at any time, but only has a partial observation of the enemy's status on each route. Her assessment of the enemy's status is probabilistic, and depends on observations made by dismounted soldiers, and the UAV feed.
4) **Transition Probability Matrix –** The transition probabilities between combat states. These are the same as the attrition rates modelled by Lanchester's equations. They are dependent on the commander's actions: by choosing a different route, the commander is attempting to minimize the attrition rate on her side, and maximize the probability of mission success.
5) **Conditional Observation Probabilities –** The probability distribution over observations, *conditional* on the state and the commander's action. Even if the commander might never fully observe the current state, her experience and training provides a mental model of how likely each possible observation would be *if* the system was in a given state. She makes decisions based on that model, and the information she receives in sequence. This is where Bayes' rule plays a role: in solving the POMDP we use the observation sequence to calculate the probability of being in each state, *conditional* on the sequence of observations, and assuming optimal decisions by the commander. In effect, we model the commander's decision making. Note that *optimal* decisions here do not mean *perfect*: as the commander's sensing degrades – due to cognitive overload for example – she is aware that bad events become more likely and adjusts her decision making accordingly, acting more conservatively and potentially aborting the mission.

[1] Finite horizon problems mostly differ in that the transition probabilities, the conditional observation probabilities, and the cost function can be defined independently for each time step. For that reason there is no need for a discount rate. However the final costs have to be specified, so the finite horizon case also has seven components.

6) **Immediate Costs –** These incurred at each time step, depending on the previous state, the current state, the commander's current action and, possibly, the observation made (for this application we assumed that the cost to enable all observations were already sunk at the beginning of the scenario). The costs were all expressed in terms of loss of life.

7) **Discount Rate –** Given as a number between zero and one (but strictly smaller than one). As the discount rate gets closer to zero, more importance is given to the consequences of immediate events at the expense of events in the long term.

The seven elements above specify a complete POMDP. Solving the POMDP provides two essential pieces: the *optimal policy*, and the *expected cost* of applying that optimal policy.

The optimal policy is an instruction set: it provides the best action to take after each observation, so that the cost is minimized in the long run. In the infinite horizon case – the one we are interested in – the policy is independent of time. In typical applications of POMDPs, such as robot navigation, the optimal policy is an important output that is used directly by the robot. However human decision makers are not robots; in our application the optimal policy is not meant to be reused. It is merely a model of an optimal decision maker that we use for the purpose of simulation.

The most important output for our application is the expected cost – expressed as the number of lives at risk – when a simulated decision maker follows the optimal policy. Our objective is to find how fast this expected number of lives at risk decreases as we improve the performance of different sensors, weapons, and protection equipment. We can then determine, for fixed budget constraints, the combination that saves the most lives. We can also compare different models of how cognitive overload affects combat outcomes.

One complication is that the optimal policy and its expected cost are not unique: they are a function of the system's starting point, at $t = 0$. In the simpler, fully observed case (a Markov Decision Process (MDP) – for an example see the corresponding section in Annex C), there is a single optimal policy, and the expected cost depends on the starting state. However in the more general, partially observed case (POMDP), the initial state does not have to be known with certainty. In this case, the optimal policy and the expected cost will be a function, not of the initial state, but of a probability distribution over *all* initial states. Each of these probability distributions is called a *belief*, and the set of all initial beliefs is defined on a $(n – 1)$–simplex, n being the number of states.

Calculating an optimal policy and an expected cost function for the whole $(n – 1)$–simplex is expensive, and makes most problems impossible to compute in practice. However if one is willing to fix the initial belief, the computation required can be reduced by a large amount. This is the approach we used in our application (Chapter 4), using the Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) algorithm [15]. The initial belief becomes just another component of the scenario.

3.2 POMDP INPUT

Solving POMDPs efficiently requires complex algorithms. Fortunately, there are freely available implementations[2,3]. In this section we rather cover how to formulate the model itself. Details on the input requirements can be found at http://pomdp.org/code/pomdp-file-spec.html.

[2] http://www.pomdp.org/.

[3] http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/.

3.2.1 Transition Probabilities Between Combat States

Combat can be modelled mathematically as successive transitions through discrete states. In this application, the state variables include the number of combatants remaining on each side. The enemy side is further divided in two subgroups, one for each route. The state is therefore given by three numbers.

We calculate the transition probabilities between combat states using the methods developed for the Continuous Time Markov Chain (CTMC). In a CTMC the transition rate between any states is constant in time, and therefore follows a Poisson process. Descriptions of the CTMCs and the Poisson process can be found in standard textbooks [11], [12], [17]. In combat applications, CTMCs are also referred to as Stochastic Lanchester Equations (SLEs). Applications to combat modelling are treated in Refs. [19] and [20].

We model the rate of hit for a single shooter and target as the product of the rate of fire, ρ, and the probability of hitting the target at each shot:

$$r = \rho\, P\,(\text{hit}). \tag{1}$$

One important property of CTMCs is that the time to transition between any two states $\mathbf{s}_i$ and $\mathbf{s}_j$ is random and distributed exponentially, with a rate $\rho_{i\,j}$. Exponentially distributed transition times might seem restrictive and limit the application of CTMCs to scenarios in which individual shooters hit targets following a Poisson process. However this is not the case. Even if individual shooters are modelled using a different process (e.g. log-normal distribution of times between shots), in the limit of many shooters the time between target hits for the whole group will appear exponential (see Section 5.9 in Ref. [12]).

We represent each combat state as a list of integers $\mathbf{s}$ of dimension N_G, equal to the total number of subgroups (i.e., $\mathbf{s} \in \mathbb{N}^{NG}$). In this study we consider a case with $N_G = 3$: one friendly subgroup and two enemy subgroups, one for each available route. The combat state can only evolve in one direction: the number of combatants remaining in each subgroup must decrease or stay the same. An additional constraint is that the total number of combatants remaining must be at least one. The total number of states that we need to care about, N_s, is therefore equal to $N_s = \prod_{k=1}^{N_G} (s_0(k) + 1) - 1$ where $\mathbf{s}_0$ is the initial state.

Let us construct the N_s-by-N_s transition rate matrix Q = $\{q(i\ ,\ j)\}$ for our combat model. For this purpose we introduce the N_G-by-N_G matrices R = $\{r\ (k,\ l)\}$ and $\mathrm{A_i}$ = $\{a_i\ (k,\ l)\}$, the latter being a set of N_s matrices (one defined for each combat state).

Each element $r\ (k,\ l)$ of R corresponds to the hit rate of a shooter in subgroup k targeting a single member of subgroup l. These elements are calculated with Equation 1, using values for $\rho\ (k,\ l)$ and $P_{\text{hit}}\ (k,\ l)$ that can be specific to each pair of sub-groups.

Each element $a_i\ (k,\ l)$ of A_i represents the fraction of subgroup k's firepower that is directed at subgroup l, in combat state i. Each row k of $\mathrm{A_i}$ therefore represents the state-dependent firing policy for the corresponding subgroup. Depending on the combat scenario it would be possible to specify different rules for constructing A(i). Here we impose three simple constraints. First, since fratricide is not considered, the diagonal elements $a_i\,(k,\ k)$ will all be zero. Elements $a_i\,(k,\ l)$ will also be zero if subgroups k and l fight on the same side. Second, the non-zero elements in each row must sum up to one $\left(\sum_{l=1}^{N_G} a_i(k,l)\right.$. This corresponds to assuming that each

subgroup's firepower is proportional to the number of shooters remaining in that subgroup, and that its firepower can be divided efficiently between several target subgroups without any overhead. Third, all non-zero elements within a same row will be proportional to the number of combatants remaining in each opposing subgroup.

Finally let us define v, the index of the subgroup sustaining an incapacitation in transition $i \rightarrow j$.

With R, A and v defined, we are ready to construct Q:

$$q(i,j) = \begin{cases} \sum_{k=1}^{N_G} r(k,v) s_t(k) a_t(v,k) & \text{if } i \rightarrow j \text{ allowed} \\ -\sum_{j' \neq t} q(i,j') & \text{if } i = j \\ 0, & \text{otherwise} \quad . \end{cases} \tag{2}$$

Since the POMDP is formulated in discrete time, we translated the transition rates of the CTMC to transition probabilities for a discrete-time Markov Chain by propagating the CTMC over a chosen time increment Δt, representing the time interval between each decision made by the commander. The CTMC can be solved over that interval by computing $(\mathrm{I} + Q\delta\Delta t)^{\Delta t/\delta}\mathbf{s}_0$, where δ is small compared to Δt, I is the identity matrix and $\mathbf{s}_0$ is the initial combat state. In practice, this computation is achieved more efficiently using the method of uniformization, which is not covered here (see Refs. [17] or [11]).

3.2.2 Initial Belief – Modelling *Imprecise* and *Inaccurate* Intelligence

Setting the initial belief must be part of the scenario definition. It can reflect the quality of the intelligence at the onset, with a more dispersed initial belief corresponding to less precise intelligence. However there is an important distinction between the *precision* of intelligence, and its *accuracy*. In the case of inaccurate intelligence, we rather want to model the consequence making anti-optimal decisions because of biased assumptions. This special case must be modelled the following way: first solve the POMDP to obtain the optimal policy under the biased initial belief, and then calculate the expected cost of that policy under the unbiased initial belief. Finally, solve the POMDP a second time[4], under the unbiased initial belief. The difference between these two expected costs represents the cost of having biased intelligence. The same approach could be used to model the consequence of misinterpreting sensor output.

3.2.3 Actions

Through the mission, the commander has several actions available, and makes decisions. These actions have consequences for the transition probabilities, the conditional observation probabilities, and the immediate rewards. For example, for a case in which a commander has to choose between two approaches, the choice of route will affect both the transition probabilities (the attrition rate will change if the number of enemy combatants is different between the two approaches), and the observation probabilities (soldiers can only observe the number of enemy combatants on the route they travel). Switching between routes might also come at a cost, reflecting the additional risk it brings to the mission.

[4] It will not be necessary to solve more than once if the solution over the entire space of initial beliefs is available.

Chapter 4 – PROOF OF CONCEPT

We formulated the problem in a `.pomdp` format file using Python software written for this project that can be found in Annex D. We solved the POMDP using the implementation of the SARSOP algorithm [15], provided by its authors at https://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/.

4.1 SCENARIO

Generally speaking, military forces are deployed within a mission in order to achieve an effect. They possess a number of capabilities which they use to shape the environment, or gain information on it, to achieve their aim. Any element within the deployed force can choose to move or to engage (with e.g. sensors or kinetic effects) within their delegated authority to achieve their sub-aim. At each level within the ORBAT, from individual rifleman to company commander, entities use their situational awareness to determine whether they should:

- CONTINUE their current plan;
- AMEND the plan, by making changes which do not affect those not under their command;
- REPLAN, since the original plan is no longer tenable; and
- WAIT for further orders.

The opportunity to make changes to the plan will be limited by the individual's position within the ORBAT, according the range of capabilities that he has under command; for instance, a section second in command has rifles and grenade / grenade launchers at his disposal, and possibly a light machine gun, whereas a platoon commander has greater firepower, by virtue of numbers, and command of e.g. anti-armor/antistructure systems. The relationship between the independent variables (those items which you can change directly and are not influenced by other variables) and the effects and measures is shown in Figure 2-1. Battlefield effectiveness (within the scope defined in this study) is restricted to having physical effects on the battlefield (neutralising enemy / destroying infrastructure) or influence effects (deterring/suppressing the enemy) in order to achieve the declared mission. Success can be determined by examining the mission success criteria, which are informed by monitoring changes in the mission effectiveness criteria, listed in the 'Measures' column.

We illustrate the combat POMDP method by applying it to the scenario shown in Figure 4-1. This simple vignette was inspired by the more extensive scenario described in Annex B. Dismounted soldiers must move to a tunnel entrance to secure it. There could be a threat on the way, on either of two routes. The commander chooses the approach route based on observations made by her soldiers or a UAV. At any time during the mission she can alternate between routes or, if the risk becomes too high, abort the mission.

Switching between routes was free in the scenario but interrupting the mission came at a cost, to reflect the consequence of having to modify the strategy as a consequence of the mission's failure, hence putting more lives at risk in the future.

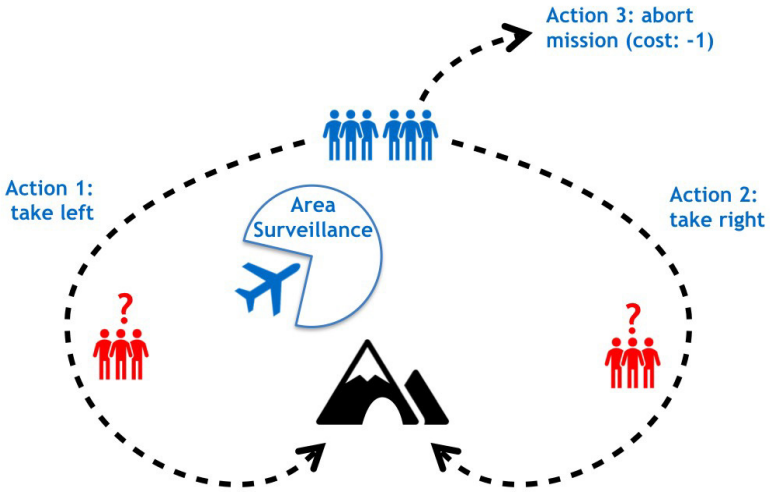


Figure 4-1: Scenario Used in the Pilot Study.

4.2 RESULTS

The combat POMDP makes it possible to look at the relative effectiveness of different sensor platforms, or to compare the trade-off between improving sensors versus personal protection or weapon lethality. The examples found in this section are based on the algorithms found in Annex D. It was not within the scope of SAS-107 to design observation models that matched existing sensor technologies; the examples that follow are therefore based on notional input parameters, given in Table 4-1. In all cases some of these parameters have been overridden, as detailed in the text and figures. For more details, refer to the source code in Annex D, starting with `tunnelRun.py`.

Table 4-1: Default Parameters Used in the Examples. They are the input to the `TunnelProblem` class constructor (see file `tunnelProblem.py` in Annex D).

Parameter	Value
`observationModelClass`	observationModels.DistinctTargetsOM
`nBlueMax`	12
`nRedMax`	6
`nRedMin`	1
`blueEffectiveness`	1.0
`redEffectiveness`	1.0
`deltaTime`	.1
`discountRate`	0.99
`blueCasualtyReward`	-1
`interruptReward`	-1
`probSingleRedDetect_bySoldier`	0.1
`probSingleRedDetect_byUav`	0.1

In future applications of combat POMDPs, we anticipate that most of the scientists' work will go into translating theoretical models of situational awareness, and decision making, into appropriate observation models and reward structures. Input parameters such as those listed in Table 4-1 will have to be based on empirical results, or at the very least estimates validated by subject matter experts. On the other hand estimating the transition rates between combat states should be more familiar to analysts, corresponding to factors from traditional models of combat, such as probability of hit, and probability of incapacitation. Some useful references for determining the transition rates are found in the combat modelling literature, such as Taylor's *Lanchester Models of Warfare* [19].

4.2.1 Sensor Portfolio Optimization

In Figure 4-2 we look at equivalent combinations of Unmanned Aerial Vehicle (UAV) sensor versus soldierborne sensors performance. The soldier-borne sensors could consist of night vision equipment, for example.

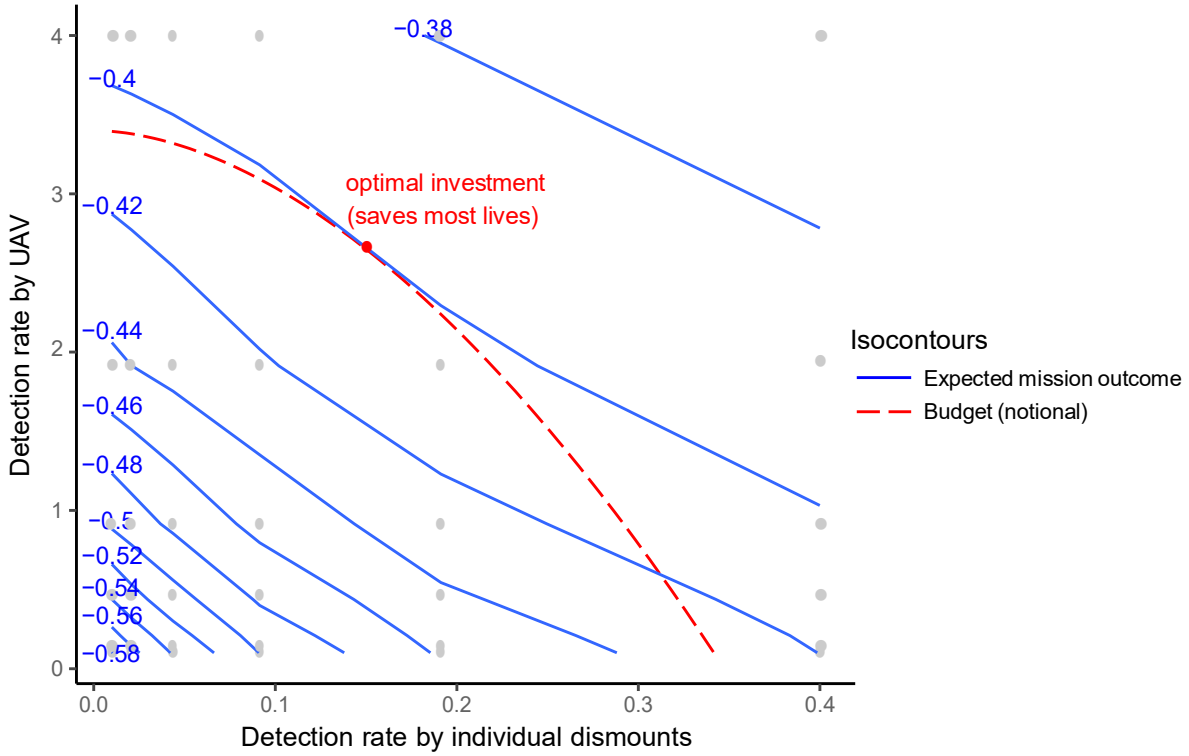


Figure 4-2: Finding the Most Effective Sensor Mix. Expected mission cost (blue contours) versus enemy detection rate by dismounts, and by UAV. The annotations in red show how results like these can be used to find the investment that saves the most lives.

4.2.2 Finding the Optimal Combination of Sensors, Weapons, and Protection Equipment

With the combat POMDP, the expected outcome of a mission is given unambiguously in terms of loss of life. This unique measure gives the power to look at the trade-off between rather different capabilities. Figure 4-3 shows a trade-off between soldier-borne sensors and a force multiplier obtained by increasing weapon lethality and the effectiveness of personal protection equipment.

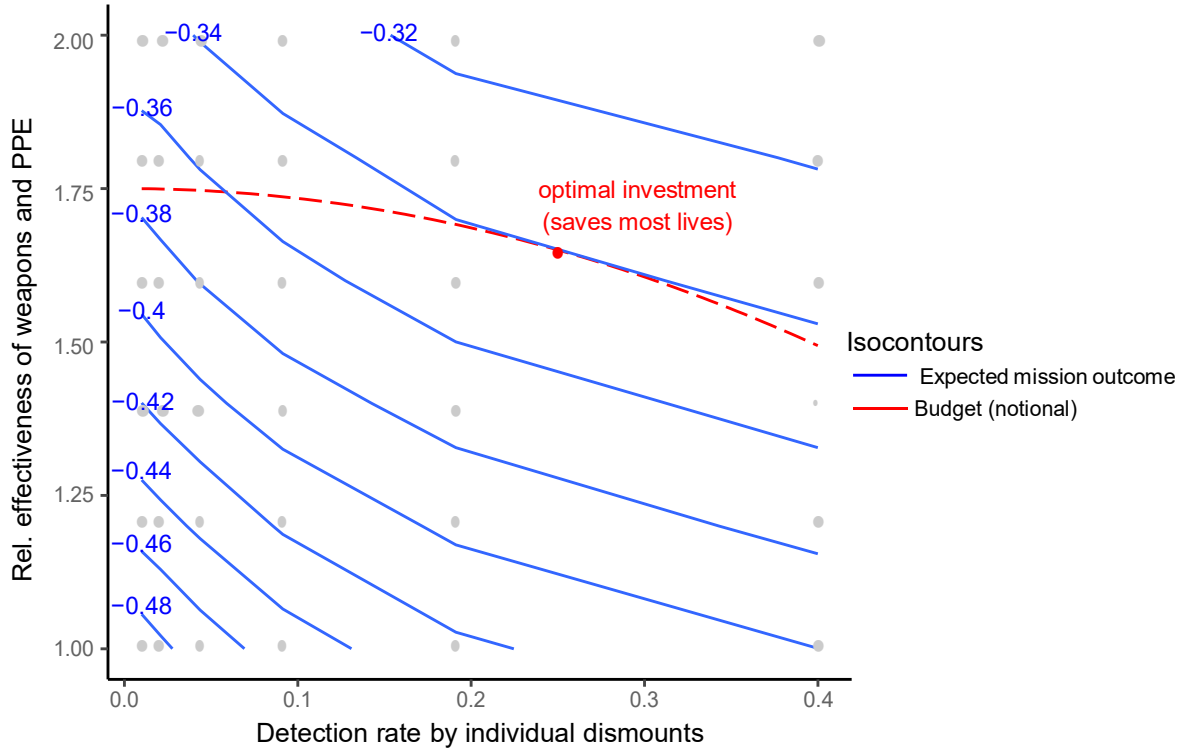


Figure 4-3: Finding the Best Mix of Sensors, Weapons and Protective Equipment. Expected loss of life (blue isocontours) dependent on enemy detection rate by dismounts, and a force multiplier corresponding to improvement in the performance of weapons and personal protection equipment for blue soldiers. The annotation in red shows a notional budget isocontour, with the red dot corresponding to what would be the optimal combination of performance parameters.

4.2.3 Modelling the Effect of Cognitive Burden

One of the parameters in the POMDP model is the time interval between decisions.[1] As decisions are further spread in time, the commander is exerting less frequent control, and must compensate by making more conservative decisions – possibly withdrawing forces earlier to avoid the possibility of heavy casualties. This is one way to look at the effect of cognitive burden, with decisions becoming increasingly arduous, and infrequent. As the commander's level of control decreases, we should expect a degradation in the mission outcome. This effect is shown in Figure 4-4.

[1] See parameter `deltaTime in tunnelProblem.py`, p. 56.

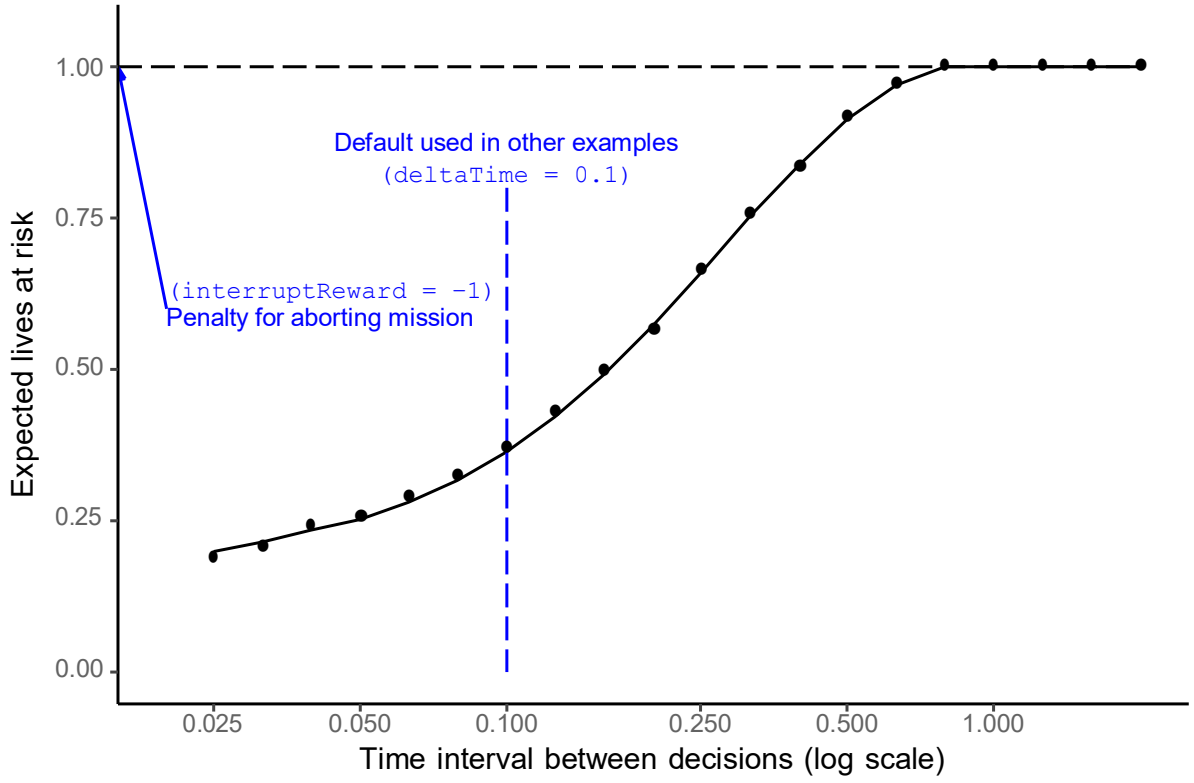


Figure 4-4: Role of Cognitive Overload. Expected lives at risk worsen when the delay between the commander's decisions increases.

The curve in Figure 4-4 has two essential features. First, the left part of the curve shows diminishing returns: reducing the interval between decisions becomes less and less beneficial. This happens because the time between decisions is becoming shorter than the enemy detection rate provided by the sensors. The information gain during that short period is small, and shortening the decision cycle does not help much. Second, the right part of the curve shows that as the time between decisions increases, the decision to withdraw at an early stage (or to forego the mission altogether) becomes optimal. Incurring a penalty of -1 becomes preferable to exposing the force to heavier losses.

This example is perhaps the simplest way to represent cognitive burden in the combat POMDP. It treats the commander's cognitive burden as constant through the mission, and does not take into account changes in cognitive burden depending on combat outcomes and decisions made.

A more realistic representation of cognitive burden can be achieved in several ways. The most obvious one would be to increase the number of actions available to the decision maker. When actions are added to the model (e.g. *call and manage medical evacuation*), it becomes possible to create a transition matrix, an observation model and a reward vector specific to that action. With this additional dimension, the modeller can therefore take into account an increase in the time between decisions. Another means would be to adjust the observation model, making it less informative as the state of combat becomes more stressful, and therefore degrading the expected combat outcome.

Cognitive burden might also result in the misinterpretation of sensor outputs. This phenomenon might be best simulated using the approach we suggest in Section 3.2.2 for modelling inaccurate intelligence.

In any case, representing cognitive burden in a combat POMDP must be supported by empirical data, or expert opinion.

Chapter 5 – CONCLUSION

Our test case shows how POMDPs can be used to explore trade-offs in investing between disparate technologies such as weapons and personal protection, and portable information technologies. It allows to derive an optimal portfolio of technologies for soldier systems. On one hand, the model is based on concepts such as dynamic programming and Markov chains, requiring a mathematical background that is typical of graduates in operational research. On the other hand, interpreting the model's output is accessible to decision makers from a broad range of backgrounds.

With regards to *Exploitation Potential*, we foresee four avenues for the extension and exploitation of the model:

- **Procurement:**
  - Quantify the value of portable C4I technology in combat;
  - Support the definition of user requirements; and
  - Support bid evaluations.
- **Capability Development:**
  - Explore the balance between kinetic and intelligence resources during the development of Concepts of Employment (CONEMP) and Concepts of Operation (CONOP); and
  - In the development of Techniques, Tactics and Procedures (TTP), characterize the potential of new portable C4I technology on SA (and ultimately on dismounted combat effectiveness).
- **Defence Science and Technology:**
  - Assess the effects of cognitive burden on dismounted combat effectiveness; and
  - Elicit new research directions.
- **Academia:**
  - As an additional approach to conducting fundamental research in the field of cognitive science and military operations research.

Chapter 6 – REFERENCES

[1] Uninhabited military vehicles (UMVS): Human factors issues in augmenting the force. Technical Report RDP RTO-TR-HFM-078, NATO STO, July 2007.

[2] Supervisory control of multiple uninhabited systems – methodologies and enabling human-robot interface technologies. Technical Report RDP RTO-TR-HFM-170, NATO STO, December 2012.

[3] Scientific support to NNAG above water warfare capability group (AWWCG). Technical Report RDP STO-TR-SCI-258, NATO STO, May 2016.

[4] System design considerations and technologies for safe high tempo operations in degraded environments. Technical Report RDP STO-TR-SCI-206, NATO STO, January 2016.

[5] R. Bellman. Dynamic Programming. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[6] A.K. Dixit and R.S. Pindyck. Investment under uncertainty. Princeton University Press, 1994.

[7] M.R. Endsley. Measurement of situation awareness in dynamic systems. Human factors, 37(1):65-84, 1995.

[8] M.R. Endsley. Toward a theory of situation awareness in dynamic systems. Human factors, 37(1):32-64, 1995.

[9] M. Grootjen, M. Neerincx and J. Veltman. Cognitive task load in a naval ship control centre: from identification to prediction. Ergonomics, 49(12-13):1238–1264, 2006.

[10] Van Beurden, M., Streefkerk, J.W., Kampphuis, W., Jetten, A., van Trijp, S., Venrooij, W., and Veldhuijs, G. A simulation dashboard to monitor cognitive workload during dismounted military. International Applied Military Psychology Symposium, June 20 – 23, Porto, Portugal, 2016.

[11] D.P. Heyman and M.J. Sobel. Stochastic models in operations research: stochastic optimization, volume 1. McGraw-Hill, 1982.

[12] S. Karlin and H. Taylor. A First Course in Stochastic Processes, vol. I. Academic Press, 1975.

[13] M.J. Kochenderfer, C. Amato and H.J.D. Reynolds. Decision making under uncertainty: theory and application. MIT Press, 2015.

[14] V. Krishnamurthi. Partially Observed Markov Decision Processes. Cambridge University Press, 2016.

[15] H. Kurniawati, D. Hsu and W.S. Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In Robotics: Science and systems, volume 2008. Zurich, Switzerland, 2008.

[16] H.J. Kushner. Acceptance speech for the Richard E. Bellman control heritage award. http://a2c2.org/awards/richard-e-bellman-control-heritage-award/2004-00-00t00000s0/harold-j-kushner, July 2004. Accessed: 2018-02-24.

[17] S.M. Ross. Introduction to probability models. Academic Press, 11 edition, 2014.

[18] N. Science and T. Organization. SAS-107 technical activity proposal (tap): Factoring situational awareness and communications in operational models of dismounted combat, October 2013.

[19] J.G. Taylor. Lanchester models of warfare. Technical report, Operations Research Society of America, March 1983. 2 volumes.

[20] A.R. Washburn and M. Kress. Combat modeling. Springer, 2009.

Annex A – COGNITIVE WORKLOAD FRAMEWORK

This annex presents the background concepts to SA and discusses the impact of other factors on its acquisition. It is important to understand that SA is a conceptual construct that helps the rationalization and discussion of the amount and quality of understanding that an individual has of the situation in which they are. Within our context, there are three aspects that directly impact on the ability of an individual to gain understanding; these are:

- The understanding of the current intent;
- The division of activities between contexts, such as reviewing a map, monitoring the horizon, etc.; and
- The cognitive workload of both undertaking the tasks and concurrently constructing that understanding.

The following subsections discuss each of these aspects in turn.

A.1 INTENT

Intent is the driver that determines the relative importance of information that is being fed to the individual. It can be argued that an individual is constantly receiving information from their senses (sensors) and is constantly monitoring this for information that helps build the higher level understanding (comprehension) of the situation. A simple example is that the key pieces of information for a soldier trying to identify IEDs and a botanist trying to identify where rare plants may be found are significantly different, even though the effective ground truth (i.e. the ground and ground cover that they are looking at) could be the same.

Therefore the intent is critical in the determination of what information is relevant and useful.

The intent also impacts on the transition between the perception of elements in the environment to the comprehension of those perceived elements and the prediction of likely futures. A simple example is when approaching a T junction where you wish to merge with the flow of traffic: perception is concerned with each of the oncoming vehicles and their distance and speed, moved to a comprehension that there is a gap between the vehicles and a prediction that that gap will remain as the traffic flow moves past you. In this case, the number of vehicles is actually irrelevant; it is the gaps within the stream of vehicles that is important. The intent means that the desired “observation” that will trigger an individual’s SA is a gap in the flow and the way that that gap is moving.

In the first example (the field) it is the perceptive aspects building the SA that are affected by the intent, in the second example (the traffic) it is the comprehended and predictive aspects that are affected by the intent.

In a military context it is assumed that the intent is encapsulated by the “Commander’s intent”; however individuals can have multiple intents (both tacit and implicit) that either overlap or, in certain circumstances, conflict. In their processing of comprehension and prediction, an individual may well apply some “fuzzy logic” to the relative importance of the different intents that they currently hold.

The individuals within a team are likely to have different individual perceptions of these intents. However a well-trained team will have complementary individual intents and will have a collective understanding of the relative importance of their different intents.

A.2 TASKS

At the task level there are also aspects that require understanding. During an activity it is extremely rare for an individual to only be conducting one task within a time "window": effectively all individuals "time slice". A task may require an individual's attention to be focused on a specific subset of their environment: e.g. when looking at a map or the output of a digital sensor, you cannot simultaneously monitoring the horizon and therefore, at that precise moment, would have less chance to acquire information from another source.

This is not an absolute "an individual can only do one thing at a time": as is discussed in the section on workload, an individual's capacity can expand to encompass what might normally be considered an excessive amount of activity in extreme circumstances.

Across a team, there is also likely to be a difference in their division of activities: i.e. the Commander is likely to spend more time on a map and on command nets than an individual soldier.

This time slicing is affected by: the type of task it is, i.e. whether it is skill-based, rule-based or knowledge-based; the time occupied; and aspects to do with the process of switching contexts, i.e. number of tasks, number of contexts and the separation of data across contexts. The problem here is that some of these aspects sit outside the profile for a particular sensor and are driven by the combination of sensors and contexts available.

In addition, switching between tasks also cost resources. So one should avoid unnecessary switching between tasks.

A.3 COGNITIVE WORKLOAD FRAMEWORK

For our purposes, cognitive workload is best described by Wickens' Multiple Resource Theory (see Figure A-1).

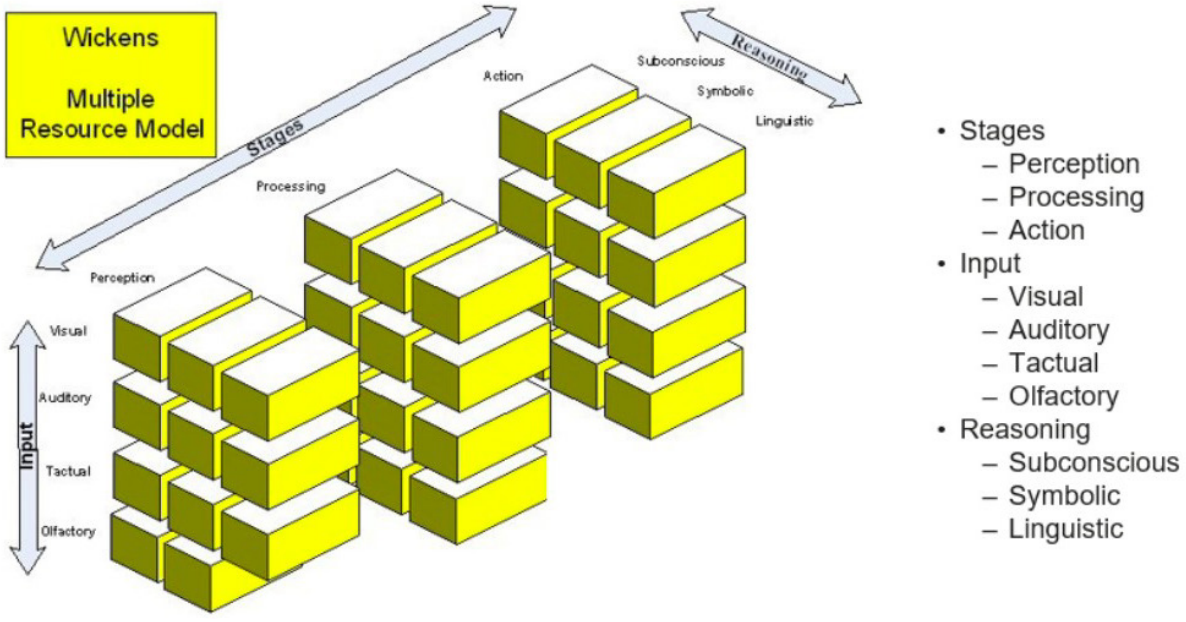


Figure A-1: Multiple Resource Model by Wickens.

Put simply, this theory presents a number of resource buckets related to the three dimensions of inputs, processing stages and reasoning/actions/decisions. As can be seen, each of these is subdivided into a number of resource areas that can be called on by an individual performing a task. This allows for a soldier to monitor a radio net whilst monitoring the horizon for other information.

These resources become important in two ways:

1) They force the separation of tasks into contexts (see above); and
2) They predict conflicts of resources that would reduce the individual's ability to achieve that task.

In our context, this manifests itself as: in times of high workload, there is a constraint on the individual to accommodate new information and then process it. I.e. an individual may have too high a workload to perceive something in the first place, but even if it is perceived the workload may be too high for the individual to use that to comprehend an aspect of the situation, or to project the consequence of that information up to the level of prediction of its influence.

It should be noted that there is not an absolute limit on the cognitive workload for any individual: people adopt coping strategies and can cope with peaks of high workload that vastly exceed the level that is manageable over an extended period of time. There is also a significant variation between individuals as to the level of cognitive workload that they can accommodate.

A.4 IMPACT ON SITUATIONAL AWARENESS (SA)

From the discussion regarding intent, we can say that the observations that we are tracing through the model would inherently embody the three levels of SA.

From the task level, we can accommodate the centre of focus modality.

From the workload, we can accommodate the overall cognitive burden.

The consequence of this is that we may need to associate the level of workload to whether or not the observation is perception, comprehension or prediction based: i.e. the workload may allow an individual to perceive something but not then directly process it into their comprehension, or comprehend it but not then be able to directly process it into prediction. It is not reasonable to assume that, if you have been able to perceive the object but at that time not been able to comprehend its significance, over the next time period you would not then be able to process it, unless during the subsequent time steps you were also restricted by workload.

The overall SA would have parameters of likelihood and time (delay).

Annex B – MILITARY SCENARIO

B.1 GENERAL

The UK provided a generic military scenario for the model, based on the action of a light role platoon within a company level operation, as part of an expeditionary NATO force. Tactical movement at platoon level is usually by foot, although in this case, initial deployment is by support helicopter from a ship. The only vehicle organic to the platoon is a four wheeled All-Terrain Vehicle ('quad'), which is not deployed with the platoon in this example.

B.2 ENVIRONMENT

The ground on which this operation is being conducted is rural and rugged. There is much forestation (copses and larger woods), which restrict visibility to short ranges. These features can be easily negotiated by foot soldiers, but vehicles are confined to tracks and roads. The host nation's population is largely centred on urban centres, but there are small settlements to support the exploitation of the forest. The weather is temperate.

B.3 ENEMY FORCES

In this scenario, the enemy force comprises irregular (militia) personnel, usually operating at no higher than platoon level (typically at a section level of 8 to 10 strength). They do not have heavy weapons / vehicles, preferring the use of IEDs, ambush and other harassment activity – rather than conventional "collective" action.

B.4 FRIENDLY FORCES MISSION

- **Company –** "A" Company (Coy) is to secure a road tunnel which is assessed as a key point, the security of which must be guaranteed to allow an armoured infantry brigade unimpeded passage through it northwards as part of the higher level plan, on a Main Supply Route. A Coy deploys by helicopter at night to locations just east of the tunnel. The mission is to clear reported enemy irregular / militia units (anticipated to be no more than section strength) from either end of the tunnel, secure it and provided a secure environment for the passage of the brigade through the area.

- **Platoon –** The scenario focuses on 1 Platoon (1 Pl), which has 4 tasks:

    1) Insert by support helicopter then move to the area of the northern end of the tunnel;
    2) Clear enemy from the northern end of the tunnel and a nearby road bridge;
    3) Protect the north end of the tunnel and provide area security; and
    4) Move 2 km north and occupy a farm complex at a T junction where another road intersects the route to be used by the armoured brigade.

  This is shown graphically below.

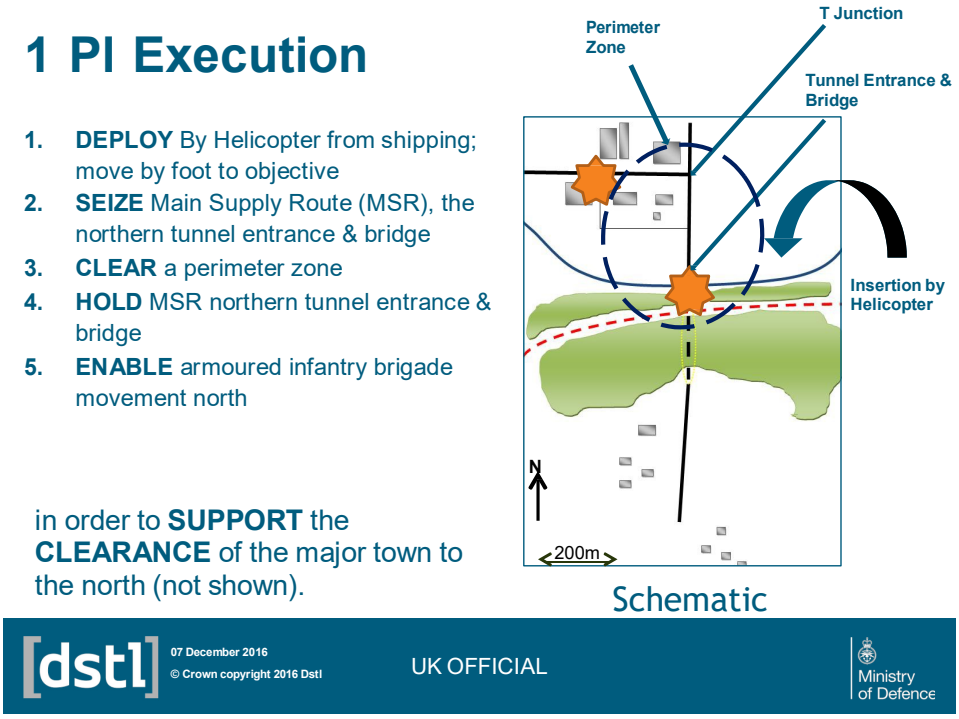


Figure B-1: Platoon Tasks.

B.5 MAIN EVENTS LIST

Starting from this scenario, an MEL was constructed for 1 Platoon's tasks. It captures:

- Sub tasks within those described above; the source of the information / direction which will form the commanders' and soldiers' initial Situational Awareness (SA) at the start of each sub task.
- A likely 'stressor' – an event or lack of knowledge, which make the local successful execution of the sub task as per the original plan an issue.
- The source of the fact that there is now a new situation – either 'internal' to the platoon, e.g. contact with the enemy with casualties taken, or 'external', e.g. information or intelligence from company HQ.
- The potential consequences to the execution envisaged in the original plan, as captured in the orders given to the platoon commander and from him/her to the soldiers through section commanders. [These unexpected events will place demands upon the 'Observation' and 'Decision' models.]

B.6 VIGNETTE

The full scenario and MEL within it are both rich and wide ranging. Here, as an example, we describe one vignette in more detail: a contact during the dismounted approach to the tunnel:

- 1 Platoon unexpectedly comes under small arms fire from an unknown number of enemy forces and suffers a number of casualties.

- The firing stops almost as soon as it began and the lead section commander reports the sounds of enemy withdrawing in the general direction of the tunnel.
- Whilst the number of casualties, the locations of the casualties and the severity of their wounds are being clarified, the Platoon Commander must decide whether he:
  - Goes firm in his current location and await reinforcements (or even withdraw) since he can no longer complete the mission given to him with the allocated resources.
  - Continues with his mission as initially planned and on the intended route. (Does he have sufficient troops to achieve the yardstick of 3:1 superiority in the attack on the tunnel? Have the enemy forces withdrawn completely or are there further pockets, which might mean selecting a different route?)
  - Continues with his mission, but in a different manner, e.g. using a different route. (What routes exist? Might there be enemy forces in different locations? Could he call for fire support?)
- The Platoon Commander needs to augment the base SA gained from the initial company Orders Group. He needs different observations to provide better understanding of enemy locations and options for a different route. His principal sources of information are internal to the platoon; observations are gathered by soldiers, using eyes, ears and issued surveillance devices and/or weapon sights. Observations are shared by voice directly between soldiers and between the commanders on a tactical VHF radio. Current radios will have narrow bandwidth and will not be capable of passing data, images or video.
- In this instance, it is confirmed that there are 3 casualties – all wounded; there are no reports of any enemy remaining on the original route, or of a greater than expected enemy force (8 to 10 people) at the tunnel. The contact conforms to the known enemy tactic of ambush and although the platoon has suffered the stressor of 3 casualties, it still can meet the 3:1 yardstick for the attack, which in any case will be against militia forces. The Platoon Commander decides to follow his original plan.

B.7 APPLICATION OF THE TECHNIQUES CONSIDERED IN THIS REPORT

- The tactical actions described above relate to the observation and decision models.
- Observations could be improved by some or all of the measures set out below (which is not an exhaustive list):
  - Provide the platoon's soldiers with improved sensors and sights, so that they have an increased capability to detect, recognise and identify enemy positions and strengths.
  - Provide the platoon with improved C4I capability, such as Dismounted Situational Awareness, so that awareness of the changing tactical situation can quickly be both established and shared.
  - Provide enhanced ISTAR capability to the platoon, e.g. an Unmanned Air System to deliver a different view of the situation ahead of the platoon.
  - Provide the platoon with increased communications bandwidth, so that pictures and/or possibly video could enhance direct line of sight observation.
- If successful, the technique could provide an analysis of consequences and cost that would follow the adoption of such measures.

Annex C – AN ILLUSTRATION OF MDPs AND POMDPs USING A THREE-STATE MODEL OF A COMBAT OUTPOST

In this section we model a combat outpost in turn as a Markov Reward Process (MRP), a MDP and a POMDP.

Let us imagine a dismounted unit having responsibility over a geographic area. Threats come in and out of the area. If unchecked, they could attack the outpost. The cost in having to defend against an attack is high. In order reduce the probability of an attack, combat patrols are scheduled. Increasing combat patrols decreases the probability of an attack, but also comes at a cost as these resources cannot be allocated to other mission objectives.

In the next sub-sections we first construct the scenario as a Discrete Time Markov Chain (DTMC), then incrementally add elements of a decision network around it, introducing a utility node in the MRP, an action node in the MDP, and finally an observation model to obtain a POMDP.

C.1 MARKOV CHAIN

Discrete Time Markov Chains (DTMCs) represent the evolution of a system as random transitions between discrete states, in discrete steps, with known transition probabilities between each. In practice these probabilities are represented as entries in a matrix, the transition matrix. Using the *transition matrix* it is possible to calculate different quantities, such as the expected time to reach one state from another, or the time spent in each state when the system runs for some time.

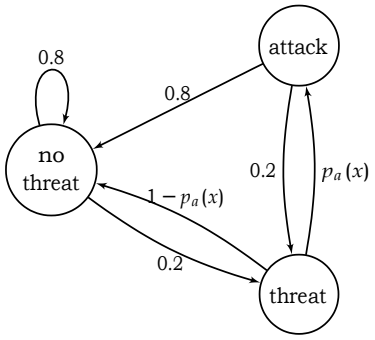
The defining property of all DTMCs, as any Markov process in general, is that they are *memoryless*: the subsequent evolution of a system only depends on the current state; it is independent of any state visited previously. The memoryless property can seem a stronger restriction than it actually is. A Markov chain can be constructed so that the system will never come back to some state visited previously, and states that only link back to themselves are called *absorbing* states. These features are particularly useful to represent attrition in combat models.

Figure C-1 shows the combat outpost problem as a Markov chain, including the state diagram, the transition matrix (T) and a graph of $p_a(x)$, the transition probability from **threat** to **attack**.

Each time step in the model can be imagined as one day. Starting in the **no threat** state, there is one chance in five (0.2) that it enters the **threat** state on any given day. From then, the threat either exits the area, returning the system to the **no threat** state, or attacks the combat post, moving the system to the **attack** state. The transition probabilities for these events depend on the number of patrols in the area, x, according to the following formula:

$$p_a(x) = \alpha(1-\beta)^x. \tag{3}$$

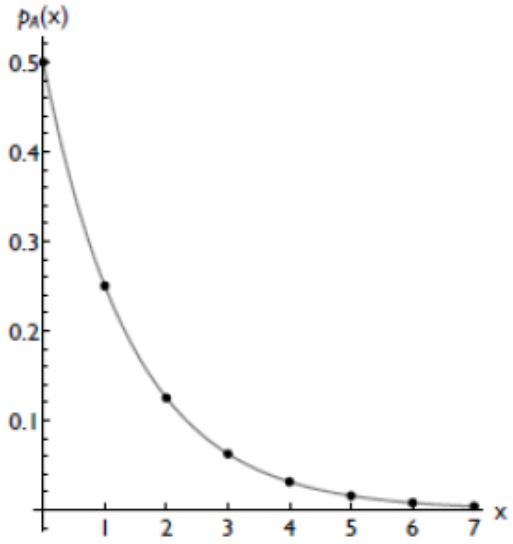
For the example we chose $\alpha = 0.5$ and $\beta = 0.5$. Figure C-1(c) shows $p_a(x)$ with these values.



(a) Diagram.

$$T = \begin{array}{c|ccc} & \text{no threat} & \text{threat} & \text{attack} \\ \text{no threat} & 0.8 & 0.2 & 0 \\ \text{threat} & 1-p_a & 0 & p_a \\ \text{attack} & 0.5 & 0.5 & 0 \end{array}$$

(b) Transition matrix.



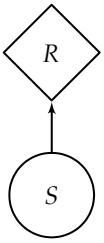
(c) Probability of attack, p_a, versus the number of patrols, x. Without patrols the probability of a pre-existing threat mounting an attack on the outpost is 50%.

Figure C-1: Discrete Time Markov Chain for the Combat Outpost Example.

C.2 MARKOV REWARD PROCESS (MRP)

C.2.1 Description

Markov Reward Processes (MRPs) add to DTMCs the idea of a reward (or penalty) associated with entering each state. This feature is illustrated in Figure C-2(a), a simple example of a *decision network*. The diagram depicts the dependency of the reward (penalty) R on the state S. At this stage the diagram is too simple to be useful: it only includes a *random node* (circle) and a *utility node* (diamond). As we progress to the next sections however the MDP will add an *action node* (square) and the POMDP an *observation model* (another type of stochastic node). Decision networks and their components will not be discussed further in this document. Details can be found in Ref. [13], Chapter 3.



(a) Decision Network for the MRP

$$R(s,x) = -\begin{array}{ll} \text{no threat} & x \\ \text{threat} & x \\ \text{attack} & x+100 \end{array}$$

(b) Reward Vector

Figure C-2: Markov Reward Process.

Figure C-2(b) shows the reward vector. It is negative (a penalty). Like the attack probability (Equation 3), it depends on x, the number of patrols. In the MRP, x can only be chosen once at $t = 0$. The number of patrols is therefore fixed at the onset, and is the same for all states.

C.2.2 Optimal Value for *x*

Our goal is to select an optimal number of patrols, x. It is the value that minimizes the average cost of running the combat outpost. This quantity is referred to as the *expected utility*. It is denoted as $U_t(s)$, where t is the step index and s is the initial state. When calculating $U_t(s)$, we usually discount the value of future events by a factor between zero and one. All other things being equal, it means that imminent events are considered more important than later ones. To make an analogy, having to pay a dollar now is always worse than paying a dollar later. However the preference for paying a dollar now versus paying *two* dollars later depends on the time elapsed, and one's discount rate. For the purpose of this example we fix $\gamma = 0.9$, simply because it allows for a quick convergence.[1]

When the time elapsed is long enough and $\gamma < 1$ then $U_t(s)$ converges. The index t can then be dropped. The value of $U(s)$ can be calculated iteratively with the following formula:

$$U(s) = R(s,x) + \gamma \sum_{s'} T(s'|s,x)U(s') \quad , \tag{4}$$

where R is the immediate reward and the T's are the transition probabilities (Figure C-1(a)).

Note that both R and T depend on x, the number of patrols. We have shown in Figure C-1(c) that patrols reduce the probability of an attack (and incurring a strong penalty of -100); however there are diminishing returns in increasing x too much. On the other hand, the reward vector (Figure C-2(b)) shows that the cost of patrols increases linearly with x. Passed a certain value, the patrols will cost more than the marginal dissuasion they provide. To find the optimum value of x, we solve Equation 4 from $x = 0$ until the maximum value of U is found. Figure C-3 shows the value of $U(s$ = no threat) versus x for $\gamma = 0.90$. The optimal number of patrols is $x = 2$, resulting in $U = -38.0$.

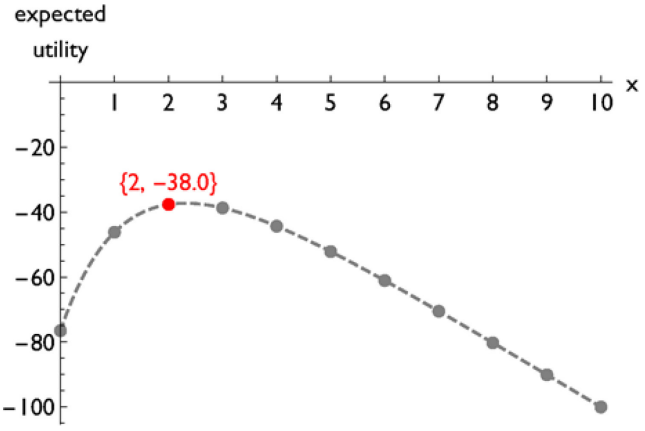


Figure C-3: Long-Term Expected Utility for the Outpost MRP versus *x*, the Number of Patrols, Assuming the Initial State "No Threat" and a Discount Factor of 0.9 per Step.

[1] If $\gamma = 0.9$ then $\gamma^{30} \sim 0.042$. Taking 1 step per day as a guideline in the outpost example, this means that we discount events to occur a month from now by 96%. This discount might be too high if the model was for an actual application, rather than an example.

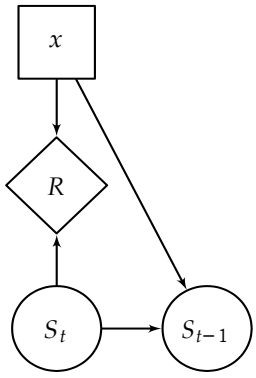
C.3 MARKOV DECISION PROCESS (MDP)

C.3.1 Description

Like MRPs, MDPs are models in which random events are formulated as a Markov chain and include rewards and/or penalties. An additional feature is that several *actions* are available to the decision-maker. These actions give some pre-defined control over the transition probabilities, T, and the rewards, R, at each step of the system's evolution.

The decision maker faces the problem of finding an optimal *policy*: a list of actions to execute in each state in order to maximize rewards (minimize penalties), as given by the expected utility $U(s)$. The choice of an optimal policy is not necessarily obvious. Systematically choosing the highest immediate reward for example might be a sub-optimal strategy in the long run. Typically, solving the Bellman equation by the method of dynamic programming allows to find an optimal policy [5]. In the long-run and with a discount rate $\gamma < 1$, the optimal policy converges to a single action for each state of the DTMC.

Figure C-4(a) shows the decision network for the MDP, with a reward node R and an action node x. In comparison with the MRP, the reward (truly, a penalty in our example) is now dependent on two factors: the state S and the action x.



(a) Decision Network for a MDP (only one step is shown). The square node indicates the possibility of a choice of action (x) by the decision maker.

$R(s,x) =$ −

	x=0	x=1	x=2	
no threat	0	1	2	…
threat	0	1	2	…
attack	100	101	102	…

(b) Reward Matrix. At each step, the decision maker can choose a value for *x*. In the long run, there is a single best *x* for each state.

Figure C-4: Markov Decision Process.

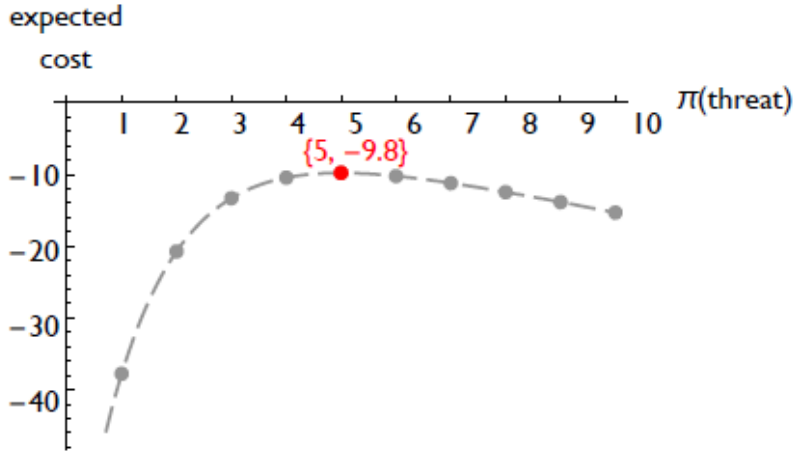
C.3.2 $\pi^*(s)$: Optimal Policy Over the States

In the MRP we were limited to choosing a single optimal value for the number of patrols x. The MDP provides more control: an optimal number of patrols must be found for each state separately. In general, finding an optimal policy for an MDP can be accomplished using a dynamic programming algorithm. Our example is so simple however that the solution is almost trivial. We already know from the transition matrix (Figure C-1(a)) that transitions from the **no threat** and **attack** states are independent of x. In these cases the only choice left is to minimize cost and choose $x = 0$. At this point the choice of an optimal policy comes down to choosing the best value for x in the **threat** state. To do this optimization we proceed the same way as in the MRP case, using a slightly modified version of Equation 4:

$$U^{\pi}(s) = R(s,\pi(s)) + \gamma \sum_{s'} T(s'|s,\pi(s)) U^{\pi}(s') \quad , \tag{5}$$

where $\pi(s)$ is the policy. In comparison with the MRP, the choice of x is no longer fixed for the whole system; in a MDP it is state-dependent.

Equation 5 is used to optimize $\pi(s$ = threat) in a manner similar to the MRP case. Figure C-5(a) shows the value of $U(s$ = no threat) versus x for $\gamma = 0.90$. The optimal number of patrol in the **threat** state is $x = 5$ (meaning that $\pi(s) = [0, 5, 0]$ if states are ordered as **no threat, threat and attack**), resulting in $U = -9.8$. This is a large improvement over $U = -38.0$ in the MRP case, attributable to the finer control allowed in the MDP.



(a) Expected Utility for the Outpost MDP versus π^* (s = Threat), the Optimal Number of Patrols Assuming the *No Threat* State and a Discount Factor $\gamma = 0.90$.

State (s)	Action (x)
no threat	0
threat	5
attack	0

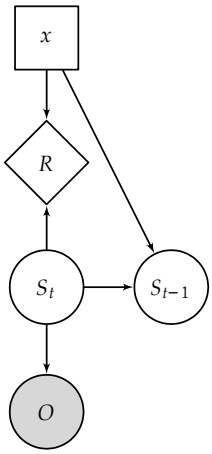
(b) Optimal Policy for the MDP.

Figure C-5: Expected Utility, and Optimal Policy for the Outpost MDP.

C.4 PARTIALLY OBSERVABLE MARKOV DECISION PROCESS (POMDP)

C.4.1 Description

In a MDP, the decision maker is aware at all time of the state. POMDPs remove that assumption. In our example the knowledge of a threat becomes probabilistic. Moreover, the probability of detecting a threat can be made to increase with the number of patrols, x. For that purpose, an Observation model O is included in the decision network (Figure C-6(a)). The Observation model takes into account imperfections in our knowledge of the state, in other words the SA. Observation models are conditional probability distributions $P(o|s, a)$. As such they can be represented by a Bayesian network (BN). In our simple example however the observation model is simple enough that we will not recourse to a BN.



(a) Decision Network for the POMDP. An observation model O has been added. O is typically a Bayesian network, composed of several random nodes. It is coloured gray to denote that fact.

$$R(s,x) = \begin{array}{lcccc} & \text{x=0} & \text{x=1} & \text{x=2} & \\ \text{no threat} & 0 & 1 & 2 & \cdots \\ \text{threat} & 0 & 1 & 2 & \cdots \\ \text{attack} & 100 & 101 & 102 & \cdots \end{array}$$

(b) Reward Matrix (same as for MDP).

Figure C-6: Partially Observable Markov Decision Process.

While the policy in a MDP can be given as a simple table (Figure C-5(b)), in a POMDP it requires a *policy diagram*. Note that for dismounted combat applications we are less interested in determining optimal actions (tactical commanders are already trained for that) than assessing how changes in SA might affect combat outcomes.

In a POMDP, the observation model can be dependent on the action. Let us imagine that the probability of detecting a threat increases with the number of patrols:

$$p_{\text{detection}} = 1 - (1 - \kappa)^x . \tag{6}$$

Figure C-7 shows the probability function if we choose $\kappa = 0.3$.

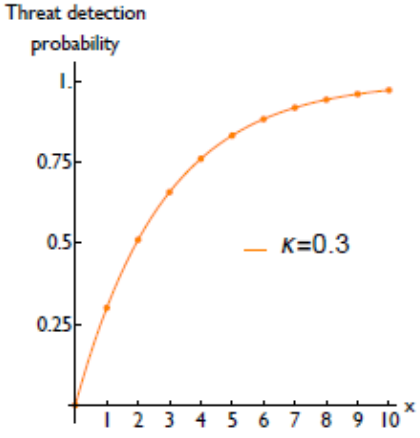


Figure C-7: Probability of Threat Detection (Equation 6) Using κ = 0.3.

Figure C-8 shows the dependence of observations on state and action.

$P(O|s, x=0) =$

	O(no threat)	O(threat)	O(attack)
no threat	1	0	0
threat	1	0	0
attack	0	0	1

$P(O|s, x=1) =$

	O(no threat)	O(threat)	O(attack)
no threat	1	0	0
threat	0.7	0.3	0
attack	0	0	1

$\vdots$

$P(O|s, x=10) =$

	O(no threat)	O(threat)	O(attack)
no threat	1	0	0
threat	0.03	0.97	0
attack	0	0	1

Figure C-8: Observation Model. $O_o(s, x)$ is the Probability of Observing o when the System State is s and the Action (Number of Patrols) is x. The middle row of O_{threat} corresponds to the graph on the left. The middle row of O_{threat} is the complement.

C.4.2 $\pi^*(b)$: Optimal Policy Over Beliefs

In the MDP cases we found an optimal policy π^* over the states s. It was a vector of length three, with one optimal action for each state. The POMDP case is different: observations have uncertainty attached to them. Most of the time we do not know the current state of the system with certainty. Rather, we maintain a probability distribution over all states, reflecting our current *belief*. Instead of a function over discrete states we now have one over a simplex, which is continuous space between vertices, one for each state. A "belief" is a point in that simplex. The closer it is to one of the vertices, the more we believe the system to be in the state associated with that vertex. Policies and the expected utility must therefore be defined on that continuous space, rather than the discrete set of states as in the MDP.

While the combat outpost example has three states, we constructed the observation model (Figure C-8) to make the detection of the **attack** state perfect. The interesting part of the belief space is therefore limited to a line between the **no threat** and **threat** states. From here on, we will assume that a t = 0, we have perfect knowledge that the system is in the **no threat** state.

For the purpose of computing $U(s)$ in the POMDP we modify Equation 5 by adding a step where beliefs are updated at each step based on the latest observation:

$$U^{\pi}(s) = R(s,x) + \gamma \left(\sum_{s'} T(s'|s,x) \sum_{o} O(o|s',x) \right) U^{\pi}(s) \quad . \tag{7}$$

The expected utility can then be given as a function of any belief vector $b(s)$ by summing over the $U^{\pi}(s)$:

$$U^{\pi} = \sum_{s} U^{\pi}(s) b(s) \tag{8}$$

Instead of determining an optimal policy over the states, we compute it over the initial beliefs. We use the observation model to take into account each subsequent observation. The POMDP policy is then expressed as a set of trees, each tree associated with a single interval of the initial belief space. Figure C-9(a) shows the set of trees in a somewhat compressed form. With our assumption that we start in the **no threat** state with perfect certainty, we should begin at the leftmost node (with action 0). The plan is then to follow at each step the branch corresponding to the latest observation. As can be seen from the graph, the policy tree takes a few steps to stabilize. Some nodes disappear since our simple observation model limits us to only a few sub-regions of the belief space. Initially however we could have any degree of belief. This diagram also allows us to choose the optimal action when information comes in from outside of the observation model, allowing us to re-calibrate our actions by starting at the top of the diagram again.

C.5 EXAMPLE: WHAT IS THE VALUE OF IMPROVING THE THREAT DETECTION PROBABILITY?

In the last section we introduced the POMDP idea using $\kappa = 0.3$ in Equation 6. In this section we compare that previous case (referred to as *baseline*) to an improved one, this time assuming $\kappa = 0.5$ in Equation 6. By comparing the expected utilities in both cases, we can see how POMDPs give us the ability to quantify the value of SA.

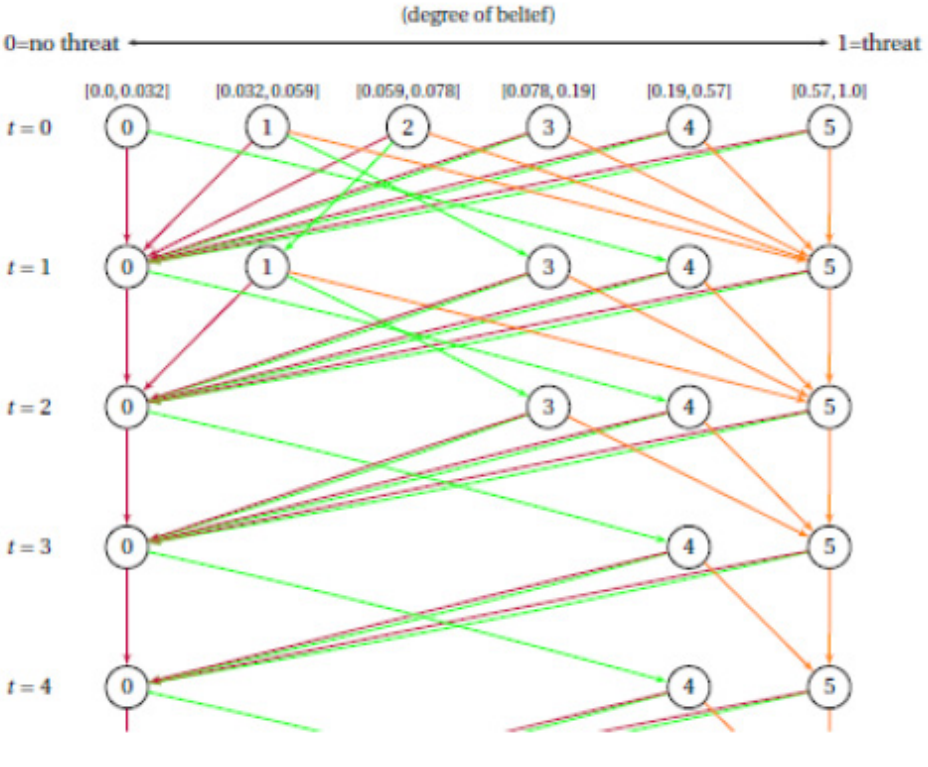


Figure C-9: Optimal Policy (π^* (b)) for the POMDP: Policy Graph. The number in each node corresponds to the optimal action at each point. Green lines indicate the optimal action following "*no threat*" observations, orange lines for "*threat*" observations and purple ones for "attack". Brackets at the top of each column correspond to ranges in the belief space between the no threat and threat states.

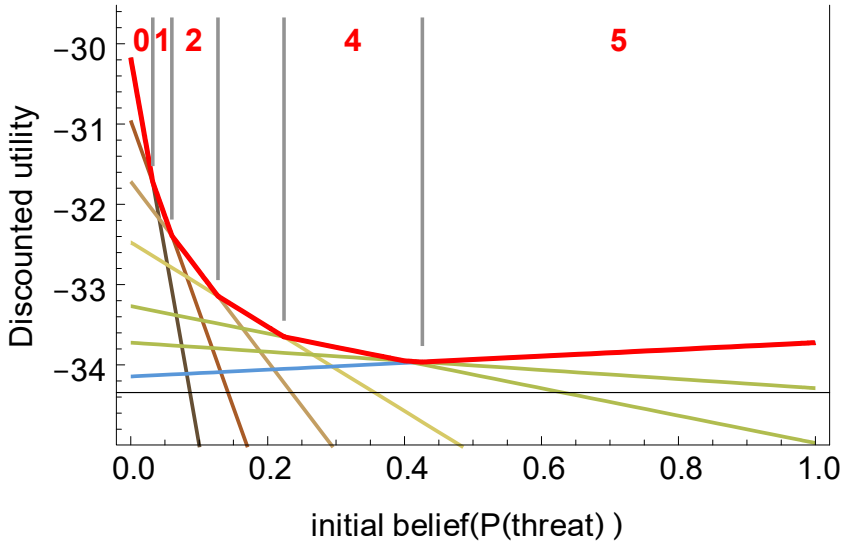


Figure C-10: Optimal Policy (π^*(b)) for the POMDP: Long-Term Expected Utility for the POMDP versus the Initial Belief.

C.5.1 Observation Model

The observation model consists of the conditional probability of making an observation, given the state of the world, and the size of patrols (x). The tables in Figure C-11 show the whole set of observation proabilities for k = 0.5, and Figure C-12 displays specifically the probability of detecting a threat, given that the threat is present, versus the size of the patrol (x). When x = 0, the threat detection probability is 0. When x is large, the threat detection probability converges to 1.

$O_{\text{no threat}}$ =

	x=0	x=1	x=2	…	x=10	…
no threat	1	1	1	…	1	…
threat	1	0.5	0.25	…	0.001	…
attack	0	0	0	…	0	…

O_{threat} =

	x=0	x=1	x=2	…	x=10	…
no threat	0	0	0	…	0	…
threat	0	0.5	0.75	…	0.999	…
attack	0	0	0	…	0	…

O_{attack} =

	x=0	x=1	x=2	…	x=10	…
no threat	0	0	0	…	0	…
threat	0	0	0	…	0	…
attack	1	1	1	…	1	…

Figure C-11: Probability of Threat Detection for κ = 0.5 in Equation 6.

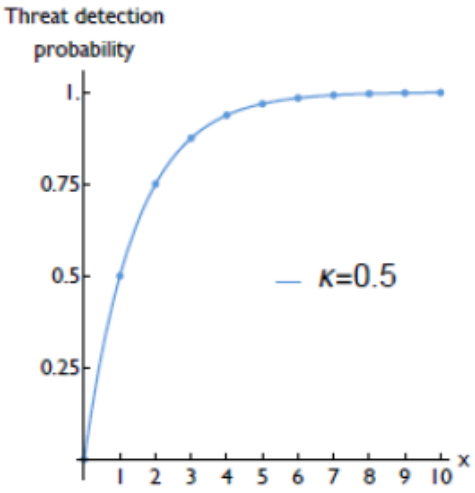


Figure C-12: Observation Model for κ = 0.5.

C.5.2 Optimal Policy

The optimal policy of a POMDP cannot depend on the initial state, since it is not known perfectly. Instead, it depends on the initial belief. Figure C-14 shows the optimal policy for our example, along with the expected utility resulting from following that policy(Figure C-14). (For more details see the caption of Figure C-9.)

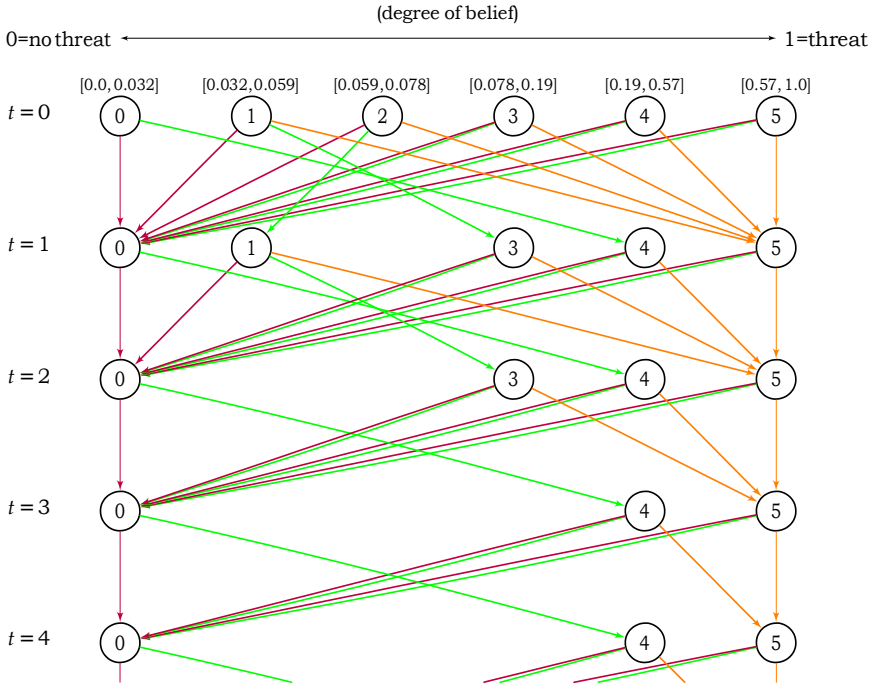


Figure C-13: Optimal Policy for the Model with Improved Detection Probability (κ = 0.5 in Equation 6): Policy Graph (π^*(b)).

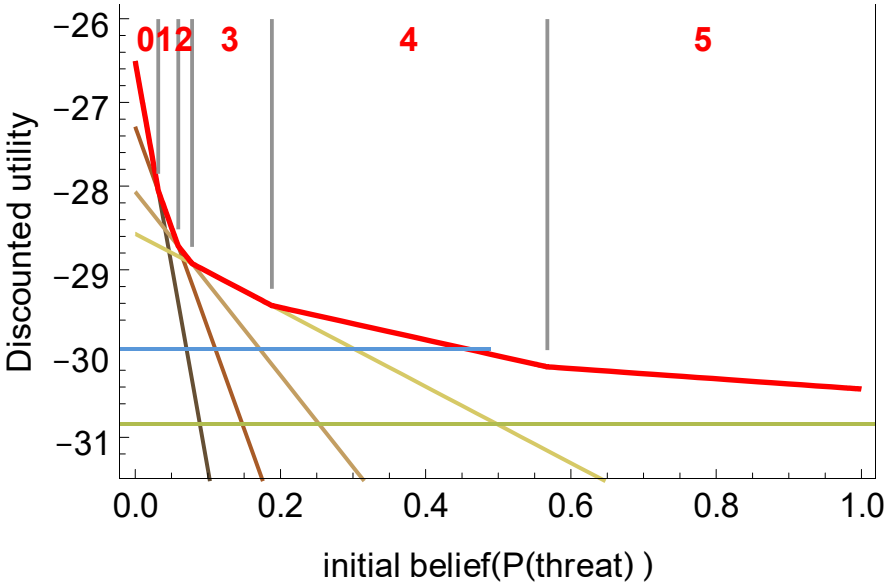


Figure C-14: Optimal Policy for the Model with Improved Detection Probability ($\kappa = 0.5$ in Equation 6): Expected utility (U^{π}(b)).

C.5.3 Comparison Between the Two Models

Assuming we start with perfect information and in an initial state of **no threat**, the expected utilities are $U = -30.1$ and $U = -26.1$ for the baseline and improved cases respectively. This difference illustrates how POMDPs can be used to quantify the gain in operational effectiveness from improving SA. Note that both cases are worse than the MDP, where we had $U = -9.8$ assuming an initial state of **no threat**. This is because a MDP represents the ideal case of perfect information *throughout the process*. Even if we start with perfect knowledge in the POMDP, uncertainty naturally increases as the system evolves, due to the noise introduced by the probability of false negatives in the observation model.

Annex D – SOURCE CODE

Continuous-Time Markov Chain class (`markovBD.py`)

```
#!/usr/bin/env python3
import pickle as pkl
import numpy as np
import scipy.sparse
import scipy.stats


def makeQ(populationSize, nServers, serviceRateMultiplier,
          failureRatePerSystem=1.0):

    def makeRange():
        return np.arange(populationSize)

    # Building index pairs for non-zero elements in the Q-matrix
    iUp = makeRange()
    jUp = makeRange() + 1
    iDown = makeRange() + 1
    jDown = makeRange()
    iDiag = np.arange(populationSize+1)
    jDiag = iDiag
    i = np.concatenate((iUp, iDown, iDiag))
    j = np.concatenate((jUp, jDown, jDiag))

    maximumFailureRate = failureRatePerSystem * populationSize
    maximumServiceRate = maximumFailureRate * serviceRateMultiplier
    # When totalServiceCapacity is split among many servers, the
    # service rate is reduced at high availability (i.e. when the
    # number of systems up for repair is lower than the number of servers)
    if nServers == 0:
        serviceRateFactors = np.zeros(populationSize)
    else:
        serviceRateFactors = np.concatenate(
            (np.ones(populationSize-nServers+1),
             np.linspace(1-1/nServers, 1/nServers, nServers-1)))
    qUp = maximumServiceRate * serviceRateFactors
    qDown = iDown * failureRatePerSystem
    qDiag = - np.insert(qUp, len(qUp), 0) - np.insert(qDown, 0, 0)
    q = np.concatenate((qUp, qDown, qDiag))

    Q = scipy.sparse.coo_matrix(
        (q, (i, j)), shape=(populationSize + 1, populationSize + 1))
```

```
    return Q.tocsr()


def uniformizeQ(Q, Lambda=None):
    nStates = Q.shape[0]
    diag = Q.diagonal()
    minLambda = max(-diag)
    if Lambda is None:
        Lambda = minLambda
    else:
        if Lambda < minLambda:
            raise ValueError(
                'Lambda input value ('+str(Lambda)+') is too small;' +
                ' it must be at least '+str(minLambda))
    ident = scipy.sparse.diags(np.ones(nStates)).tocsr()
    U = ident + Q / Lambda
    return(Lambda, U)


def deUniformize(U, Lambda):
    nStates = U.shape[0]
    ident = scipy.sparse.diags(np.ones(nStates))
    Q = (U - ident) * Lambda
    return Q


def makeEpochRange(tmin, tmax, deltaT=1/16):
    return np.linspace(tmin, tmax, 1 + (tmax - tmin) / deltaT)


def insertJumps(path):
    jumpIdx = np.nonzero(path[:-1, -1] - path[1:, -1])[0] + 1
    nJumps = len(jumpIdx)
    jumpTimes = path[jumpIdx, 0]
    jumpBases = path[jumpIdx-1, 1]
    newPath = np.zeros((len(path) + nJumps, 2))
    oldIdx = 0
    for i in range(nJumps):
        newPath[oldIdx + i:jumpIdx[i] + i] = path[oldIdx:jumpIdx[i]]
        newPath[jumpIdx[i] + i] = [jumpTimes[i], jumpBases[i]]
        oldIdx = jumpIdx[i]
    newPath[oldIdx + nJumps:] = path[oldIdx:]
    return newPath
```

```
class Propagator:

    confidenceBounds = [.005, .995]

    def __init__(self, Q):
        self.nStates = Q.shape[0]
        self.Lambda, self.U = uniformizeQ(Q)
        self.matrixCache = {0: scipy.sparse.identity(self.nStates).tocsr()}

    def addExponentsIfMissing(self, exponents):
        existingExponents = set(self.matrixCache.keys())
        newExponents = np.array(
            list(set(exponents) - existingExponents), dtype='int')
        newMatrices = self.U**newExponents
        self.matrixCache.update(
            (i, mat) for i, mat in zip(newExponents, newMatrices))

    def calcExponentRange(self, t, cb=confidenceBounds):
        emin, emax = [int(scipy.stats.poisson.ppf(z, self.Lambda * t))
                      for z in cb]
        exponents = np.arange(emin, emax + 1)
        coefs = scipy.stats.poisson.pmf(exponents, self.Lambda * t)
        return coefs / coefs.sum(), exponents

    def makeTransitionMatrix(self, t):
        # find coefficients and matrix exponents for the weighted sum
        coefs, exps = self.calcExponentRange(t)
        # update cache with missing matrix powers (if any)
        self.addExponentsIfMissing(exps)
        # initialize the transition matrix with zeros
        T = scipy.sparse.csr_matrix((self.nStates, self.nStates))
        # build the transition matrix from the problem's uniformized matrix
        for c, e in zip(coefs, exps):
            T += c * self.matrixCache[e]
        return T

    def propagateState(self, xStart, t):
        # create the transition matrix
        T = self.makeTransitionMatrix(t)
        # convert the start state to a sparse vector
        xStart_sparse = scipy.sparse.csr_matrix(xStart)
        xEnd = xStart_sparse.dot(T)
        return xEnd
```

```
    def simulatePath(self, x0, t):
        p = .99
        nmax = int(scipy.stats.poisson.ppf(p, self.Lambda * t))
        epochs = np.array([0.])
        # Generate event epochs. Each iteration creates nmax epochs. If
        # p is high enough, then this shoud require at most a few
        # iterations.
        while epochs[-1] < t:
            timesBetweenEvents = np.random.exponential(1/self.Lambda, nmax)
            epochs = np.concatenate(
                (epochs, epochs[-1] + timesBetweenEvents.cumsum()))
        # Trim epochs
        epochs = epochs[epochs <= t]
        # Generate states
        levels = np.empty(len(epochs), dtype='int')
        levels[0] = x0
        stateRange = np.arange(self.nStates)
        for i in range(len(epochs) - 1):
            oldState = levels[i]
            newState = np.random.choice(
                stateRange, p=self.U[oldState].toarray().flatten())
            levels[i+1] = newState
        path = np.column_stack((epochs, levels))
        return path


if __name__ == '__main__':
    populationSize = 15
    nServers = 2
    serviceRateFactors = np.array([0.5, 1.0, 2.0])
    Qdict = {s: makeQ(populationSize, nServers, s) for s in serviceRateFactors}
    propDict = {s: Propagator(Q) for s, Q in Qdict.items()}

    minEpoch, maxEpoch = (0., 3.)
    epochs = makeEpochRange(minEpoch, maxEpoch)
    initState = np.append(np.zeros(populationSize), 1)

######################################################################
# Calculate state vectors

    def propagateStates():
        # Fleet decay
        stateVecs1 = {
            s: {t: propDict[s].propagateState(initState, t) for t in epochs}
            for s in serviceRateFactors}
```

```
        # Fleet recovery
        stateVecs2 = {s1:
                      {t1:
                       {s2:
                        {t2:
                         propDict[s2].propagateState(
                             stateVecs1[s1][t1], t2 - t1)
                         for t2 in makeEpochRange(t1, maxEpoch)}
                        for s2 in serviceRateFactors[serviceRateFactors > s1]}
                       for t1 in epochs}
                      for s1 in serviceRateFactors}
        stateVecsDict = {'decay': stateVecs1, 'recover': stateVecs2}
        return stateVecsDict

    # Execution time can be long - leave commented out for testing

    # stateVecsDict = propagateStates()
    # with open('decayRecoverData.pkl', 'wb') as f:
    #     pkl.dump(stateVecsDict, f)


##########################################################################
# Simulate paths

    def simulatePaths(nPaths):

        def simulatePathsDecay(nPaths):
            pathsDecay = {s:
                          [propDict[s].simulatePath(populationSize, maxEpoch)
                           for _ in range(nPaths)] for s in serviceRateFactors}
            return pathsDecay

        def simulatePathsRecover(pathsDecay):
            def makeFullPath(pathDecay, t, s2):
                p1 = pathDecay[pathDecay[:, 0] <= t]
                halfwayState = p1[-1, 1]
                recoveryDuration = maxEpoch - t
                p2 = propDict[s2].simulatePath(halfwayState, recoveryDuration)
                p2[:, 0] += t
                return np.concatenate(
                    (p1, p2, [[maxEpoch, p2[-1, 1]]]), axis=0)
            return {s1:
                    {t:
                     {s2:
                      [makeFullPath(pathDecay, t, s2)
```

```
                    for pathDecay in pathsDecay[s1]]
                   for s2 in serviceRateFactors[serviceRateFactors > s1]}
                  for t in epochs} for s1 in serviceRateFactors}

    pathsDecay = simulatePathsDecay(nPaths)
    pathsRecover = simulatePathsRecover(pathsDecay)
    pathsDict = {'decay': pathsDecay, 'recover': pathsRecover}
    return pathsDict

pathsDict = simulatePaths(5)
with open('decayRecoverPaths.pkl', 'wb') as f:
    pkl.dump(pathsDict, f)
```

Useful functions (`pomdpFunctions.py`)

```
#!/usr/bin/env python3
"""Utility functions for the SAS-107 classes."""

import numpy as np

def truncateProbabilityVector(vec, truncatedWeight):
    """Take the smallest nonzero elements up to a combined weight of
truncatedWeight, and make them zero.

    """
    vecOrder = np.argsort(vec)
    removeIdx = vecOrder[vec[vecOrder].cumsum() < truncatedWeight]
    vec[removeIdx] = 0.
    return vec / vec.sum()

def isIterable(obj):
    """Returns True if obj is iterable"""
    return hasattr(type(obj), '__iter__')

def makeTupleToIdFunc(subgroupSizes):
    def tupleToId(tup, mode='raise'):
        """Translates a state tuple to a single, unique index. See the
documentation of numpy.ravel_multi_index for an explanation of the
argument 'mode'.

        """
        if isIterable(tup[0]):
            # input is a sequence of states. Transform it for use in
            # ravel_multi_index .
            arg = np.vstack(tup).T
        else:
            # input is a single state. Pass it directly.
            arg = tup
        return np.ravel_multi_index(arg, subgroupSizes + 1, mode)
    return tupleToId

def makeIdToTupleFunc(subgroupSizes):
    def idToTuple(idx):
        """Translates a state index to a state tuple."""
        tup = np.unravel_index(idx, subgroupSizes + 1)
        if isIterable(idx):
            return np.vstack(tup).T
        else:
```

```
            return tup
    return idToTuple
```

Observation model class (`observationModels.py`)

```
#!/usr/bin/env python3

import numpy as np
import scipy.sparse
import scipy.stats
import pomdpFunctions as funcs

import importlib
importlib.reload(funcs)

class ObservationModel:
    """Parent class. Child classes must implement the calcProbObs method

    The implementation is fragile because it defines nActions and
    nStates; these should be passed instead by the TunnelProblem
    object to the ObservationModel contructor.
    """

    def __init__(self,
                 subgroupSizes,
                 probSingleRedDetect_bySoldier,
                 probSingleRedDetect_byUav,
                 actionNames=['goLeft', 'goRight', 'interrupt']):
        # Input parameters
        self.actionNames = actionNames
        self.actionDict = {actionNames[i]: i for i in range(len(actionNames))}
        self.subgroupSizes = np.array(subgroupSizes)
        self.probSingleRedDetect_bySoldier = probSingleRedDetect_bySoldier
        self.probSingleRedDetect_byUav = probSingleRedDetect_byUav

        # Transformed parameters
        self.nActions = len(actionNames)
        self.nStates = (self.subgroupSizes + 1).prod()
        self.nObservations = (self.subgroupSizes + 1).prod()
        self.tupleToId = funcs.makeTupleToIdFunc(self.subgroupSizes)
        self.idToTuple = funcs.makeIdToTupleFunc(self.subgroupSizes)
        # stopObs is usually chosen to be the one with index 0.
        self.stopObs = np.zeros(len(self.subgroupSizes), dtype='int')
        self.stopObsId = self.tupleToId(self.stopObs)

    def calcProbObs(self, *args, **kwargs):
        """Placeholder. This method must be implemented by child classes."""
        pass
```

```
    def makeObservationMatrices(self,
                                truncate_rows=True,
                                truncatedWeight=0.001):
        """Creates the observation model.

The observation model is a matrix where each element (i,j) corresponds
to P(obs_i|state_j). When truncate_rows is True, we force the smallest
elements on each row to zero, one by one until we have taken at most
truncatedWeight away (typically a small fraction, for example 1%).
This increases the observation model's sparsity without changing its
properties too much.

        """
        OList = [scipy.sparse.dok_matrix((self.nStates, self.nObservations))
                 for _ in range(self.nActions)]
        actionIdsRemaining = list(range(self.nActions))  # all actions
        interruptId = self.actionDict['interrupt']
        # the observation matrix for the 'interrupt' action is trivial
        OList[interruptId][:, self.stopObsId] = 1.
        # remove 'interrupt' action
        actionIdsRemaining.remove(interruptId)
        # Iterate over remaining actions
        for actionId in actionIdsRemaining:
            for i in range(self.nStates):
                endState = self.idToTuple(i)
                nBlue = endState[0]
                nLeft, nRight = endState[1:]
                # We do not consider false positives, only false
                # negatives; the commander can't make the mistake of
                # counting more blue or red than there are on the
                # ground.
                observations = [[x, y, z]
                                for x in [nBlue]
                                for y in range(nLeft+1)
                                for z in range(nRight+1)]
                obsIds = np.ravel_multi_index(
                    np.array(observations).T,
                    self.subgroupSizes + 1)
                row = np.zeros(self.nObservations)
                for k in range(len(observations)):
                    j = obsIds[k]
                    obs = observations[k]
                    row[j] = self.calcProbObs(actionId, endState, obs)
                if truncate_rows:
```

```
                    row = funcs.truncateProbabilityVector(row, truncatedWeight)
                OList[actionId][i] = row
        return OList


class IndistinctTargetsOM(ObservationModel):
    """Detection by individual sensors, without data fusion.
    """

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def calcProbMaxOneSide(self, nSeenMax, nActual, probSingle, nSensors):
        if nSeenMax >= nActual:
            pMoreSingle = 0.
        else:
            # Note: sf is defined as (1 - cdf).
            pMoreSingle = scipy.stats.binom.sf(nSeenMax, nActual, probSingle)

        if nSeenMax == 0:
            pLessSingle = 0
        else:
            pLessSingle = scipy.stats.binom.cdf(
                nSeenMax-1, nActual, probSingle)
        # Prob. that at least one in group detects more than nSeenMax
        pMoreGroup = 1 - scipy.stats.binom.pmf(0, nSensors, pMoreSingle)
        # Prob. that all in group detect less than nSeenMax
        pLessGroup = pLessSingle ** nSensors
        # Leftover probability that highest detected by any in group
        # is actually nSeenMax
        return np.array((pLessGroup, pMoreGroup))

    def calcProbObs(self, actionId, endState, obs):
        """Each sensor detects a subset of Red according to a binomial
distribution. The observation is the maximum number of Red detected by
any sensor, for each Red subgroup.

        """
        actionName = self.actionNames[actionId]
        nBlueNow = endState[0]
        nRedNow = endState[1:]
        nBlueObs = obs[0]
        nRedObs = obs[-2:]
        if actionName == 'interrupt':
            if np.count_nonzero(obs) == 0:
```

```
                return 1.
            else:
                return 0.
        elif nBlueObs != nBlueNow:
            return 0.
        else:
            if actionName == 'goLeft':
                orderedObs = nRedObs
                orderedActuals = nRedNow
            elif actionName == 'goRight':
                orderedObs = reversed(nRedObs)
                orderedActuals = reversed(nRedNow)

            obsActionSide, obsOtherSide = orderedObs
            actActionSide, actOtherSide = orderedActuals

            probSoldiersLess, probSoldiersMore = self.calcProbMaxOneSide(
                obsActionSide,
                actActionSide,
                self.probSingleRedDetect_bySoldier,
                nBlueNow)
            probSoldiersExactly = 1 - probSoldiersLess - probSoldiersMore
            probUavActionLess, probUavActionMore = self.calcProbMaxOneSide(
                obsActionSide,
                actActionSide,
                self.probSingleRedDetect_byUav,
                1)
            probUavOtherLess, probUavOtherMore = self.calcProbMaxOneSide(
                obsOtherSide,
                actOtherSide,
                self.probSingleRedDetect_byUav,
                1)
            probUavActionExactly = 1 - probUavActionLess - probUavActionMore
            probUavOtherExactly = 1 - probUavOtherLess - probUavOtherMore

            probActionSide = probSoldiersLess * probUavActionExactly +\
                probSoldiersExactly * probUavActionLess +\
                probSoldiersExactly * probUavActionExactly

            return probActionSide * probUavOtherExactly


class DistinctTargetsOM(ObservationModel):
    """Targets are distinguishable from each other - as if there was a
    data fusion capacity that combines all individual sensor outputs
```

```
    unambiguously.

    """

    def __init__(self, *args, **kwargs):
        self.probObsMemo = {}
        super().__init__(*args, **kwargs)

    def calcProbObs(self, actionId, endState, obs):
        actionName = self.actionNames[actionId]
        nBlueNow = endState[0]
        nRedNow = endState[1:]
        nRedObs = obs[-2:]
        # Simplest case is if the action is "interrupt"
        if actionName == 'interrupt':
            if np.count_nonzero(obs) == 0:
                return 1.
            else:
                return 0.
        else:
            # Memoization saves a bit of time
            if actionName == 'goLeft':
                key = tuple([nBlueNow] +
                            list(nRedNow) +
                            list(nRedObs))
            elif actionName == 'goRight':
                key = tuple([nBlueNow] +
                            list(nRedNow[-1::-1]) +
                            list(nRedObs[-1::-1]))
            if key in self.probObsMemo:
                return self.probObsMemo[key]
            # If prob obs isn't memoized we must calculate it.
            probSingleRedDetect_fromGround = (
                1 - (1 - self.probSingleRedDetect_bySoldier)**nBlueNow)
            probSingleRedDetect_actionSide = (
                1 - (1 - probSingleRedDetect_fromGround) *
                (1 - self.probSingleRedDetect_byUav))
            probSingleRedDetect_otherSide = self.probSingleRedDetect_byUav

            if actionName == 'goLeft':
                probLeftSingle = probSingleRedDetect_actionSide
                probRightSingle = probSingleRedDetect_otherSide
            elif actionName == 'goRight':
                probLeftSingle = probSingleRedDetect_otherSide
                probRightSingle = probSingleRedDetect_actionSide
```

```
        else:
            raise(ValueError('action name "'+actionName+'" undefined.'))

        probLeftObs, probRightObs = [scipy.stats.binom.pmf(k, n, p)
                                     for k, n, p in zip(
                                       nRedObs,
                                       nRedNow,
                                       [probLeftSingle, probRightSingle])]

        probObs = probLeftObs * probRightObs
        self.probObsMemo[key] = probObs
        return probObs
```

Tunnel problem class (`tunnelProblem.py`)

```
#!/usr/bin/env python3

import numpy as np
import scipy.sparse
import scipy.stats
import markovBD
import pomdpFunctions
import observationModels

import importlib
importlib.reload(pomdpFunctions)
importlib.reload(observationModels)


# Print large arrays without ellipses
np.set_printoptions(threshold=np.inf, precision=3)


class TunnelProblem:

    def __init__(self,
                 observationModelClass=observationModels.DistinctTargetsOM,
                 nBlueMax=12,
                 nRedMax=6,
                 nRedMin=2,
                 blueEffectiveness=1.0,
                 redEffectiveness=1.0,
                 deltaTime=.1,
                 discountRate=0.99,
                 blueCasualtyReward=-1,
                 interruptReward=-1,
                 rateSingleRedDetect_bySoldier=0.01,  # each period
                 rateSingleRedDetect_byUav=0.01):

        # Set attributes
        if nRedMin > nRedMax:
            raise ValueError('nRedMin (' + str(nRedMin) + ')' +
                             ' must be smaller or equal to' +
                             ' nRedMax (' + str(nRedMax) + ').')
        self.nBlueMax = nBlueMax
        self.nRedMax = nRedMax
        self.nRedMin = nRedMin
        self.blueEffectiveness = blueEffectiveness
```

```
        self.redEffectiveness = redEffectiveness
        self.deltaTime = deltaTime
        self.discountRate = discountRate
        self.blueCasualtyReward = blueCasualtyReward
        self.interruptReward = interruptReward
        # Time to detection modelled as an exponential random variable
        self.probSingleRedDetect_bySoldier = (
            1 - np.exp(-deltaTime * rateSingleRedDetect_bySoldier))
        self.probSingleRedDetect_byUav = (
            1 - np.exp(-deltaTime * rateSingleRedDetect_byUav))

        # Actions
        self.actionNames = ['goLeft', 'goRight', 'interrupt']
        self.nActions = len(self.actionNames)
        self.actionDict = {self.actionNames[i]: i
                           for i in range(len(self.actionNames))}

        # States
        self.subgroupSizes = np.array(
            [self.nBlueMax, self.nRedMax, self.nRedMax])
        self.nStates = (self.subgroupSizes + 1).prod()
        # Utility functions
        self.tupleToId = pomdpFunctions.makeTupleToIdFunc(self.subgroupSizes)
        self.idToTuple = pomdpFunctions.makeIdToTupleFunc(self.subgroupSizes)

        # Define the interrupt state as (0,0,0)
        self.stopState = np.zeros(len(self.subgroupSizes), dtype='int')
        self.stopStateId = self.tupleToId(self.stopState)

        # Observations
        self.observationModel = observationModelClass(
            self.subgroupSizes,
            self.probSingleRedDetect_bySoldier,
            self.probSingleRedDetect_byUav,
            self.actionNames)
        self.nObservations = self.observationModel.nObservations

    # 2.3 Initial belief
    def makeInitialBelief(self):
        initialBelief = np.zeros(self.nStates)
        # The enemy is split between the left and the right routes,
        # but we don't know how exactly.
        initialBelief[self.tupleToId((self.nBlueMax,
                                      self.nRedMax,
                                      self.nRedMin))] = 1.
```

```
        initialBelief[self.tupleToId((self.nBlueMax,
                                      self.nRedMin,
                                      self.nRedMax))] = 1.
        # Normalize the state vector
        initialBelief = initialBelief / initialBelief.sum()
        return initialBelief

    # 3. Transition probabilities
    # 3.1 Given the action, detect if state is terminal
    def stateIsTerminal(self, state, actionId):
        nBlue = state[0]
        nRedLeft, nRedRight = state[1:]
        actionName = self.actionNames[actionId]
        ans = False
        if nBlue == 0:
            ans = True
        else:
            if actionName == 'goLeft' and nRedLeft == 0:
                ans = True
            elif actionName == 'goRight' and nRedRight == 0:
                ans = True
            elif actionName == 'interrupt':
                ans = True
        return ans

    # 3.2 Calculate transition rates
    def calcTransitionRate(self, actionId, startState, endState):

        diff = startState - endState

        # Transitions backwards are not allowed
        isBackwards = (diff < 0).all()
        # Transitions to self have a transition rate of zero
        isToSelf = (diff == 0).all()
        # If any of these conditions are met, return 0.
        if isBackwards or isToSelf:
            return 0.

        actionName = self.actionNames[actionId]
        if actionName == 'interrupt':

            if (endState == self.stopState).all():
                return 1.
            else:
                return 0.
```

```
        else:  # action is 'goLeft' or 'goRight'

            startStateIsTerminal = self.stateIsTerminal(startState, actionId)
            transitionIsForbidden = (diff.sum() != 1)
            if startStateIsTerminal or transitionIsForbidden:
                return 0.

            [nBlue, nRedLeft, nRedRight] = startState
            [blueIsIncapacitated, redIsIncapacitatedLeft,
             redIsIncapacitatedRight] = diff
            if actionName == 'goLeft':
                if blueIsIncapacitated:
                    return nRedLeft * self.redEffectiveness
                elif redIsIncapacitatedLeft:
                    return nBlue * self.blueEffectiveness
                else:
                    return 0.
            elif actionName == 'goRight':
                if blueIsIncapacitated:
                    return nRedRight * self.redEffectiveness
                elif redIsIncapacitatedRight:
                    return nBlue * self.blueEffectiveness
                else:
                    return 0.
            else:
                raise ValueError('action name "'+actionName+'" not recognized')

    # 3.3 Calculate Q-matrices
    def makeQMatrix(self, actionId):
        nSubgroups = len(self.subgroupSizes)
        statesFrom = self.idToTuple(range(self.nStates))
        Q = scipy.sparse.dok_matrix((self.nStates, self.nStates))
        for i in range(self.nStates):
            # The only possible end states result from a single
            # incapacitation in a single subgroup. We generate these end
            # states below. On occasion, one of the generated states will
            # have -1 as an element, and be invalid: this happens we are
            # subtracting from an already empty substate. We take care of
            # these in the nested loop that follows.
            startState = statesFrom[i]
            endStates = startState - np.identity(nSubgroups, dtype='int')
            # To take care of invalid endStates, having -1 as an element,
            # we use mode='clip' in self.tupleToId below. This transforms all
            # -1 elements back to zero. The endState then becomes
```

```
            # identical to the startState, and calcTransitionRate should
            # return 0.
            for endState in endStates:
                j = self.tupleToId(endState, mode='clip')
                if j != i:
                    Q[i, j] = self.calcTransitionRate(
                        actionId, startState, endState)
        # Diagonal elements qii of Q-matrix
        Q.setdiag(-Q.sum(axis=1))
        return Q

    def makeTransitionMatrices(self):
        actionSubset = ['goLeft', 'goRight']
        QDict = {actionName: self.makeQMatrix(
            self.actionDict[actionName]) for actionName in actionSubset}
        propDict = {actionName: markovBD.Propagator(Q)
                    for actionName, Q in QDict.items()}
        TDict = {actionName: prop.makeTransitionMatrix(self.deltaTime)
                 for actionName, prop in propDict.items()}
        # TDict = {actionName: prop.U for actionName, prop in propDict.items()}
        # For the 'interrupt' action
        interruptMatrix = scipy.sparse.dok_matrix((self.nStates, self.nStates))
        interruptMatrix[:, self.stopStateId] = 1.
        TDict['interrupt'] = interruptMatrix
        TList = [TDict[self.actionNames[i]] for i in range(self.nActions)]
        return TList

    # 5 Rewards

    def makeCasualtyRewardDict(self, Tmat):
        rewardDict = {}
        for startStateId, endStateId in scipy.sparse.dok_matrix(Tmat).keys():
            startState, endState = self.idToTuple((startStateId, endStateId))
            nBlueCasualties = (startState - endState)[0]
            rewardDict[(startStateId, endStateId)] = \
                nBlueCasualties * self.blueCasualtyReward
        return rewardDict

    def makeInterruptRewardDict(self, Tmat):
        rewardDict = {}
        for startStateId, endStateId in scipy.sparse.dok_matrix(Tmat).keys():
            if startStateId != self.stopStateId:
                startState, endState = \
                    self.idToTuple((startStateId, endStateId))
                rewardDict[(startStateId, endStateId)] = self.interruptReward
```

```
        return rewardDict

    # 6 Make the input file using the functions above
    def formatPreamble(self):
        lines = [
            'discount: ' + str(self.discountRate),
            'values: reward',
            'states: ' + str(self.nStates),
            'actions: ' + str(self.nActions),
            'observations: ' + str(self.observationModel.nObservations)
            ]
        return lines

    def formatBelief(self):
        initialBelief = self.makeInitialBelief()
        lines = ['start: ' +
                 np.array2string(initialBelief,
                                 max_line_width=np.inf).strip('[]')]
        return lines

    def formatTransitions_compact(self, TList):
        lines = []
        for actionId in range(self.nActions):
            Tsparse = TList[actionId].todok()
            for k, v in Tsparse.items():
                lines.append(' : '.join(
                    str(x) for x in ('T', actionId, *k)) + ' ' + str(v))
        return lines

    def formatTransitions_explicit(self, TList):
        lines = []
        for actionId in range(self.nActions):
            lines += ['T : ' + str(actionId)]
            Tsparse = TList[actionId]
            lines += [np.array2string(
                row.toarray(), max_line_width=np.inf).strip('[]')
                      for row in Tsparse]
        return lines

    def formatObs_compact(self, OList):
        lines = []
        for actionId in range(self.nActions):
            Obs = OList[actionId]
            for k, v in Obs.items():
                lines.append(' : '.join(
```

```
                str(x) for x in ('O', actionId, *k)) + ' ' + str(v))
        return lines

    def formatObs_explicit(self, OList):
        lines = []
        for actionId in range(self.nActions):
            Obs = OList[actionId]
            lines.append(' : '.join(str(x) for x in ['O', actionId]))
            lines.append(np.array2string(
                Obs.toarray(), separator=' ',
                max_line_width=np.inf).replace('[', '').replace(']', ''))
        return lines

    def formatRewards(self, TList):
        interruptId = self.actionDict['interrupt']
        otherIds = set(range(self.nActions)) - set([interruptId])
        rewardDict = {
            i: self.makeCasualtyRewardDict(TList[i]) for i in otherIds}
        rewardDict[interruptId] = \
            self.makeInterruptRewardDict(TList[interruptId])
        lines = []
        for actionId, dic in rewardDict.items():
            for key, val in dic.items():
                i, j = key
                lines.append(
                    ' : '.join(str(x) for x in ['R', actionId, i, j, '*']) +
                    ' ' + str(val))
        return lines

    def formatFullInput(self):
        TList = self.makeTransitionMatrices()
        OList = self.observationModel.makeObservationMatrices()
        return (self.formatPreamble() +
                self.formatBelief() +
                self.formatTransitions_compact(TList) +
                self.formatObs_compact(OList) +
                self.formatRewards(TList))
```

Batch run functions (`tunnelRun.py`)

```
#!/usr/bin/env python3
import tunnelProblem
import observationModels

import numpy as np
import time
import subprocess
import uuid
import pickle
import json
import os.path
import itertools
from collections import OrderedDict

import importlib
importlib.reload(tunnelProblem)
importlib.reload(observationModels)


def makePomdpxFile(inputDir, basename, obj, addUID=False):
    """Create POMDPX input file. First, a temporary POMDP file is created.
It is then converted to POMDPX using pomdpconvert.

    """
    if addUID:
        basename = basename+'_'+str(uuid.uuid4())
    # We must first write the input file to disk because pomdpsol does
    # not take standard input.
    tic = time.perf_counter()
    pomdpFilePath = os.path.join(inputDir, basename + '.pomdp')
    pomdpxFilePath = os.path.join(inputDir, basename + '.pomdpx')
    with open(pomdpFilePath, 'w') as f:
        f.write('\n'.join(obj.formatFullInput()))
    toc = time.perf_counter()
    print('POMDP file creation : ' + str(toc - tic) + 's.')
    # convert pomdp file to pomdpx format
    tic = time.perf_counter()
    subprocess.run(['pomdpconvert', pomdpFilePath], stdout=subprocess.PIPE)
    toc = time.perf_counter()
    print('Conversion to POMDPX : ' + str(toc - tic) + 's.')
    return pomdpxFilePath
```

```
def launchPomdpsol(pomdpxFilename,
                   flags=['--fast', '--randomization'],
                   precision=.01):
    """Launches pomdpsol and collects the standard output as a string."""
    exe = ['pomdpsol']
    args = flags + ['--precision', str(precision)]
    proc = subprocess.run(exe + args + [pomdpxFilename],
                          stdout=subprocess.PIPE,
                          stderr=subprocess.DEVNULL)
    # return raw stdout, to be parsed by parseOutput
    return proc.stdout


def parseOutput(s):
    """Parse standard output from pomdpsol."""
    lines = [line.strip() for line in s.split(b'\n')]
    lowerBound, upperBound = (float(x) for x in lines[-7].split()[3:5])
    return {'low': lowerBound, 'high': upperBound}


def makeInputFileName(params):
    longShortTransl = {'nBlueMax': 'nb',
                       'nRedMax': 'nrmax',
                       'nRedMin': 'nrmin',
                       'deltaTime': 'dt',
                       'blueEffectiveness': 'beff',
                       'interruptReward': 'rew',
                       'rateSingleRedDetect_bySoldier': 'rsol',
                       'rateSingleRedDetect_byUav': 'ruav'}
    filename = '_'.join([short + str(params[long])
                         for long, short in longShortTransl.items()])
    return filename


def singleRun(inputParameters, inputDir='./', outputDir='./',
              basename='singleRun', verbose=True):
    tp = tunnelProblem.TunnelProblem(**inputParameters)
    if verbose:
        print('Started...')
    pomdpxFilePath = os.path.join(inputDir, basename + '.pomdpx')
    if os.path.exists(pomdpxFilePath):
        print(pomdpxFilePath +
              ' already exists; skipping POMDPX file creation.')
    else:
        makePomdpxFile(inputDir, basename, tp)
```

```
    tic = time.perf_counter()
    outputFilePath = os.path.join(outputDir, basename + '.out')
    if os.path.exists(outputFilePath):
        print(outputFilePath +
              ' already exists; skipping calculation.')
        with open(outputFilePath, 'rb') as f:
            rawOutput = f.read()
    else:
        # launch solver and store standard output for future runs
        rawOutput = launchPomdpsol(pomdpxFilePath)
        with open(outputFilePath, 'wb') as f:
            f.write(rawOutput)
    result = parseOutput(rawOutput)
    toc = time.perf_counter()
    if verbose:
        print('Computing solution: '+str(round(toc-tic))+'s.')
        print('Value bounds: '+str(result))
    return tp, result


def batchRun(params, inputDir, outputDir, batchName, verbose=True):
    # Find parameters for which a range of values has been provided
    gridParams = {}
    for key, val in params.items():
        if hasattr(val, '__iter__'):
            gridParams[key] = val
    tuples = itertools.product(*gridParams.values())
    output = []
    for tup in tuples:
        if verbose:
            print('**************************************************')
            print('Batch iteration parameters: {' +
                  ', '.join([str(k) + ': ' + str(v)
                             for k, v in zip(gridParams.keys(), tup)]) +
                  '}')
        pdict = OrderedDict(list(params.items()) +
                            list(zip(gridParams.keys(), tup)))
        tp, result = singleRun(inputParameters=pdict,
                               inputDir=inputDir,
                               outputDir=outputDir,
                               basename=makeInputFileName(pdict))
        # No need to save a copy of the observation model
        del(pdict['observationModelClass'])
        output.append({'params': pdict, 'result': result})
    # Pickle the output
```

```
    with open(outputDir + '/' + batchName + '.pkl', 'wb') as f:
        pickle.dump(output, f)
    # Make grid
    grid = [[x['params'][k] for k in gridParams.keys()] +
            [np.mean(list(x['result'].values()))]
            for x in output]
    # Export grid to JSON so it can be read easily into R
    with open(outputDir + '/' + batchName + '.json', 'w') as f:
        json.dump(grid, f)
    return output


if __name__ == "__main__":

    sarsopDir = '../SarsopInputFiles'
    os.makedirs(sarsopDir, exist_ok=True)
    outputDir = '../Data'
    os.makedirs(outputDir, exist_ok=True)
    batchOutputDir = '../BatchOutput'
    os.makedirs(batchOutputDir, exist_ok=True)

    myInput = OrderedDict(
        {'observationModelClass': observationModels.DistinctTargetsOM,
         'nBlueMax': 12,
         'nRedMax': 6,
         'nRedMin': 1,
         'deltaTime': .1,
         'blueEffectiveness': 1.,
         'interruptReward': -1,
         'rateSingleRedDetect_bySoldier': 1.,
         'rateSingleRedDetect_byUav': 1.})

    # Single run, to test if everything works
    tp, result = singleRun(
        inputParameters=myInput,
        inputDir=sarsopDir,
        outputDir=outputDir,
        basename=makeInputFileName(myInput))

    # Batch runs

    # sensors: soldier vs uav
    gridLength = 6
    batchOutput = batchRun(
        params=OrderedDict(
```

```
            list(myInput.items()) +
            [['rateSingleRedDetect_bySoldier',
              np.geomspace(0.01, 0.4, gridLength)],
              ['rateSingleRedDetect_byUav',
               np.geomspace(0.1, 4.0, gridLength)]]),
        inputDir=sarsopDir,
        outputDir=outputDir,
        batchName='soldierVsUav')

# soldiers: sensor vs protection
gridLength = 6
newParams = [['rateSingleRedDetect_bySoldier',
                np.geomspace(0.01, 0.4, gridLength)],
               ['blueEffectiveness',
                np.linspace(1., 2., gridLength)]]
# Move parameters so they are in the right order for plotting
[myInput.move_to_end(k) for k, v in newParams]
batchOutput = batchRun(
    OrderedDict(list(myInput.items()) + newParams),
    inputDir=sarsopDir,
    outputDir=outputDir,
    batchName='detectionVsEffectiveness')

# time interval between decisions
gridLength = 20
batchOutput = batchRun(
    params=OrderedDict(
        list(myInput.items()) +
        # Equal time intervals on log scale
        [['deltaTime',
          np.geomspace(0.025, 2., gridLength)]]),
    inputDir=sarsopDir,
    outputDir=outputDir,
    batchName='timeIntervals')
```

Make figures (`makeFigs.R`)

```
library("tidyverse")
library("directlabels")
library("ggthemes")
library("jsonlite")
library("gridExtra")
library("extrafont")


loadJsonData <- function(path) {
    ## jsonFilename <- paste(basename, ".json", sep="")
    rawData <- jsonlite::fromJSON(readChar(path, file.info(path)$size))
    ## Using tibble's default naming scheme for columns ("V1", etc) would
    ## be convenient. Unfortunately that results in an error that I
    ## haven't been able to debug yet. So I name columns "x", "y", etc
    nc <- ncol(rawData)
    tidyData <- rawData %>% as_tibble() %>% setNames(c("x", "y", "z")[1:nc])
    return(tidyData)
}

## Plot functions

## Function to extract a legend
## from: https://stackoverflow.com/a/21279370/997123
## also: https://github.com/tidyverse/ggplot2/wiki/
   Share-a-legend-between-two-ggplot2-graphs
g_legend <- function(a.gplot){
  tmp <- ggplot_gtable(ggplot_build(a.gplot))
  leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
  legend <- tmp$grobs[[leg]]
  return(legend)
}

makeDummyPlotWithLegend <- function() {
    ggplot(data=tibble(x=c(1,2,1,2),
                       y=c(1,2,2,1),
                       z=c('a','a','b','b')),
           aes(x,y,color=z)) +
        geom_line() +
        scale_colour_manual(
            name="Isocontours",
            values=c("blue", "red"),
            breaks=c("a", "b"),
            labels=c("Expected mission outcome", "Budget (notional)")) +
        theme_base() +
```

```
        theme(legend.key.height = unit(.025, "npc"),
              legend.key.width = unit(.05, "npc"),
              text = element_text(size=myFontSize)) +
        ## line below from: https://stackoverflow.com/a/31519478/997123
        guides(color = guide_legend(override.aes = list(linetype =
        c("solid", "dashed"))))
}

stitchLegend <- function(a.plot, legend) {
    return(arrangeGrob(a.plot , legend,
                       widths=c(3/4, 1/4),
                       ncol = 2))
}

plotContour <- function(data, b0, xtan, ytan, mtan, fontSize=10) {
    xbreaks <- sort(unique(data$x))
    ybreaks <- sort(unique(data$y))
    c <- mtan * xtan / (ytan - b0)
    a <- (ytan - b0) / xtan**c
    plot <- ggplot(data=data, aes(x, y, z=z)) +
        stat_function(fun=function(x) {a*x**c + b0}, colour="red", linetype=2) +
        ## there will be a warning about esthetic "fill" being
        ## ignored, but it is in fact necessary for direct.label to
        ## work.
        stat_contour(aes(fill=..level..), size=.5) +
        ylim(c(min(data$y), max(data$y))) +
        xlim(c(min(data$x), max(data$x))) +
        theme_classic() +
        geom_point(colour="gray") +
        theme(text = element_text(size = fontSize)) +
        annotate("point", x = xtan, y = ytan,
                 color = "red") +
        annotate("text", x = xtan, y = ytan,
                 label = "optimal investment\n(saves most lives)",
                 colour = "red",
                 hjust = 0.1,
                 vjust = -0.5,
                 size = 3)

    labelled <- direct.label(plot, list("top.pieces", colour="blue", cex=.75))
}

##
## Main section
##
```

```
myFontSize = 10
datadir = "../Data"
figDir = "../Figures/"
if (!dir.exists(figDir))
{
    dir.create(figDir)
}

## 1. Contour plots
basenames = c("soldierVsUav", "detectionVsEffectiveness")
paths = paste0(datadir, "/", basenames, ".json")
myData <- lapply(paths, loadJsonData) %>% setNames(basenames)
## These parameters specify the curve for the fake budget isocontour in each plot.
budgetParams <- list("soldierVsUav" = list(b0=3.4,
                                           xtan=0.15,
                                           ytan=2.65,
                                           mtan=-9),
                     "detectionVsEffectiveness" = list(b0=1.75,
                                                       xtan=0.25,
                                                       ytan=1.65,
                                                       mtan=-.8))

## Combine data and budgetParams in plot
myPlots <- lapply(basenames,
                  function(name){
                      do.call(plotContour,
                              c(list(data=myData[[name]]),
                                budgetParams[[name]]))})
names(myPlots) <- basenames

## Label axes
myPlots$soldierVsUav <- myPlots$soldierVsUav +
    xlab("Detection rate by individual dismounts") +
    ylab("Detection rate by UAV")
myPlots$detectionVsEffectiveness <- myPlots$detectionVsEffectiveness +
    xlab("Detection rate by individual dismounts") +
    ylab("Rel. effectiveness of weapons and PPE")

## This is the best way I found to add a legend to the contour plots
## with fake budget isocontour: create a dummy plot, extract its
## legend, and arrange it side by side with the contour plot.
legend <- g_legend(makeDummyPlotWithLegend())
myPlots$soldierVsUav <- stitchLegend(myPlots$soldierVsUav, legend)
myPlots$detectionVsEffectiveness <-
    stitchLegend(myPlots$detectionVsEffectiveness, legend)
```

```
## Save plots
lapply(seq_along(myPlots),
       function(i){ggsave(paste0(figDir, names(myPlots)[i], ".pdf"),
                          myPlots[[i]],
                          width=unit(6,"in"),
                          height=unit(4, "in"))}) %>% invisible


## 2. Decision period vs lives at risk
basename <- "timeIntervals"
myData <- loadJsonData(paste0(datadir, "/", basename, ".json"))
## annotation origin point
xa1 <- min(myData$x) - 0.005
ya1 <- 0.6
xa2 <- 0.1
ya2 <- 0.8
ggplot(myData, aes(x, -y)) +
    geom_line() +
    geom_point() +
    ## baseline
    ## annotate("segment",
    ##          x=0,
    ##          y=0,
    ##          xend=max(myData$x),
    ##          yend=0,
    ##          linetype=2) +
    annotate("segment",
             x=0,
             y=1,
             xend=max(myData$x),
             yend=1,
             linetype=2) +
    scale_x_log10(breaks = c(0.025, 0.05, 0.1, 0.25, 0.5, 1.)) +
    theme_classic() +
    xlab("Time interval between decisions (log scale)") +
    ylab("Expected lives at risk") +
    ylim(0, -min(myData$y)+0.05) +
    ## arrow to origin
    annotate("segment",
             x=xa1,
             y=ya1,
             xend=0,
             yend=1,
             arrow=arrow(length=unit(0.02, "npc"), type="closed", angle=20),
```

```
            color="blue") +
    ## label about interrupt cost
    annotate("text", x = xa1, y = ya1 - .040,
            label = "Penalty for aborting mission",
            colour = "blue",
            hjust = 0,
            vjust = -0.5,
            size = 3) +
    annotate("text", x = xa1, y = ya1,
            label = "(interruptReward = -1)",
            colour = "blue",
            family = "Courier New",
            hjust = 0,
            vjust = -0.5,
            size = 3) +
    ## arrow to default dt
    annotate("segment",
            x=xa2,
            y=ya2,
            xend=0.1,
            yend=0,
            ## arrow=arrow(length=unit(0.02, "npc"), type="closed", angle=20),
            color="blue",
            linetype=2) +
    ## label about interrupt cost
    annotate("text", x = xa2, y = ya2 + .040,
            label = "Default used in other examples",
            colour = "blue",
            hjust = 0.5,
            vjust = -0.5,
            size = 3) +
    annotate("text", x = xa2, y = ya2,
            label = "(deltaTime = 0.1)",
            colour = "blue",
            family = "Courier New",
            hjust = 0.5,
            vjust = -0.5,
            size = 3)
ggsave(paste0(figDir, "/", basename, ".pdf"),
       width=unit(6,"in"), height=unit(4, "in"))
```

REPORT DOCUMENTATION PAGE

1. Recipient's Reference	2. Originator's References	3. Further Reference	4. Security Classification of Document
	STO-TR-SAS-107 AC/323(SAS-107)TP/865	ISBN 978-92-837-2189-5	PUBLIC RELEASE

5. Originator

Science and Technology Organization
North Atlantic Treaty Organization
BP 25, F-92201 Neuilly-sur-Seine Cedex, France

6. Title

Factoring Communications and Situational Awareness in Operational Models of Dismounted Combat

7. Presented at/Sponsored by

Final Report of SAS-107.

8. Author(s)/Editor(s)	9. Date
Multiple	March 2020
10. Author's/Editor's Address	**11. Pages**
Multiple	92

12. Distribution Statement

There are no restrictions on the distribution of this document. Information about the availability of this and other STO unclassified publications is given on the back cover.

13. Keywords/Descriptors

Combat modelling
Dismounted combat
Optimization
Partially Observed Markov Decision Processes (POMDP)
Situational awareness

14. Abstract

Defence funds dedicated to dismounted soldier systems are finite and must be divided among multiple programs. In the process, decision makers must balance investments between disparate combat technologies. Each of these technologies have, on their own, the potential to improve combat outcomes. However, deciding on the right mix can be difficult: some technologies improve lethality and protection, others improve Situational Awareness (SA). Which combination of sensors, displays, weapons, and Personal Protective Equipment (PPE) provides the best expected mission outcomes?

In this report we present a way to perform such comparisons. We present a mathematical combat model that considers the joint effects of situational awareness and lethality and looks at combat outcomes in terms of expected lives saved. The model can therefore be used to design an optimal equipment portfolio, one that will save the most lives. Our approach relies on solving Partially Observable Markov Decision Processes, and the examples we present are limited to dismounted soldier systems. Partially Observable Markov Decision Processes (POMDPs) however are a general class of probabilistic models that are also applicable to other scales of ground, air and maritime combat.

This report addresses two audiences: scientists, as operators, and decision makers, as users. On one hand, the model is based on concepts such as dynamic programming and Markov chains, requiring a mathematical background that is typical of graduates in operational research. On the other hand, interpreting the model's output is accessible to decision makers from a broad range of backgrounds.

NORTH ATLANTIC TREATY ORGANIZATION

BP 25
F-92201 NEUILLY-SUR-SEINE CEDEX • FRANCE
Télécopie 0(1)55.61.22.99 • E-mail mailbox@cso.nato.int

SCIENCE AND TECHNOLOGY ORGANIZATION

DIFFUSION DES PUBLICATIONS
STO NON CLASSIFIEES

Les publications de l'AGARD, de la RTO et de la STO peuvent parfois être obtenues auprès des centres nationaux de distribution indiqués ci-dessous. Si vous souhaitez recevoir toutes les publications de la STO, ou simplement celles qui concernent certains Panels, vous pouvez demander d'être inclus soit à titre personnel, soit au nom de votre organisation, sur la liste d'envoi.

Les publications de la STO, de la RTO et de l'AGARD sont également en vente auprès des agences de vente indiquées ci-dessous.

Les demandes de documents STO, RTO ou AGARD doivent comporter la dénomination « STO », « RTO » ou « AGARD » selon le cas, suivi du numéro de série. Des informations analogues, telles que le titre est la date de publication sont souhaitables.

Si vous souhaitez recevoir une notification électronique de la disponibilité des rapports de la STO au fur et à mesure de leur publication, vous pouvez consulter notre site Web (http://www.sto.nato.int/) et vous abonner à ce service.

CENTRES DE DIFFUSION NATIONAUX

ALLEMAGNE
Streitkräfteamt / Abteilung III
Fachinformationszentrum der Bundeswehr (FIZBw)
Gorch-Fock-Straße 7, D-53229 Bonn

BELGIQUE
Royal High Institute for Defence – KHID/IRSD/RHID
Management of Scientific & Technological Research
for Defence, National STO Coordinator
Royal Military Academy – Campus Renaissance
Renaissancelaan 30, 1000 Bruxelles

BULGARIE
Ministry of Defence
Defence Institute "Prof. Tsvetan Lazarov"
"Tsvetan Lazarov" bul no.2
1592 Sofia

CANADA
DGSIST 2
Recherche et développement pour la défense Canada
60 Moodie Drive (7N-1-F20)
Ottawa, Ontario K1A 0K2

DANEMARK
Danish Acquisition and Logistics Organization
(DALO)
Lautrupbjerg 1-5
2750 Ballerup

ESPAGNE
Área de Cooperación Internacional en I+D
SDGPLATIN (DGAM)
C/ Arturo Soria 289
28033 Madrid

ESTONIE
Estonian National Defence College
Centre for Applied Research
Riia str 12
Tartu 51013

ETATS-UNIS
Defense Technical Information Center
8725 John J. Kingman Road
Fort Belvoir, VA 22060-6218

FRANCE
O.N.E.R.A. (ISP)
29, Avenue de la Division Leclerc
BP 72
92322 Châtillon Cedex

GRECE (Correspondant)
Defence Industry & Research General
Directorate, Research Directorate
Fakinos Base Camp, S.T.G. 1020
Holargos, Athens

HONGRIE
Hungarian Ministry of Defence
Development and Logistics Agency
P.O.B. 25
H-1885 Budapest

ITALIE
Ten Col Renato NARO
Capo servizio Gestione della Conoscenza
F. Baracca Military Airport "Comparto A"
Via di Centocelle, 301
00175, Rome

LUXEMBOURG
Voir Belgique

NORVEGE
Norwegian Defence Research
Establishment
Attn: Biblioteket
P.O. Box 25
NO-2007 Kjeller

PAYS-BAS
Royal Netherlands Military
Academy Library
P.O. Box 90.002
4800 PA Breda

POLOGNE
Centralna Biblioteka Wojskowa
ul. Ostrobramska 109
04-041 Warszawa

PORTUGAL
Estado Maior da Força Aérea
SDFA – Centro de Documentação
Alfragide
P-2720 Amadora

REPUBLIQUE TCHEQUE
Vojenský technický ústav s.p.
CZ Distribution Information Centre
Mladoboleslavská 944
PO Box 18
197 06 Praha 9

ROUMANIE
Romanian National Distribution
Centre
Armaments Department
9-11, Drumul Taberei Street
Sector 6
061353 Bucharest

ROYAUME-UNI
Dstl Records Centre
Rm G02, ISAT F, Building 5
Dstl Porton Down
Salisbury SP4 0JQ

SLOVAQUIE
Akadémia ozbrojených síl gen.
M.R. Štefánika, Distribučné a
informačné stredisko STO
Demänová 393
031 01 Liptovský Mikuláš 1

SLOVENIE
Ministry of Defence
Central Registry for EU & NATO
Vojkova 55
1000 Ljubljana

TURQUIE
Milli Savunma Bakanlığı (MSB)
ARGE ve Teknoloji Dairesi
Başkanlığı
06650 Bakanliklar – Ankara

AGENCES DE VENTE

The British Library Document
Supply Centre
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
ROYAUME-UNI

Canada Institute for Scientific and
Technical Information (CISTI)
National Research Council Acquisitions
Montreal Road, Building M-55
Ottawa, Ontario K1A 0S2
CANADA

Les demandes de documents STO, RTO ou AGARD doivent comporter la dénomination « STO », « RTO » ou « AGARD » selon le cas, suivie du numéro de série (par exemple AGARD-AG-315). Des informations analogues, telles que le titre et la date de publication sont souhaitables. Des références bibliographiques complètes ainsi que des résumés des publications STO, RTO et AGARD figurent dans le « NTIS Publications Database » (http://www.ntis.gov).

NORTH ATLANTIC TREATY ORGANIZATION

BP 25
F-92201 NEUILLY-SUR-SEINE CEDEX • FRANCE
Télécopie 0(1)55.61.22.99 • E-mail mailbox@cso.nato.int

SCIENCE AND TECHNOLOGY ORGANIZATION

DISTRIBUTION OF UNCLASSIFIED STO PUBLICATIONS

AGARD, RTO & STO publications are sometimes available from the National Distribution Centres listed below. If you wish to receive all STO reports, or just those relating to one or more specific STO Panels, they may be willing to include you (or your Organisation) in their distribution.

STO, RTO and AGARD reports may also be purchased from the Sales Agencies listed below.

Requests for STO, RTO or AGARD documents should include the word 'STO', 'RTO' or 'AGARD', as appropriate, followed by the serial number. Collateral information such as title and publication date is desirable.

If you wish to receive electronic notification of STO reports as they are published, please visit our website (http://www.sto.nato.int/) from where you can register for this service.

NATIONAL DISTRIBUTION CENTRES

BELGIUM
Royal High Institute for Defence – KHID/IRSD/RHID
Management of Scientific & Technological Research for Defence, National STO Coordinator
Royal Military Academy – Campus Renaissance
Renaissancelaan 30
1000 Brussels

BULGARIA
Ministry of Defence
Defence Institute "Prof. Tsvetan Lazarov"
"Tsvetan Lazarov" bul no.2
1592 Sofia

CANADA
DSTKIM 2
Defence Research and Development Canada
60 Moodie Drive (7N-1-F20)
Ottawa, Ontario K1A 0K2

CZECH REPUBLIC
Vojenský technický ústav s.p.
CZ Distribution Information Centre
Mladoboleslavská 944
PO Box 18
197 06 Praha 9

DENMARK
Danish Acquisition and Logistics Organization (DALO)
Lautrupbjerg 1-5
2750 Ballerup

ESTONIA
Estonian National Defence College
Centre for Applied Research
Riia str 12
Tartu 51013

FRANCE
O.N.E.R.A. (ISP)
29, Avenue de la Division Leclerc – BP 72
92322 Châtillon Cedex

GERMANY
Streitkräfteamt / Abteilung III
Fachinformationszentrum der Bundeswehr (FIZBw)
Gorch-Fock-Straße 7
D-53229 Bonn

GREECE (Point of Contact)
Defence Industry & Research General Directorate, Research Directorate
Fakinos Base Camp, S.T.G. 1020
Holargos, Athens

HUNGARY
Hungarian Ministry of Defence
Development and Logistics Agency
P.O.B. 25
H-1885 Budapest

ITALY
Ten Col Renato NARO
Capo servizio Gestione della Conoscenza
F. Baracca Military Airport "Comparto A"
Via di Centocelle, 301
00175, Rome

LUXEMBOURG
See Belgium

NETHERLANDS
Royal Netherlands Military Academy Library
P.O. Box 90.002
4800 PA Breda

NORWAY
Norwegian Defence Research Establishment, Attn: Biblioteket
P.O. Box 25
NO-2007 Kjeller

POLAND
Centralna Biblioteka Wojskowa
ul. Ostrobramska 109
04-041 Warszawa

PORTUGAL
Estado Maior da Força Aérea
SDFA – Centro de Documentação
Alfragide
P-2720 Amadora

ROMANIA
Romanian National Distribution Centre
Armaments Department
9-11, Drumul Taberei Street
Sector 6
061353 Bucharest

SLOVAKIA
Akadémia ozbrojených síl gen M.R. Štefánika, Distribučné a informačné stredisko STO
Demänová 393
031 01 Liptovský Mikuláš 1

SLOVENIA
Ministry of Defence
Central Registry for EU & NATO
Vojkova 55
1000 Ljubljana

SPAIN
Área de Cooperación Internacional en I+D
SDGPLATIN (DGAM)
C/ Arturo Soria 289
28033 Madrid

TURKEY
Milli Savunma Bakanlığı (MSB)
ARGE ve Teknoloji Dairesi Başkanlığı
06650 Bakanliklar – Ankara

UNITED KINGDOM
Dstl Records Centre
Rm G02, ISAT F, Building 5
Dstl Porton Down, Salisbury SP4 0JQ

UNITED STATES
Defense Technical Information Center
8725 John J. Kingman Road
Fort Belvoir, VA 22060-6218

SALES AGENCIES

The British Library Document Supply Centre
Boston Spa, Wetherby
West Yorkshire LS23 7BQ
UNITED KINGDOM

Canada Institute for Scientific and Technical Information (CISTI)
National Research Council Acquisitions
Montreal Road, Building M-55
Ottawa, Ontario K1A 0S2
CANADA

Requests for STO, RTO or AGARD documents should include the word 'STO', 'RTO' or 'AGARD', as appropriate, followed by the serial number (for example AGARD-AG-315). Collateral information such as title and publication date is desirable. Full bibliographical references and abstracts of STO, RTO and AGARD publications are given in "NTIS Publications Database" (http://www.ntis.gov).

ISBN 978-92-837-2189-5