Dynamic interdiction games on PERT networks

Eli Gutin

Supervisor: Dr. Daniel Kuhn
Second marker: Dr. Wolfram Wiesemann

Department of Computing

Imperial College

Submitted in part fulfillment of the degree of

MEng Computing

2012

Acknowledgements

My thanks go to Dr. Daniel Kuhn and Dr. Wolfram Wiesemann for encouraging me, for the long meetings and their time. I am grateful to Dr. Kuhn for agreeing to supervise me at an unusually late stage.

Finally I thank my personal tutor, Prof. Murray Shanahan, for his fantastic support throughout my undergraduate studies.

Abstract

In a PERT network interdiction game one player tries to get a project, represented as a PERT graph, finished as quickly as possible by allocating resources, expediting the project's tasks and choosing between different technologies to use. At the same time a second player aims to maximally delay the project's completion date by carefully choosing which tasks to disrupt.

We develop a novel continuous time model of the game, showing that it can be solved as an MDP. We then implement an efficient exact dynamic programming algorithm for it and propose extensions involving implementation uncertainty and crashing with renewable resources. In addition, we demonstrate that approximate techniques for solving our models can alleviate the 'the curse of dimensionality'.

Contents

Nomenclature

Roman Symbols

$\exp(x)$	Another way of expressing e^x
$\mathbb{1}_{[P(x)]}$	The indicator function
inf	Infimum (greatest lower bound)
sup	Supremum (lowest upper bound)
e	Vector where every entry is 1
$x \sim \exp(\lambda)$	x is exponentially distributed with intensity λ
ADP	Approximate dynamic program
CPM/PERT	Critical path method/ Project evaluation and review technique
LP	Linear program
MDP	Markov decision process
MILP	Mixed integer linear program
MP	A Markov process

Chapter 1

Introduction

Interdiction games are quantitative methods for either analyzing vulnerabilities or finding ways to disrupt computer and transportation networks Cormican et al. [1998]. They have also been used to combat illegal drug trafficking and smuggling of radioactive material across borders Morton et al. [2007]. In this thesis we consider a motivating example which is the interdiction of an adversary's project Brown et al. [2009]. It has direct applications in crime prevention and defence. For instance, an important part of the war on drugs is interdicting the harvesting and production of narcotics Shipani [2010]; other examples include development of biological or nuclear weapons.

The project interdiction game is between two players. There is a manager who schedules tasks and allocates resources to them, aiming to complete the project in the most timely way. The other player, the interdictor, seeks to optimally undermine that effort using his own limited resources.

Our aim is to build a dynamic model of this game that runs in continuous time and that accounts for uncertainty. Exisiting approaches assume that the project plays out in a known deterministic way using the PERT (project review and evaluation technique) framework. In practice, however, projects are notorious for never going according to plan. We can introduce uncertainty by combining the previous method, probability theory and using techniques from the field of stochastic optimization.

One of the main challenges of this thesis is not only to propose a reasonable model that is also amenable to analysis, but to also be able to efficiently solve non-trivial problem instances on a computer. The latter is not always achievable because problems involving uncertainty and sequential decision making are generally intractable and suffer from a phenomenon that the scientist, Richard Bellman, famously called 'the curse of dimensionality'.

We also consider this thesis as an investigation into the use of exact and approximate dynamic programming to solve problems related to PERT projects.

1.1 Contributions

- We develop a new model for interdicting a project under uncertainty with random task durations using a Markov process representation.
- We implement an efficient and specialized dynamic programming algorithm to solve the new model.
- We extend this new model to allow for the project manager to expedite tasks and make interdiction success probabilistic. Also we suggest a way to solve the problem with discounted costs.
- We investigate approximate dynamic programming methods and propose some for solving our problem.
- We derive an MILP for solving the deterministic PERT interdiction problem with multiple technologies and demonstrate its computational feasibility.

1.2 Report structure

Chapter 2 contains the core background. We start by giving an overview of project management and PERT methods. Then we give a literature survey of interdiction games. Finally we define what an MDP is and describe some approaches to solving them.

Chapter 3 is about the derivation of a general deterministic interdiction game that can be used on flow networks and shortest path problems. Then we show an application to maximally delay an adversary's project. At the end we extend it to projects where there is a choice for the manager on how to complete some milestones.

In Chapter 4, we develop a model for playing the interdiction game in continuous time and with random task durations. We start with the simplest version where the project manager cannot crash tasks. Then we show that this can be solved with a discrete-time MDP and give an efficient algorithm for doing so. The chapter is concluded with various extensions including task crashing, implementation uncertainty and discounted costs.

Chapter 5 covers approximate dynamic programming. We describe our choice of approximation architecture and list some popular algorithms including a recent one.

In Chapter 6, we evaluate the methods given in Chapters 3 through to 5 using numerical experiments and finally give our conclusion and suggestions for future work.

Chapter 2

Background

2.1 CPM/PERT

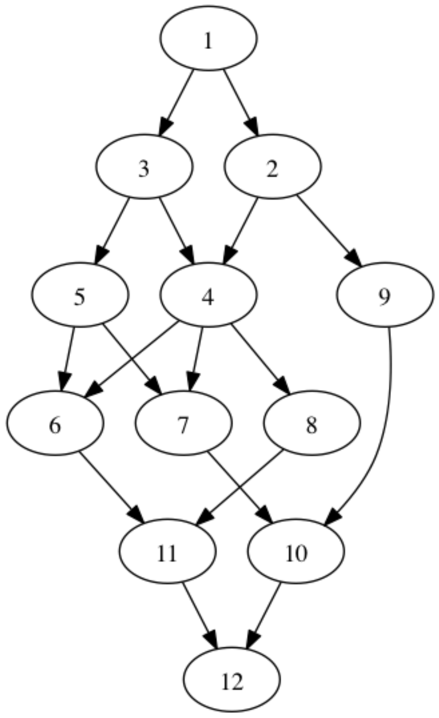
A project is a coordinated effort to achieve some goals. It consists of time and resource-consuming tasks (or activities) that need to be completed. An activity is an indivisible and non-repeatable unit of work Moder and Phillips [1964]. For example in a construction project the tasks can include excavation, bricklaying or interior plumbing. Some tasks depend on others, for example, putting up a wall requires first laying the foundation.

Success in large, critical projects is contingent on careful planning. It is vital, for example, to produce reliable estimates on the expected completion date of the project (we will call this the makespan) and the probability that deadlines are met. A project manager also needs to know when to schedule activities, which activities will delay the project if they take longer than planned and how much delay can be tolerated. One must decide on resource allocation and consider trade-offs between time and costs.

In the late 1950s the CPM and PERT techniques were developed independently by Kelley and Walker [1959] and the US Navy to address these questions. These techniques were only truly different at the time they began. Both have over time been extended, revised and are now so similar that their names have been amalgamated into CPM/PERT (we will use the term PERT throughout). Their techniques were notably used for the Polaris nuclear submarine program.

PERT represents projects as a graph. There are two ways: the AOA (activity-on-arc) or AON (activity-on-node) representation. We adopt the *AON* convention. A project is represented by a directed acyclic graph $G = (V, E)$, a network, whose vertices $V = \{1, ..., n\}$ correspond to the projects tasks and whose edges $E \subseteq V \times V$ encode the

Figure 2.1: A PERT network with 10 tasks. 1 and 12 are dummy tasks.



immediate technological precedences among the tasks. The precedence relation induces a strict partial order $\prec$ on V , where $u \prec v$ exactly if there exists a nontrivial directed path from u to v. Without loss of generality we assume that 1 is the unique source and n is the unique sink node of the graph, that is, $1 \prec u \prec n$ for all $u \in V$. This can always be enforced by appending dummy tasks to the project. We let $V^+(u), V^-(u) \subseteq E$ denote the outgoing, respectively incoming edge sets of u.

E can be divided into $E_{FS}, E_{FF}, E_{SS} \subseteq E$ and $E_{SF} \subseteq E$ that form a partition of E. These relations represents finish-start, finish-finish, start-start and start-finish *types* of relationships between activities.

- The most common is the finish-start relationship where $(u, v) \in E_{FS}$ iff v can only start when u has finished. The reason for this is usually resource or technological limitations. For example, it is impossible to cast metal without having aqcuired a metallurgical furnace.

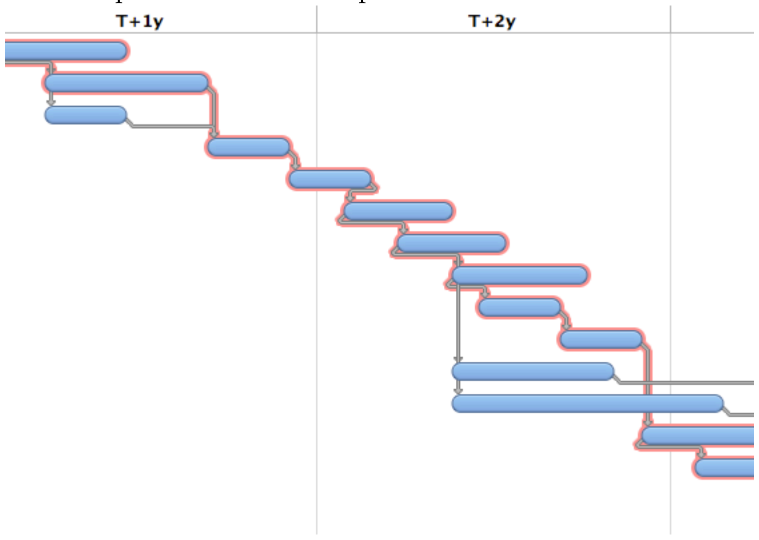
- A finish-finish relation between $(u, v) \in E_{FF}$ means that v can only finish after u has finished. The meaning of start-start is similar.

- The last relationship, start-finish is rarely used. In fact, it is not encountered anywhere in the case study that we use.

There is another way to create even more elaborate precedence relationships by introducing a measure called *lag*. If a task can start k weeks before another finishes then we say that it has a lag of $-k$ weeks. Lag occurs in practice where careful coordination can allow one task to begin prematurely before its prerequisite has completely finished. For example if a wall is partially built, it is possbile in theory to start painting some of it. It is equally possible to have a positive lag which means that some activity must wait some extra time after one of its predecessors has completed.

While depicting a project as a ordinary network graph is widely accepted, some project managers and other professionals prefer a project management specific visual representation. In this we can see which tasks are active at which points in time. It is called a Gantt chart.

Figure 2.2: Part of the Gantt chart for a project designed in OmniPlan. The activities highlighted in red are part of the critical path.



The questions of earliest expected project completion time and task scheduling can be answered simultaneously with a single algorithm. This algorithm computes the *critical path*. A critical path is a sequence of activities (an $1 \rightarrow \ldots \rightarrow n$ path in G) where a delay in any one of the activities will prolong the makespan. Therefore, the length of the critical path, that is the sum of the durations of the activities that lie on it, equals the estimated completion date. It is possible to have more than one critical path. The algorithm is covered in Hillier and Lieberman [1967] and we will briefly explain it here. Let s_v^e, s_v^l, f_v^e and f_v^l be the earliest start, latest start, earliest finish and latest finish times of $v \in V$. We initialize $s_1^e = s_1^l = f_1^e = f_1^l = 0$. Then we proceed breadth-first through G and for each v we calculate the following updating equations

$$s_v^e := \max_{u \in V^-(v)} f_u^e \tag{2.1}$$

$$f_v^e := d_v + s_v^e \tag{2.2}$$

where d_v is the task duration. Once n is reached the makespan is s_n^l. Then we set $f_n^l = s_n^l = s_n^e$ and go backwards in the same recursive fashion calculating

$$f_v^l := \min_{u \in V^+(v)} s_u^l \tag{2.3}$$

$$s_v^l := f_v^l - d_v \tag{2.4}$$

To find out the *slack* in an activity (the amount it may be delayed without delaying the whole project) , we subtract $s_v^l - s_v^e$. The slack is zero iff the task is on the critical path.

PERT is also used to compute probabilistic estimates on the time to complete a project. These techniques are very rough and make some assumptions: Task durations are independent random variables, they follow a Beta distribution, the project length is Gaussian distributed and the mean critical path will be always be the critical path. Each activity has three parameters, an optimistic duration estimate o, the pessimistic p and the most likely m. These quantities are used to work out the mean and variance of an activity's duration as follows:

$$\mu = \frac{o + 4m + p}{6} \tag{2.5}$$

$$\sigma^2 = \left(\frac{p - o}{6}\right)^2 \tag{2.6}$$

The mean and the variance of the project duration is then the sum of the means and variances of activities on the critical path. To work out the probability of the duration being less than some deadline one can, using the assumptions given, use the Gaussian CDF. The central limit theorem provides some justification of the Gaussian distribution assumption if there are many activities on the critical path.

2.2 Interdiction problems

An interdiction problem aims to find the optimal way to undermine an opponent as he aims to accomplish some goals. The ones which have been of particular interest focus on the interdiction of an opponent's operation of a network. These problems are useful not only in a militaristic setting against an enemy but also to find vulnerabilities in one's own network.

In this dissertation we focus on the interdiction of a project, which is something that was proposed recently Brown et al. [2005, 2009] but the ones that have been studied the most and date back to the 1960s Wollmer [1964] are those involving maximum network flows Cormican et al. [1998]; Wood [1993] and shortest path problems Israeli and Wood [2002]. More recently other problems have been modelled such as smuggling and intrusion Morton et al. [2007] and the multi-commodity flow problem Lim and Smith [2006].

2.2.1 Stochastic network interdiction

This is a variant on a deterministic problem where the objective is to minimize the maximum flow from source s to sink t through a capacitated network by attacking edges and thereby reducing their capacity or deleting them. In this problem the interdiction attempts are independent binary random variables and the edge capacities may also be random. An interdiction is a binary random variable because it may be successful or partially successful (or completely unsuccessful).

Instead of formulating it as a straightforward large deterministic MILP, which would have an exponential number of scenarios, Cormican and Wood found a sequential approximation algorithm which efficiently computes upper and lower bounds on the optimal value at each iteration. Their approach is based on iteratively splitting (which they call refining) partitions of the support of the binary random variables. They then condition the expectation of the second stage problem on each of the subspaces from the partition. Because the second stage recourse problem is a concave function, they used Jensen's inequality for the upper bound. The tricky part was to find a lower bound. They managed this by reformulating the recourse problem by moving the term with the first stage interdiction variables to the objective and thus obtain an equivalent convex minimization problem. The minimization subproblem is itself solved using Benders decomposition as there may be a large number of scenarios. The solution to the minimization lower bound problem is then used to compute the upper bound. Afterwards the partition is refined and the steps repeated until the difference between the lower and upper bounds is sufficiently small.

The worst case complexity of their algorithm is exponential because in the worst case, the entire space of interdiction successes/failures would need to be enumerated. However empirically they demonstrated that the partition set does not grow too large before a good enough approximation is reached.

Cormican and Wood were also able to extend this approach to problems where multiple interdictions were possible and the capacities were discrete random variables.

2.2.2 Smuggling problems

The smuggling model was described by Morton et al. [2007]. It is another example of a two stage stochastic interdiction game. In this one, the interdictor wants to minimize the probability that a smuggler gets from s to t in a transportation network without getting caught. He does so by installing sensors on certain edges so that there is a lower probability of the smuggler traversing each of them successfully. It is stochastic because the s and t nodes are given by a probability mass function over a set of scenarios. The interdictor can install a limited number of such sensors and only on a strict subset of the edges of the network. A simplified version of the problem considers a bipartite network where the edges from one partition to the other can represent border crossings of a country. It is only on these border crossings that sensors may be placed.

Morton et al. had also formulated a version of the problem where the interdictor and proliferator had different perceptions of the network. They agreed on the topologies but might have different views of the probability of crossing an edge undetected. The smuggler may also not be aware of the interdictor's exact sensor placements. They solved the easier problem as a direct large scale deterministic MILP. The second problem was harder and they used a custom branch and bound algorithm.

2.2.3 PERT interdiction

Brown et al. [2009] propose a Benders decomposition style algorithm to solve the PERT interdiction problem that we examine later. The algorithm alternates between an upper bound given by the master problem and a lower bound from a subproblem. To ensure convergence they add cuts to bound the maximum project time for a given proliferator's interdiction and expediting plan. They also overcome an issue where the interdiction plans might repeat themselves and force the proliferator to use a certain decision plan by introducing what they call SECs (solution elimination constraints).

The algorithm iteratively solves many MILPs and while it is guaranteed to converge there is a maximum number of iterations that can be specified if ϵ optimality is not reached by then.

2.3 Markov decision process

The Markov decision process (MDP) Bellman [1957]; Puterman [1994] is a model for solving sequential decision making under uncertainty problems. It is an extension of a Markov Chain where transitions probabilities are determined through actions taken by a decision maker. Built in are rewards (or costs) to motivate the agent influencing the process. A Markov decision problem can then be stated as the optimization of some objective criterion such as the average or total reward accumlated over a time horizon by the agent. MDPs are an attractive tool because of their versatility. Unfortunately their practical use has been limited due to their exponential complexity (one of the main challenges of this project).

The MDP is a 4-tuple. $\langle \Omega, \mathcal{A}, p.(.,.), r.(.) \rangle$ where Ω is the state/sample space, $\mathcal{A}$ the action space, $p_a(x, y)$ the transition probability function and $r_a(x)$ the reward function. The set of actions admissable in x is written as A_x.

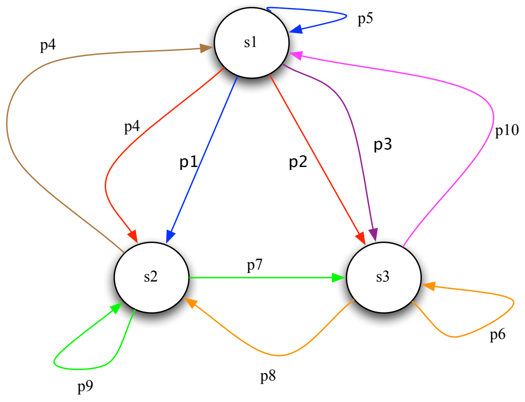
A function $\pi : \Omega \to \mathcal{A}$ is called a policy and is used to direct the agent on which action to take in a given state. $p_a(x, y)$ and $r_a(x)$ are then functions of the chosen action a. We sometimes write $r_\pi(x)$ and $p_\pi(x, y)$ which are shorthands for $r_{\pi(x)}(x)$ and $p_{\pi(x)}(x, y)$. In general π can be defined as $\pi : \Omega \times \mathcal{A} \to [0, 1]$, the *probability* of taking an action in a state. The latter type of policies are called *randomized*. An MDP is called *stationary* if its transitions and rewards are time and history independent. The MDPs encountered in this report are all either stationary or randomized stationary. The final parameter of an MDP model is the *time horizon* H which is how long the process runs for. If infinite, the rewards are usually discounted by a factor $\gamma \in (0, 1)$ which prevents unbounded objective values (as we see next).

The MDP problem is usually

$$\underset{\pi \in \Pi}{\text{maximize}} \quad \mathbb{E}^\pi \left(\sum_{t=0}^{H} \gamma^t r_\pi(x_t) \mid x_0 \sim p_0 \right) \tag{2.7}$$

where Π is a space of admissable policies, $\mathbb{E}^\pi$ is the expectation conditioned on the Markov Chain generated by π, x_t is the random state at time t and p_0 is the probability distribution of starting states.

Figure 2.3: Markov decision process. Colors represent actions. Lines of the same color denote transitions that result from choosing the action of that color



It can be shown that (2.7) is solved with the following equations for every state

$$V(x) = \max_{a \in A_x} \left\{ r_a(x) + \gamma \sum_{y \in} p_a(x,y) V(y) \right\} \tag{2.8}$$

Where $V(x)$ can be thought of as the expected *value* of being in state x. Sometimes we use costs instead of rewards; the 'negative' of $r_a(x)$ becomes $g_a(x)$ and the value is called the cost-to-go, written as $J(x)$. The value for a fixed policy is denoted $V^\pi(x)$.

Using $V(.)$, the optimal stationary policy is found from:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A_x} \left\{ r_a(x) + \gamma \sum_{y \in} p_a(x,y) V(y) \right\} \tag{2.9}$$

2.3.1 MDP LP solution

(2.8) can be solved as an LP:

$$\begin{aligned} \underset{v}{\text{minimize}} \quad & \sum_{x \in \Omega} v_x \\ \text{subject to} \quad & v_x \geq r_a(x) + \gamma \sum_{y \in \Omega} p_a(x,y) v_y \quad x \in \Omega, a \in A_x \end{aligned} \tag{2.10}$$

At optimality, a subset of the constraints will be satisfied as equalities - those corresponding to optimal actions. Once the optimal solution v^* is found, the policy can be extracted from (2.9) where entries of v give the expected total cost function for each state. The dual LP below will be used later

$$\begin{aligned}
&\underset{\theta}{\text{maximize}} && \sum_{x\in\Omega}\sum_{a\in A_x} r_a(x)\theta_{xa} \\
&\text{subject to} && \sum_{a\in A_x}\theta_{xa} - \gamma\sum_{y\in\Omega}\sum_{a\in A_y} p_a(x,y)\theta_{ya} = p_0(x) \quad \forall x\in\Omega \\
& && \theta \geq 0
\end{aligned} \tag{2.11}$$

where $p_0(.)$ is the probability mass function of the starting state. With it we can add constraints based on costs of performing actions. The variables θ_{xa} also give us a measure of the expected fraction of time spent performing a in x. So a policy can be calculated from

$$\pi^*(x,a) = \frac{\theta^*_{xa}}{\sum_{b\in A_x}\theta^*_{xb}} \tag{2.12}$$

where $\pi^*(x,a)$ is the probability of choosing action a in x. In the absense of additional constraints on (2.11) the value of $\pi^*(x,a) = 1$ or 0 for all x and a which means that the policy is deterministic.

2.3.2 Dynamic programming algorithms for MDP - Policy and value iteration

Dynamic programming (DP) is a type of algorithm that solves a problem having *optimal substructure* and *overlapping subproblems*. Optimal substructure exists when the whole problem can be solved by dividing it into smaller *sub-problems*, that have the same structure, and solving those sub-problems recursively. This is true for MDPs and can be seen from (2.8). A problem is said to have overlapping subproblems if it is possible that at least two sub-problems have some dependencies in common (dependencies in terms of solutions to other sub-problems). The idea of DP is to store every solution to a sub-problem so that they need not be recomputed if needed again. In the case of MDPs, the dynamic programming algorithms store $V(x)$. Notice that the algorithm to get the critical path of a PERT network, described in section 2.1, is precisely a dynamic programming algorithm.

Policy iteration starts with an some initial candidate policy π_0. It then iteratively improves the policy until convergence. Improvement is done by calculting the value function under the current policy and then obtain a new policy using (2.8). Because the last equation can be shown to be a contraction mapping the algorithm terminates.

Value iteration is another algorithm. It works by iterating through all states and updating their expected values $V(x)$ using (2.8). It then repeats this step until V

converges to V^*. The optimal policy is then computed from (2.9) and V^*.

Algorithm 1: Value iteration

```
Input: MDP $M = \langle \Omega, \mathcal{A}, p.(.,.), r.(.) \rangle$
Output: Value function $V(x)$
repeat
    $V := V'$
    foreach $x \in \Omega$ do
        $V'(x) := \max_{a \in A_x} \{r_a(x) + \gamma \sum_{y \in \Omega} p_a(x,y) V(y)\}$
    end
until $||V - V'|| \leq \epsilon$
return $V$
```

Algorithm 2: Policy iteration

```
Input: MDP $M = \langle \Omega, \mathcal{A}, p.(.,.), r.(.) \rangle$, Initial candidate $\pi_0$
Output: Optimal policy $\pi^*$
$\pi' := \pi_0$
repeat
    $\pi := \pi'$
    Solve the following system of equations for $V^\pi$ :
    $V^\pi(x) = r_\pi(x) + \gamma \sum_{y \in \Omega} p_\pi(x,y) V^\pi(y),\ x \in \Omega$
    foreach $x \in \Omega$ do
        $\pi'(x) := \text{argmax}_{a \in A_x} \{r_a(x) + \gamma \sum_{y \in \Omega} p_a(x,y) V^\pi(y)\}$
    end
until $\pi' = \pi$
return $\pi$
```

Chapter 3

Deterministic interdiction games

In this chapter we state the general deterministic interdiction game. Then we show that it can be solved as a MILP. We demonstrate an application of the framework for deciding on a strategy to maximally delay a project. Currently, it is solved with an iterative decomposition algorithm which is slow to converge. We show how to solve this problem directly and exactly.

3.1 A framework for deterministic interdiction games

We formulate a certain type of interdiction game as a tractable optimization problem with binary decision variables for the interdictor. There are situations where it is possible use binary variables for the follower as we will see later on.

In the following derivation, we treat the problem as a Stackelberg game which occurs in two steps. The leader (interdictor) makes a decision and the follower responds to it. We solve the problem:

$$\sup_{x\in X} \inf_{y\in\Delta(x)} f(x,y) \tag{3.1}$$

where

$$X = \{x = (x_B, x_C) \in \{0,1\}^{n_b} \times \mathbb{R}^{n_c} \mid Ax \leq b\} \tag{3.2}$$

for some coefficient matrix $A \in \mathbb{R}^{m\times(n_c+n_b)}$ and $b \in \mathbb{R}^m$. The follower's feasible set is a polytope parameterized by x_B and is represented as

$$\Delta(x) = \left\{ y \in \mathbb{R}^n_+ \;\middle|\; x_B^\top G_i y + b_i^\top y \geq h_i^\top x_B + v_i \text{ for } i = 1, ..., p \right\} \tag{3.3}$$

Where $G_i \in \mathbb{R}^{n_B\times n}$, $b_i \in \mathbb{R}^n$, $h_i \in \mathbb{R}^{n_B}$ for $i = 1, ..., p$. The h_i are rows of $H \in \mathbb{R}^{p\times n_B}$;

columns of H are denoted as h^i.

The follower's objective function $f : \{0,1\}^{n_b} \times \mathbb{R}^{n_c} \times {\mathbb{R}_+}^n \to \mathbb{R}$ is defined as.

$$f(x, y) = c^\top x + x^\top Dy + d^\top y \tag{3.4}$$

(3.1) is equivalent to

$$\sup_{x\in X} \{\tau \in \mathbb{R}^n \mid \tau \le f(x, y),\ \forall y \in \Delta(x)\} \tag{3.5}$$

(3.5) is a semi-infinite problem Lopez and Still [2011] because it has a finite number of variables but an infinite number of constraints. An equivalent problem with a finite set of constraints is found by making $\tau \le f(x, y)$ hold for the worst possible choice of y, namely, the minimization of f with respect to it Bertsimas and Sim [2002].

$$\sup_{x\in X} \left\{ \tau \in \mathbb{R}^n \;\middle|\; \tau \le \inf_{y\in\Delta(x)} f(x, y) \right\} \tag{3.6}$$

We then substitute in the dual problem of the minimization term.

$$\sup_{x\in X, z\in\Phi(x)} \{\tau \in \mathbb{R}^n \mid \tau \le g(x, z)\} \tag{3.7}$$

Where g is the dual objective and $\Phi(x)$, the dual feasible set. We can expand the second stage problem from (3.1) to obtain.

$$\inf_{y\in\mathbb{R}^n_+} \left\{ x^\top Dy + d^\top y \mid x_B^\top G_i y + b_i^\top y \ge h_i^\top x_B + v_i, \text{ for } i = 1, .., p \right\} \tag{3.8}$$

Note that the $c^\top x$ term is constant with respect to y so we have dropped it for now. We can get the same optimal objective value later by adding it back. Since this problem is parameterized by x, it is linear in y. Therefore, we find the dual LP directly.

$$\sup_{u\in\mathbb{R}^p_+} \left\{ u^\top H^\top x_B + u^\top v \mid u^\top G_i^\top x_B + u^\top b^i \le (d^i)^\top x + d_i, \text{ for } i = 1, .., n \right\} \tag{3.9}$$

However, it is not possible to substitute (3.9) into (3.6) directly because there are bilinear terms in u and x_B in both the constraint and objective. Doing so will give non-convex constraints.

First, we simplify the problem. It is possible to convert the objective into a constraint by changing the problem into one with an additional variable and constraint.

For this reason we will, replace the objective with some linear function $f^\top u$ where f is a constant.

$$\sup_{u\in\mathbb{R}^p_+} \left\{ f^\top u \mid u^\top G_i^\top x_B + u^\top b^i \le (d^i)^\top x + d_i, \text{ for } i = 1,..,n \right\} \tag{3.10}$$

Here is a formulation of the problem using $p + n \times n_B = \mathcal{O}(n^2)$ variables and $n \times (n_B + 1) = \mathcal{O}(n^2)$ constraints. In this, let $\alpha^+ = (\alpha_1^+, \dots, \alpha_n^+), \alpha^- = (\alpha_1^-, \dots, \alpha_n^-) \in \mathbb{R}_+^{n_B\times n}$. Also we define $G_i^+ := (\max\{g_{jk}^i, 0\})_{jk}$ and likewise $G_i^- := (\min\{g_{jk}^i, 0\})_{jk}$. That is, G_i^+, G_i^- are the non-negative, respectively non-positive elements of G_i. One can check that $G^+ + G^- = G_i$. The following is an LP and when substituted back into the original problem (3.6) is a MILP. I will prove its equivalence to (3.10).

$$\begin{aligned}
\underset{u\in\mathbb{R}^p_+}{\text{maximize}} \quad & f^\top u && (P2)\\
\text{subject to} \quad & e^\top \left(\alpha_i^+ - \alpha_i^-\right) + u^\top b^i \le (d^i)^\top x + d_i && i = 1, ..., n\\
& \alpha_i^+ \ge G_i^+ u - M(e - x_B) && i = 1, ..., n\\
& \alpha_k^- \le M x_B && i = 1, ..., n\\
& \alpha_i^- \le -G_i^- u && i = 1, ..., n\\
& \alpha^+, \alpha^- \in \mathbb{R}_+^{n_B\times n}
\end{aligned}$$

Proposition 1. (3.10) *and P2 are equivalent.*

Proof. We show that their feasible sets are equal. Suppose (u, α^+, α^-) is a FS in P2, I show that the constraints in (3.10) of the form $u^\top G_i^\top x_B + u^\top b^i \le (d^i)^\top x + d_i$ are true for any i. For any k and i $\alpha_{ik}^+ \ge x_k(G_i^+ u)_k$ and $\alpha_{ik}^- \le -(G_i^- u)_k x_k$. Using this it follows that:

$$\begin{aligned}
u^\top G_i^\top x_B + u^\top b^i &= \sum_{i=1}^{n_B} x_k((G_i^+ + G_i^-)u)_k + u^\top b^i\\
&= \sum_{k=1}^{n_B} x_k(G_i^+ u)_k + \sum_{k=1}^{n_B} x_k(G_i^- u)_k + u^\top b^i\\
&\le \sum_{k=1}^{n_B} \alpha_{ik}^+ + \sum_{k=1}^{n_B}(-\alpha_{ik}^-) + u^\top b^i\\
&= e^\top \alpha_i^+ - e^\top \alpha_i^- + u^\top b^i\\
&\le (d^i)^\top x + d_i
\end{aligned}$$

Conversely suppose that u is a FS in (3.10). We see that for any i, there exist $\alpha_i^+, \alpha_i^- \in \mathbb{R}_+^{n_B}$ such that, (1) $e^\top \alpha_i^+ - e^\top \alpha_i^- + u^\top b^i \leq (d^i)^\top x + d_i$ and (2), $\alpha_i^+ \geq G_i u - M(e - x)$ and (3), $\alpha_i^- \leq \min\{Mx_B, -G_i^- u\}$. Let's make $\alpha_{ik}^+ = x_k (G_i^+ u)_k$ and $\alpha_{ik}^- = -x_k (G_i^- u)_k$. To check (2) we see that $\alpha_{ik}^+ = x_k (G_i^+ u)_k \geq (G_i u)_k - M(1 - x_k)$. Statement (3) can be verified by by checking the two cases cases $x_k = 0$ and $x_k = 1$. To prove (1):

$$
\begin{aligned}
e^\top \left(\alpha_i^+ - \alpha_i^-\right) + u^\top b^i &= \sum_{k=1}^{n_B} \alpha_{ik}^+ + \sum_{k=1}^{n_B} (-\alpha_{ik}^-) + u^\top b^i \\
&= \sum_{k=1}^{n_B} x_k (G_i^+ u)_k + \sum_{k=1}^{n_B} x_k (G_i^- u)_k + u^\top b^i \\
&= \sum_{k=1}^{n_B} x_k (G_i u)_k + u^\top b^i \\
&\leq (d^i)^\top x + d_i
\end{aligned}
$$

The objective functions are identical in both problems and therefore, because the feasible sets for u are equal (as shown), the optimal solutions and objective values are too. □

Now we can state the final MILP by making x a decision variable and adding back the $c^\top x$ term to the objective function.

(3.6).

$$
\begin{array}{lll}
\underset{x \in X, u \in \mathbb{R}_+^p}{\text{maximize}} & c^\top x + f^\top u & (P2) \\
\text{subject to} & e^\top \left(\alpha_i^+ - \alpha_i^-\right) + u^\top b^i \leq (d^i)^\top x + d_i & i = 1, ..., n \\
& \alpha_i^+ \geq G_i^+ u - M(e - x_B) & i = 1, ..., n \\
& \alpha_k^- \leq M x_B & i = 1, ..., n \\
& \alpha_i^- \leq -G_i^- u & i = 1, ..., n \\
& \alpha^+, \alpha^- \in \mathbb{R}_+^{n_B \times n} &
\end{array}
$$

3.2 Project interdiction game

In this section we will derive the MILP to optimally delay an adversary's project based on a model by Brown et al. [2009]. We use the project representation from Chapter 2.

In this game, a project is *planned* by two players. The follower schedules the start times of tasks s_v and also allocates resources to *minimize* the project's makespan.

The second player chooses which tasks to delay to do the opposite i.e. *maximize* the makespan.

For each task we associate a duration d_v and make $d_1 = d_n = 0$. There are a set R of non-renewable types of resources. Each activity needs c_{vr} units of resource r. Tasks may be completed faster i.e. in less time than d_v by *crashing* them (allocating extra resources to them). There is a separate cost, a_{vr}, for expediting tasks. The follower's decision variables are $(s_v)_{v \in V}$ the start times, $(e_v)_{v \in V}$ the crashing amounts for each task and his budget for each resource $r \in R$ is b_r.

The interdictor is faced with costs of delaying a task $h_{vr'}$ in terms of interdiction resource $r' \in R'$. Interdicting a task delays it by an amount δ_v. For every interdiction resource r' there is a budget $b_{r'}$ and the follower's decision variables are x_v for either yes=1/no=0 decisions on delaying that task.

Table 3.1: Data for the interdiction problem Brown et al. [2009]

Symbol	**Explanation**
V	The set of tasks
$1, n$	Artificial start and end vertices
E	Precedence relation
R	Non-renewable proliferator resources (e.g. monetary, physical etc).
R'	Non-renewable interdictor resources (e.g. diplomatic)
d_v	The nominal duration of task v in weeks
$\underline{d}_i$	The minimum expedited duration of i in weeks
f'_{uv}	Equal to 1 if $(u, v) \in E_{FF} \cup E_{FS}$ and 0 otherwise
f''_{uv}	Equal to 1 if $(i, j) \in E_{FF} \cup E_{SF}$ and 0 otherwise
c_{vr}	Units of $r \in R$ required to complete i with r
a_{vr}	Per-week cost to expedite i with additional units of r
l_{uv}	Constant lag or lead times between pairs of tasks
δ_i	Delay from interdicting i
$h_{vr'}$	Interdiction cost in r'-dollars
b_r	Proliferator's budget for resource r
$b_{r'}$	Interdictor's budget for resource r'

Table 3.2: Decision variables from Brown et al. [2009]

Symbol	Explanation
s_v	The earliest start time of v
e_v	Amount v is expedited in weeks
x_v	1 if v is interdicted, 0 otherwise
θ_v	1 if v is completed, 0 otherwise [1]

3.2.1 Fixed decision plan for the project manager

To derive the formulation, we first consider a simpler sub-problem. Assume there is one technology. Consider the second stage problem. At this point the leader has made a choice of which tasks to interdict, which the follower has complete knowledge of. So x_v are constant and the vectors of earliest start times s and expediting amounts e are decision variables.

$$\begin{array}{lll}
\text{minimize} & s_n & (PMIN) \\
\text{subject to} & s_v - s_u + f'_{uv}e_u - f''_{uv}e_v \geq f'_{uv}\,(d_u + \delta_u x_u) & (u,v) \in E \\
 & -\, f''_{uv}(d_v + \delta_u x_u) + l_{uv} & \\
 & e_u \leq d_u - \underline{d}_u & u \in V \\
 & \displaystyle\sum_{u\in V} c_{ur} + \sum_{u\in V} a_{ur}e_u \leq b_r & r \in R \\
 & s \geq 0, e \geq 0 &
\end{array}$$

The first constraint ensures that no task can start before all of its predecessors' finish times. The second constraint places an upper limit on how much each task can be expedited and the third is a budget constraint. PMIN is a linear program and its dual

[1] θ_v is assumed to always be 1 for now

is shown below.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{(u,v)\in E} y_{uv}\left(f'_{uv}\left(d_u + \delta_u x_u\right) - f''_{uv}(d_v + \delta_v x_v) + l_{uv}\right) && (LONP)\\
& - \sum_{u\in V} z_u(d_u - \underline{d_u}) - \sum_{r\in R} w_r \left(b_r - \sum_{u\in V} c_{ur}\right) \\
\text{subject to} \quad & \sum_{v\in V^-(u)} y_{vu} - \sum_{v\in S(u)} y_{uv} \leq \begin{cases} 1 & \text{if } u = 1 \\ -1 & \text{if } u = n \\ 0 & \text{otherwise} \end{cases} && u \in V \\
& \sum_{v\in V^+(u)} f'_{uv} y_{uv} - \sum_{v\in V^-(u)} f''_{uv} y_{vu} \\
& - \sum_{r\in R} a_{ur} w_r - z_u \leq 0 && u \in V \\
& y \geq 0, z \geq 0, w \geq 0
\end{aligned}
$$

Note how the dual is a longest path problem where $y_{uv} > 0$ if edge (u, v) is in the path and 0 otherwise. The first constraint enforces conservation in a flow network where the capacity of each edge is 1. The dual variables y correspond to the prececedence constraint of each edge, v the expediting limit constraints of the primal problem and w, the resource constraints. For the reasons given we call this LONP (longest path).

Using the dual problem (3.2.1) and the linearization technique shown earlier, we find that the single-technology project interdiction game can be solved with this MILP

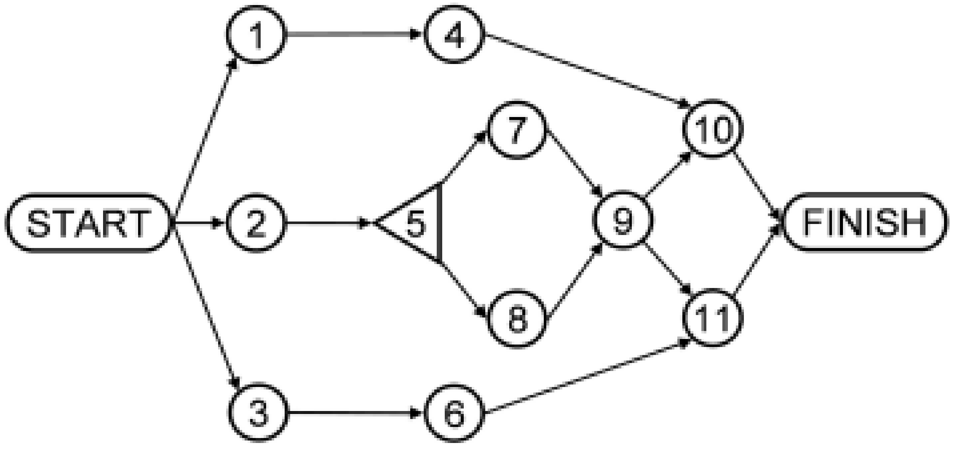
$$
\begin{aligned}
\text{maximize}\quad & \tau && (FPINT)\\
\text{subject to}\quad & \tau \leq \sum_{(u,v)\in E} y_{uv}(f'_{uv}d_u - f''_{uv}d_v + l_{uv}) \\
& - \sum_{u\in V} z_v(d_u - \underline{d_u}) - \sum_{r\in R} w_r b_r + \sum_{u\in V}(\alpha_u^+ - \alpha_u^-) \\
& \sum_{v\in V^-(u)} y_{vu} - \sum_{v\in V^+(u)} y_{uv} \leq \begin{cases} 1 & \text{if } u = 1 \\ -1 & \text{if } u = n \\ 0 & \text{otherwise} \end{cases} && u \in V \\
& \sum_{v\in V^+(u)} f'_{uv} y_{uv} - \sum_{v\in V^-(u)} f''_{uv} y_{vu} \\
& - \sum_{r\in R} a_{ur} w_r - z_u \leq 0 && u \in V \\
& \alpha_u^+ \leq \delta_u \sum_{v\in V^+(u)} f'_{uv} y_{uv} && u \in V \\
& \alpha_u^+ \leq M x_u && u \in V \\
& \alpha_u^- \geq \delta_u \left(\sum_{j\in V^-(u)} f''_{vu} y_{vu} \right) - M(1 - x_u) && u \in V \\
& \sum_{u\in V} h_{ur'} x_u \leq b_{r'} && r' \in R' \\
& y \geq 0, z \geq 0, w \geq 0, \alpha^+ \geq 0, \alpha^- \geq 0, x \in \{0,1\}^n
\end{aligned}
$$

3.2.2 Project interdiction with a decision plan

Now we consider an extension to a *decision* PERT network Moder and Phillips [1964]. This is a representation of a project that allows for alternative ways of completing it. Essentially, the project is broken down into milestones. Between these milestones are choices of which tasks to finish in order to reach the next milestone. For example, in the nuclear proliferation case study there are different competing uranium enrichment technologies (where enrichment is treated as a milestone or sub-project).

We use the notation from Brown et al. [2009]. Let $D \subset V$ be the set of decision vertices and P_d the set of technology choices associated with decision vertex $d \in D$. Tasks are partioned into sets V_p for $p \in P_d$ and $V_0 := V \setminus \bigcup_{d\in D, p\in P_d} V_p$, so that $\left(\bigcup_{d\in D, p\in P_d} V_p\right) \cup V_0 = V$ and for every d and $p, q \in P_d$, $p \neq q$ $V_p \cap V_q = \emptyset$. The project

Figure 3.1: An example of a decision PERT network from Harney et al. [2006]. The triangular node represents the milestone or decision task. The project manager can choose between executing 7 or 8 while the remaining tasks are mandatory.



management decision problem from Brown et al. is

$$
\begin{array}{lll}
\text{minimize} & s_n & (DPMIN) \\
\text{subject to} & s_v - s_u + f'_{uv} e_u - f''_{uv} e_v \geq f'_{uv}\,(d_u + \delta_u x_u) & (u,v) \in E \\
& - f''(d_v + \delta_v x_v) + l_{uv} - M(1-\theta_u) & \\
& e_u \leq d_u - \underline{d}_u & u \in V \\
& \sum\limits_{u \in V} c_{ur} y_u + \sum\limits_{u \in V} a_{ur} e_u \leq b_r & r \in R \\
& \theta_u = 1 & u \in V_0 \\
& \theta_u \leq y_v & v \in V^+(u),\ u \in V_p \\
& \theta_d = \sum\limits_{v \in V^+(u)} \theta_v & d \in D \\
& s \geq 0, e \geq 0, y \in \{0,1\}^n, \theta \in \{0,1\}^n &
\end{array}
$$

We introduced binary variables θ. When $\theta_u = 0$ the task is not completed and the first constraint is relaxed, since this vertex is effectively eliminated. The other new constraints enforce that exactly one successor of decision vertex $d \in D$ is chosen, that once that successor is chosen, all its successors must be completed and finally all other tasks not associated with any technology are compulsory. DPMIN , unlike PMIN, is a 0-1 problem and it cannot be dualized in the same way. It is possible to relax DPMIN to an LP but this produces infeasible solutions and a very wide lower bound on the optimum for the proliferator. This therefore, when incorporated into the interdiction game gives overly conservative solutions for the interdictor.

Instead we exploit the nature of this specific problem. There are $\prod_{d \in D} |V^+(d)|$ possible values for θ (which is the number of different combinations of technologies) and in the case study from Harney et al. Harney et al. [2006] there are just 3.

Definition 1. *For any $\theta \in \{0,1\}^n$, we define $G_\theta = (V_\theta, E_\theta)$ as a subgraph of G where $V_\theta := \{u \in V \mid \theta_u = 1\}$ and $E_\theta := \{\ (u,v) \in E |\ \theta_u = \theta_v = 1\}$. The successors of $u \in V_\theta$ are denoted $V_\theta^+(u)$ and similarly the predecessors as $V_\theta^-(u)$.*

Proposition 1. *The following problem (called DLONP)*

$$\min_{\theta \in \Theta} g_x(\theta) \tag{3.11}$$

is equivalent to DPMIN where $g_x(\theta)$ is the optimal solution to LONP with $G = G_\theta$.

Proof. Let $v(P,G)$ be the optimal objective function value of problem P on a project network graph G and likewise $F(P,G)$ be its feasible set. Let $p^* := v(DPMIN, G)$ and θ^* be the optimal value. Then $p^* = v(PMIN, G_{\theta^*}) = v(LONP, G_{\theta^*})$ since PMIN is the problem for a fixed decision plan and strong duality holds between PMIN and LONP. Also by definition, $v(LONP, G_{\theta^*}) = g_x(\theta^*)$. Suppose that $g_x(\theta^*)$ is not optimal for DLONP. Then there exists θ' for which $g_x(\theta') < g_x(\theta^*)$ and therefore $g_x(\theta') = v(LONP, G_{\theta'}) = v(PMIN, G_{\theta'}) = s'_n$ for some $(s', e', \theta') \in F(DPMIN, G)$ and $s'_n < v(DPMIN, G)$. □

Finally, to solve the interdiction problem, we require

$$\sup_{x \in X} \inf_{\theta \in \Theta} g_x(\theta) = \sup_{x \in X} \{\tau \mid \tau \leq g_x(\theta)\ \theta \in \Theta\}$$

This translates to the following MILP:

$$
\begin{aligned}
\text{maximize}\quad & \tau && (DPINT)\\
\text{subject to}\quad & \tau \le \sum_{(u,v)\in E_\theta} y_{\theta uv}(f'_{uv}d_u - f''_{uv}d_v + l_{uv}) \\
& - \sum_{u\in V_\theta} v_{\theta u}(d_u - \underline{d_u}) - \sum_{r\in R} w_{\theta r}\left(b_r - \sum_{v\in V_\theta} c_{ur}\right) \\
& + \sum_{u\in V_\theta} (\alpha^+_{\theta u} - \alpha^-_{\theta u}) && \theta\in\Theta \\
& \sum_{v\in V^-_\theta(u)} y_{\theta vu} - \sum_{v\in V^+_\theta(u)} y_{\theta uv} \le \begin{cases} 1 & \text{if } u=1 \\ -1 & \text{if } u=n \\ 0 & \text{otherwise} \end{cases} && u\in V_\theta,\ \theta\in\Theta \\
& \sum_{v\in V^+_\theta(u)} f'_{uv}u_{\theta uv} - \sum_{v\in V^-_\theta(u)} f''_{uv}u_{yji} \\
& - \sum_{r\in R} a_{ur}w_{\theta r} - z_{\theta u} \le 0 && u\in V_\theta,\ \theta\in\Theta \\
& \alpha^+_{\theta u} \le \delta_u \sum_{v\in V^-_\theta(u)} f'_{uv}u_{\theta uv} && u\in V_\theta,\ \theta\in\Theta \\
& \alpha^+_{\theta u} \le M x_u && u\in V_\theta,\ \theta\in\Theta \\
& \alpha^-_{\theta u} \ge \delta_u \sum_{v\in V^-_\theta(u)} f''_{vu}y_{\theta vu} - M(1-x_u) && u\in V_\theta,\ \theta\in\Theta \\
& \sum_{u\in V} h_{ur'}x_u \le b_{r'} && r'\in R' \\
& y_\theta \ge 0, z_\theta \ge 0, w_\theta \ge 0, \alpha^+_\theta \ge 0, \alpha^-_\theta \ge 0 && \theta\in\Theta \\
& x \in \{0,1\}^n
\end{aligned}
$$

Remark 1. *Compared to FPINT, DPINT has at most $|\Theta|$ times the number of constraints and variables and $|\Theta| = \prod_{d\in D}|V^+(d)|$. In practice however $|\Theta|$ may be quite small which makes this tractable. FPINT itself has $\mathcal{O}(n)$ constraints and $\mathcal{O}(n^2)$ variables.*

Chapter 4

Interdiction games under uncertainty

In the last chapter we analyzed a project interdiction game where it was assumed that tasks have a fixed duration that are determined by decisions here and now. In real life this is not true in general.

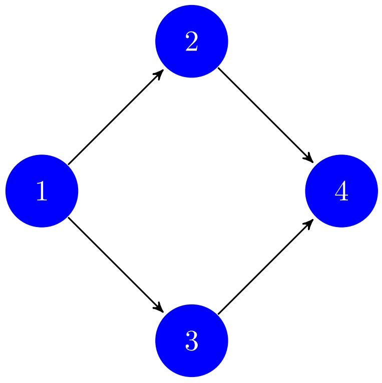
Now we formulate a problem where task durations are random and decisions to interdict are made over time. The goal is to find an optimal *policy* for which tasks to delay in different states of the project. This differs from DPINT, where a plan was computed for both sides before the project began. Instead of attempting to convert DPINT in its entirety, we first simplify it so that there is no crashing and only one technology. We also crucially assume that task processing times are *independent* and *memoryless*. The following example shows that the timing of interdictions is important and is something that our new model will take into account unlike the nominal one.

Example 1. *Consider the PERT network in Figure 4.1 and suppose the interdictor can only delay one task. The optimal choice according to DPINT would be to interdict* 3 *now. However because the durations are memoryless and independent, it would actually be better to wait until the one of them finishes and only delay whichever is left. So it could instead be that activity* 2 *is optimal to interdict (if* 3 *completes before it).*

4.1 Problem statements and assumptions

The input to our game is a directed acyclic graph $G = (V, E)$ and an integer interdiction budget $b \geq 0$, as before. We are also given n dimensional vectors μ and ν. For each task $v \in V$, μ_v is its duration's exponential distribution parameter (or rate) under normal

Figure 4.1: PERT network with 2 activities.



Task	Normal	Delayed
2	40	58
3	45	60

Table 4.1: Data for the PERT network in Figure 4.1. Each entry in the table is the mean task duration.

conditions and $\nu_v \leq \mu_v$ is its rate when delayed. The game plays out in continuous time and the project is managed under an early start policy. This means that every task is initiated as soon as all its predecessors complete. At any instant the interdictor must choose which tasks he will delay. Delaying each task has an integer non-discounted cost $c_v \geq 0$ associated with it. Let H be the set of tasks that have been delayed during the game. It is required that $\sum_{v \in H} c_v \leq b$. The interdictor's goal is to maximize the expected time until the finish task can begin (i.e. the project is complete).

We also make the following additional assumptions.:

- All activity durations are mutually independent and exponentially distributed.
- An interdiction attempt on a task can only be made when that task is in progress.
- If an interdiction is performed on a task $v \in V$, its residual processing time is then exponentially distributed with intensity ν_v
- An interdiction can be done at most once on task.

The first assumptions allows our model to enjoy the memoryless property; making it possible for us to solve the problem analytically. The second assumption is reasonable since we consider every task in a project to be an indivisible unit of work. If there is some interdiction on a task that can be performed earlier than that task begins we assume that it works by delaying some prerequisite activities and refine our model by introducing them into the network. The last assumption is equivalent to stating that there are two modes of execution for any task - a normal and delayed one and it is only possible to switch from the former to the latter. This also simplifies the problem but is possible to generalize.

4.2 The decision process

We are going to base our model on the Markov process by Kulkarni and Adlakha [1986] which they describe it in terms of activity-on-arc networks. However, like in Creemers et al. [2010]; Sobel et al. [2009], we adopt the activity on node convention.

At any time $t \geq 0$, a task can either be *idle* , that is not yet started, *active*, in other words processing, or *finished*. As we are working with interdiction games, we allow for a task that is active to also be *normally active* or *delayed active* (or simply *delayed*). We encode the state of the interdiction game as a tuple $x = (I_x, N_x, D_x, F_x, b_x)$. Where the integer $b_x \in \{0, \dots, n\}$ is the remaining budget, I_x, N_x, D_x, F_x are the sets of idle, active, delayed and finished tasks that form a an exhaustive partition of V. For convenience we define $L_x = N_x \cup D_x$ as the set of all tasks in progress. We sometimes write x_t as the state at time t.

The *starting* state of the game is $x_0 = (V \backslash \{1\}, \{1\}, \emptyset, \emptyset, b)$. That is one in which only the source node 1 is active, the remainder idle and b is the initial budget. Conversely, any state with $F_x = V_x \backslash \{n\}$ or $F_x = V_x$ is a *terminal* state.

Definition 1. *A state x is said to be admissable if it satifies the causality property, that is*

$$u \in L_x \implies v \in F_x \ , \forall v \prec u \tag{4.1}$$

and additionally

$$b - b_x \geq |D_x| \tag{4.2}$$

We can now define Ω as the space of all admissable states and $\Omega^\infty \subset \Omega$ as the set of terminal states. Note that because the project manager follows the earliest start rule, the states which are visited satisfy an additional property

$$v \in F_x \ , \forall v \prec u \implies u \in L_x \cup F_x \tag{4.3}$$

From our definition of a state, we find that $|\Omega| \leq 4^n b$. But because of the constraints (4.1), (4.2) and (4.3) it is usually that $|\Omega| \ll 4^n b$.

In our game we allow the interdictor at any time to perform actions $a \in \mathcal{A}$ where $\mathcal{A} \subseteq 2^V$ is the action space. These are choices of which tasks to delay in each state. An action a is feasible in x iff $a \subseteq N_x$ and $|a| \leq b_x$. We refer to $a = \emptyset$ as the null action which is always feasible. The set of all feasible actions is A_x.

State transitions are defined in the following way. Let $x \in \Omega$ be the current state at time $t \geq 0$ and $a \in A_x$ the action taken. We have an immediate *post-decision* state $y = y^0_{xa}$. As soon as some task $v \in L_x$ completes (the exogenous information), we call the successor state $y = y^v_{xa}$. In the remainder of this report we will assume w.l.o.g that each interdicting any task has a cost of 1.

Definition 2. *$E_x(v)$ is the eligible set. That is, the set of activities that can start if the current state is x and activity v has finished.*

$$E_x(v) = \{u \in I_x \mid w \prec u \Rightarrow w \in F_x \cup \{v\}\}$$

Successor state	$y = y^v_{xa}$	$y = y^0_{xa}$
Interdiction budget	$b_y = b_x$	$b_y = b_x - \lvert a \rvert$
Idle tasks	$I_y = I_x \backslash E_x(v)$	$I_y = I_x$
Normal active	$N_y = (N_x \backslash \{v\}) \cup E_x(v)$	$N_y = N_x \backslash a$
Delayed active	$D_y = D_x \backslash \{v\}$	$D_y = D_x \cup a$
Finished	$F_y = F_x \cup \{v\}$	$F_y = F_x$

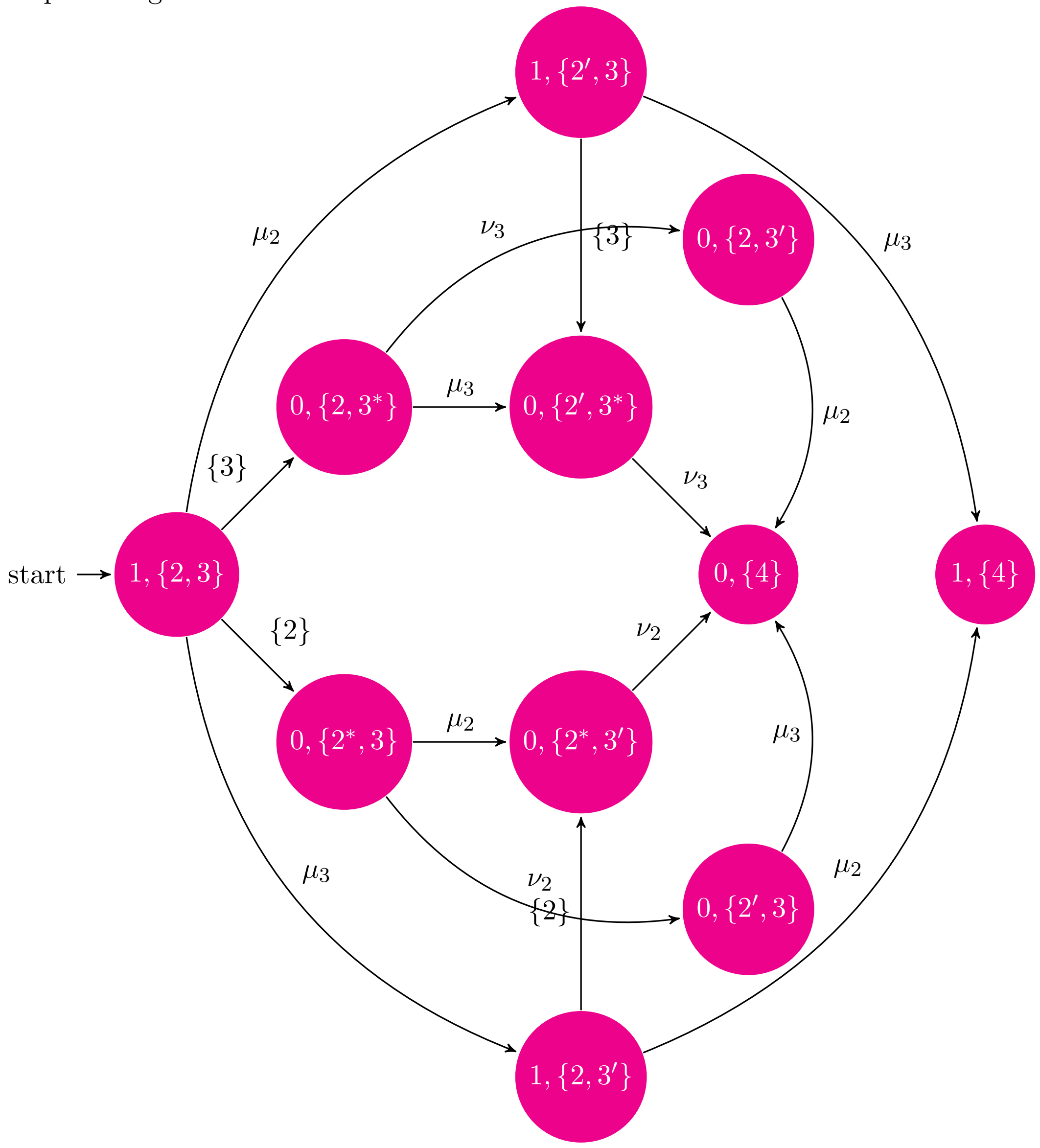
Table 4.2: This table shows how to construct the post-exogenous-information and post-decision states.

4.3 Discrete time MDP formulation

We can find an optimal policy to the above continuous time problem by using a discrete time finite horizon MDP. Assume for now that the interdictor can only make decisions on which tasks to delay to delay at time $t = 0$ or $t = c_v$ where c_v is the completion time of a task $v \in V$ (A1). This is equivalent to requiring that any action at any time other than $t = 0$ or $t = c_v$ is $\emptyset$. Later we show that adding this restriction does not make the solution any less optimal thanks to the memoryless property.

First we define $q_{y|xa}$ to be the rate of the activity whose completion causes a tran-

Figure 4.2: Graph of Example 1's continuous time MDP where the budget is 1. The vertices are states and are labeled with the current budget and the set of non-dummy and non-idle activities. a' means that task a is finished while a^* means it is delayed active. A transition to a post-decision state is labeled with the action and a transition to a post-exogenous -information state is labeled with its rate.



sition $x \to y$ under action a under assumption (A1). That is

$$q_{y|xa} := \begin{cases} \mu_v & \text{if } y = y^v_{za}, z = y^0_{xa}, v \notin D_z \\ \nu_v & \text{if } y = y^v_{za}, z = y^0_{xa}, v \in D_z \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

Theorem 1. *Let $\pi^* : \Omega \to \mathcal{A}$ be the optimal policy for the DTMDP $\langle \Omega, \mathcal{A}, p.(.,.), r.(.) \rangle$ where Ω, $\mathcal{A}$ are the original state and action spaces respectively and $r.(.)$, $p.(.,.)$ are defined as:*

$$r_a(x) := \left(\sum_{y \in \Omega} q_{y|xa} \right)^{-1}$$

and

$$p_a(x, y) := q_{y|xa} r_a(x)$$

Taking action $\pi^*(x_t)$ at time $t = 0$ or $t = c_v$ maximizes the expected project makespan under assumption (A1).

Furthermore, π^* can be computed from the Bellman equations

$$\pi^*(x) = \operatorname*{argmax}_{a \in A_x} \left\{ r_a(x) + \sum_{y \in \Omega} p_a(x, y) V(y) \right\} \qquad x \in \Omega \backslash \Omega^\infty \tag{4.5}$$

$$\pi^*(x) = \emptyset \qquad x \in \Omega^\infty \tag{4.6}$$

where

$$V(x) = \max_{a \in A_x} \left\{ r_a(x) + \sum_{y \in \Omega} p_a(x, y) V(y) \right\} \qquad x \in \Omega \backslash \Omega^\infty \tag{4.7}$$

$$V(x) = 0 \qquad x \in \Omega^\infty \tag{4.8}$$

Proof. We can use an inductive argument as the MDP is transient. Assume $x \in \Omega^\infty$ then $V(x) = 0$ as required since there is nothing left to process. Let $x \notin \Omega^\infty$, a be a an action and c_v for $v \in L_x$ be the random completion times. The post-decision state is immediately $z = y^0_{xa}$. The expected time spent in z is then

$$\mathbb{E} \left(\min_{v \in L_z} \{c_v\} \right) = \sum_{v \in \Omega} q_{y|xa} = r_a(x)$$

since $A_z = \{\emptyset\}$ and using the property of exponential distributions. The expected time *after* transitioning out of z, assuming inductively that $V(y)$ is the expected time from the next state y is then

$$\mathbb{E}(V(y)) = \sum_{v \in L_z} \Pr\left(c_v = \min_{u \in L_z}\{c_u\}\right) V(y^v_{za}) = \sum_{y \in \Omega} \underbrace{\left(\frac{q_{y|xa}}{\sum_{z \in \Omega} q_{z|xa}}\right)}_{p_a(x,y)} V(y)$$

Adding the above two expressions gives the expected project makespan under a and maximizing over the actions A_x yields (4.5). □

Proposition 1. *The interdictor's policy is no less optimal under restriction (A1) than it would be if decisions could be made at any time.*

Proof. Let x_t be the game's state at t and the optimal action be a^*. Let $s > t$ be a time before the next transition and x_s be the post-decision state at time s then $N_{x_s} = N_{x_t} \backslash a^*$, $F_{x_s} = F_{x_t}$, $I_{x_s} = I_{x_t}$ and $D_{x_s} = D_{x_t} \cup a^*$ by definition. For any action $a_s \in A_{x_s}$ we can find an action $a_t \in A_{x_t}$ such that $V^{a_s}(x_s) = V^{a_t}(x_t)$. More precisely $a_t = a_s \cup a^*$ which follows from the memoryless property of activity durations and (4.5). It must also be that $a_s \cap a^* = \emptyset$ because the tasks in a^* would no longer be available to delay and therefore $a_s = a_t \backslash a^*$. We find that for any $b \in A_{x_s}$ that

$$V^{\emptyset}(x_s) = V^{a^*}(x_t) \geq V^{a^* \cup b}(x_t) = V^b(x_s)$$

and therefore $\emptyset \in \text{argmax}_{a \in A_{x_s}}\{V^a(x_s)\}$. The implication of this is that the optimal action in any intermediary state x_s is the null action and therefore we only need search for optimal policies with (A1). □

4.4 Algorithm for solving larger PERT networks

We have shown a discrete time formulation for solving a dynamic interdiction game that is optimal in the continuous setting. Unfortunately using standard algorithms that solve the MDP exactly such as policy, value iteration or linear programming is computationally infeasible for even medium sized PERT networks (with 30 activities or more). We instead use a specialized dynamic programming algorithm proposed by Creemers et al. [2010] and adapt it to our problem. The algorithm addresses a major computational bottleneck which is the need to store the state and decision space in memory. With this algorithm we can solve for medium sized PERT networks exactly.

A key ingredient in the development of our algorithm is a characteristic set which Creemers et al. [2010]; Kulkarni and Adlakha [1986] refer to as a UDC (uniformly directed cut) since it was a cut of the activity on arc flow network. Since we use do not use AOA our definition is the following:

Definition 3. *A UDC U of V, induced by the relation $\prec$, is an inclusion maximal antichain. It is a subset of V which satisfies the following:*

- $u \nprec v \wedge v \nprec u$ *for* $u, v \in U$
- $\neg \exists U'$ *s.t.* $\exists v \in V$ *s.t.* $U' = U \cup \{v\} \wedge$ *the above holds for U'*

The algorithm consists of two main steps. The first is to generate the set of UDCs $\mathfrak{U}$ (Definition 3). Creemers et al. [2010] showed that $\mathfrak{U}$ is a partition of the state space Ω [1]. We use the same algorithm as Creemers et al. [2010]; Stork and Uetz [2005] which is given below. Essentially we enumerate the nodes of a tree T. The root node

```
Procedure FindUDCs(node k, ancestors)
  $\mathfrak{U} := \emptyset$
  if $antichain(\{k\} \cup ancestors)$ then
      maximal := true
      foreach $v \in V, v \neq k$ do
          if $antichain(\{k, v\} \cup ancestors)$ then
              maximal := false
      if $maximal$ then
          $\mathfrak{U} := \mathfrak{U} \cup (\{k\} \cup ancestors)$
          return $\mathfrak{U}$
      if $k < n$ then
          foreach $i := k + 1$ to $n$ do
              $\mathfrak{U} := \mathfrak{U} \cup FindUDCs(i, ancestors \cup \{k\})$
  else Discard this node
```

corresponds to the empty set and its children are correspond to activities $1, \ldots, n$. Each node j, associated to the activity j, has children $j+1, \ldots, n$. There are $2^{|V|}$ nodes in such a tree and it can be shown that there is a bijection between the tree nodes and the power set 2^V. Stork and Uetz [2005] use a depth first search procedure to visit nodes of a tree and prune it whenever they finds a non-admissable subset. We implemented this as a recursive procedure for simplicity. This algorithm is polynomial in the number of UDCs but exponential in the worst case.

[1]Our state space is not exactly the same as in their paper but the ideas are similar

Figure 4.3: PERT network with 6 activities

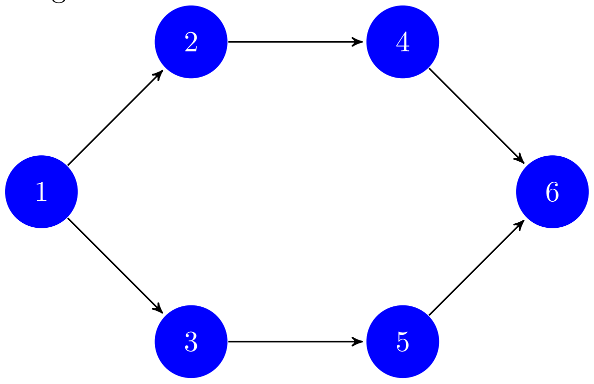
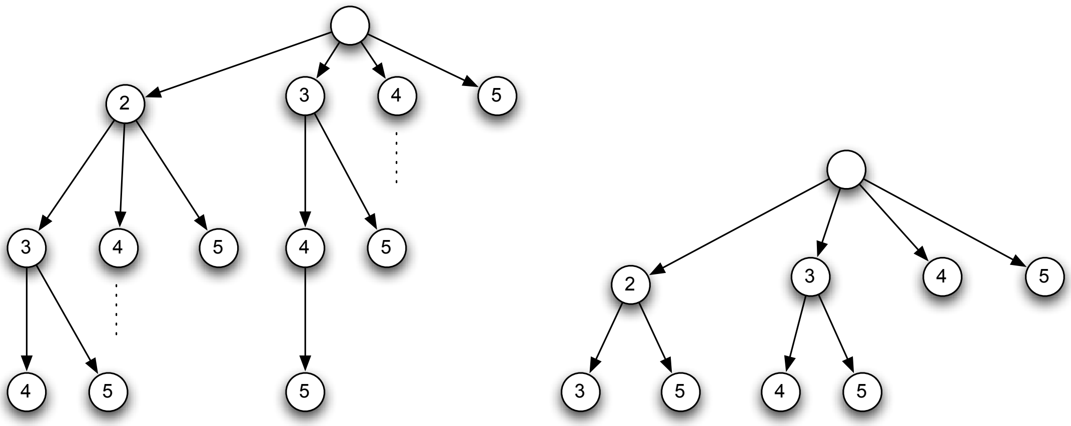


Figure 4.4: An example tree T (left) and its pruned counterpart (right). The resulting UDCs are $\{2,4\},\{2,5\},\{3,4\}$ and $\{3,5\}$



Example 2. *Consider the PERT network in Figure 6.2. The algorithm for finding the UDCs will produce the trees shown in Figure 6.3. Notice that the leftmost subtree has been discarded because task 2 precedes 3.*

With each $U \in \mathcal{U}$ we associate a rank

$$r(U) := |\{i \in V \mid \exists j \in V \text{s.t. } j \prec i\}| \tag{4.9}$$

that counts the number of predecessor activities. We index UDCs by their rank and from them generate a UDC network $N = (\mathcal{U}, S)$ where $S(U, V)$ if and only if there exists an activity v that is active or delayed in some state of U and whose completion causes a transition to a state in V. Creemers et al. [2010] showed that inter-UDC transitions are only possible from a lower to a higher ranked UDC. During the UDC enumeration procedure we record transitions and the number of incoming ones to a each $U_i \in \mathcal{U}$. We store this value as a variable l_i. Each UDC has a set of states $\sigma(U)$ where each $x \in \sigma(U)$ represents an admissable partition of U Kulkarni and Adlakha [1986] into L_x, F_x and I_x. Since our focus are interdictions games, we split L_x into the normally active and delayed active sets and produce copies for different $b_x \in \{0, \dots, b\}$. Likewise Creemers et al. [2010] showed that there is a surjection $\sigma^{-1} : \Omega \to 2^V$ from the state space to the UDC space $\mathcal{U}$ which applies to our problem since it is a 'special case'.

In order to compress the storage requirements we encode a state into an integer tertiary value that is used as an index into an array associated with a UDC. Creemers et al. [2010] used this to perform a a binary search when looking up a $V(x)$. This is preferred to a general hash table because the latter needs to map a potentially infinite universe of keys to the array whereas we 'know' all the states a priori. To emphasize how valuable this is, it is worth noting that we are able in theory to store approximately 256,000,000 in memory at once on an ordinary PC. To make this work we order the elements of U as $1, \dots, |U|$ and let $\rho_i \in V$ denote the task in the ith position. We will now show an appropriate tertiary value function that can be used in for our problem.

Table 4.3: This table shows the states of the $\{2, 5\}$ UDC of Figure 4.3 and their τ values

N_x	D_x	F_x	b_x	$\tau(x)$
2		4,5	0	25
	2	4,5	0	24
2,5		4	0	19
2	5	4	0	18
5	2	4	0	17
2		4,5	1	9
2,5		4	1	3

Proposition 2. *Let $m = |\sigma^{-1}(x)|$. For $x, y \in U$ if $\tau(x) < \tau(y)$, where*

$$\tau(x) := \sum_{i=1}^{m} 2^{i-1} \left(\mathbb{1}_{[N_x]}(\rho_i) + 2^m \mathbb{1}_{[F_x]}(\rho_i) \right) + 4^m (b - b_x) \tag{4.10}$$

and $\mathbb{1}_{[N_x]}(\rho_i)$ is an indicator function that is 1 if $p_i \in N_x$ else 0, then $V(y)$ will not depend on $V(x)$

Proof. It is sufficient to show that x does not occur after y in any sample run of the MDP. Suppose that it does, then $b_x \leq b_y$ and $F_x \supseteq F_y$. If $F_x = F_y$ then $b_x < b_y$ as the only changes that have happened is some tasks have been delayed.

$$\tau(x) \geq 4^m(b - b_x) > 4^m(b - b_y) + 2^{2m-1} - 1 \geq \tau(y)$$

If $F_x \supset F_y$, it must be that

$$\begin{aligned}\tau(x) &\geq 4^m(b - b_y) + \sum_{i=1}^{m} 2^{i-1}\left(\mathbb{1}_{[N_x]}(\rho_i) + 2^m \mathbb{1}_{[F_x]}(\rho_i)\right)\\ &> 4^m(b - b_y) + \left(2^{m-1} - 1\right) + \sum_{i=1}^{m} 2^{m+i-1}\mathbb{1}_{[F_y]}(\rho_i)\\ &\geq \tau(y)\end{aligned}$$

So in either case we contradict our assumption that $\tau(x) < \tau(y)$. □

Proposition 3. *For any $U \in \mathfrak{U}$ and $x, y \in U$, $x \neq y$ iff $\tau(x) \neq \tau(y)$.*

Proof. $x \neq y$ iff one of $N_x \neq N_y$ or $F_x \neq F_y$ is true. Since $\tau(.)$ depends on these sets it must be different for x and y. If the $\tau(x) \neq \tau(y)$ then at least one of the terms must differ in (4.10) also making the state differ too. □

Corollary 1. *$\tau : \Omega \to \mathbb{Z}$ is an injective function.*

What makes this algorithm capable of solving larger PERT networks is that the need to compute Bellman's equation for every state in one go is decomposed into several steps. In each one we solve all the states of a particular UDC before moving onto the next one. When the value function V's values for a UDC are no longer required they are freed up. This would happen if there is no way to transition to the UDC in question. Thus we only need to keep a fraction of states in memory at any given time. Furthermore we only need to apply a backward induction that solves the value and optimal decision for each state once which makes the algorithm $\Theta(|\Omega|)$. The ordering imposed by the tertiary function τ guarantees that the value computed for every state is always available for the next one within the same UDC. Since inter UDC transitions can only go from a lower to strictly higher rank it is also possible to solve UDCs of the same rank in parallel. To implement a paralellized version we would need to enforce synchronization in the deleting step to avoid race conditions.

Algorithm 3: Solve for optimal policy

```
Input: Activities $1, \dots, n$, Relation $\prec$, Vectors $\mu$ and $\nu$
Output: The optimal interdiction policy $\pi^*$
Generate the UDC network $N(\mathcal{U}, S)$
$V(x) := 0$ for $x \in \Omega^\infty$
foreach $r := n-1, \dots, 1$ do
    foreach $U$ of rank $r$                         /* May be done in parallel */
    do
        foreach $x \in \sigma(U)$ in decreasing order of $\tau(x)$ do
            $V(x) := \max_{a \in A_x}\{r_a(x) + \sum_{y \in \Omega} p_a(x,y)V(y)\}$
            $\pi^*(x) := \mathrm{argmax}_{a \in A_x}\{r_a(x) + \sum_{y \in \Omega} p_a(x,y)V(y)\}$
        end
        foreach $U_i$ such that $S(U, U_i)$ do
            $l_i := l_i - 1$
            if $l_i = 0$ then
                Free all data associated with $U_i$
            end
        end
    end
end
```

4.5 Software implementation challenges

We will briefly explain some challenges of efficiently implementing the algorithm in Java. The first prototype we developed was notoriously slow. After profiling the application, we found that this was caused by a hot spot in the procedure for generating the next state because we used the `HashSet` class's methods to find it. Instead, we tried using a bit vector representation of a set and bitwise operations with the `BigInteger` class. Next we introduced a cache to store a state 'template' that would look up which tasks are active (normally and delayed) and finished if a certain activity completes. Also this cache would only exist in a UDC evaluation step. The two modifications sped up the code 20 times on average!

4.6 Robust MDP formulation and assumptions

We have proposed a dynamic model of an interdiction game and an algorithm to solve it. However, our goal is to build on the deterministic model of Brown et al. [2005, 2009], discussed in Chapter 3. We have simplified the problem by considering only one technology, allowing only integer interdiction costs and not considered the possiblity of the project manager crashing tasks.

In this section we extend our model by letting the project manager expedite tasks. We assume that he possesses renewable resources $k = 1, \dots, K$ where each has a constant availability R_k that is always constant. We assume that he is able to revise his decision, like the interdictor only when some task completes or at time 0 (A2) and that he does this after the interdictor (A3). We will show that, just as in the case of the interdictor, this restriction does not make the adversary's policy any less optimal. We introduce his decision variables in state x as $\{\xi_v\}_{v \in L_x}$ which are the amounts by which he crashes tasks in progress. For example, if $\xi_v = 2$ he will crash v by a factor of 2 and we shall explain shortly what this means in our model. We assume that crashing an activity v requires a_{kv} amount of resource k per unit of crashing and thus we require that $\sum_{v \in L_x} a_{kv}\xi_v \leq R_k$ for $k = 1, \dots, K$. Additionally we constrain that $0 < l_v \leq \xi_v \leq u_v < \infty$ for $v \in L_x$ and we let Ξ_x be the polyhedral uncertainty set of all such admissable allocations in x. We assume that this intersection of a polytope and hyper-rectangle is non-empty, in other words for every k, R_k is large enough to accommodate the minimum resource allocation given by the vector $l \in \mathbb{R}^+_P$, where $P = |L(x)|$, to all activities in progress in any state. The uncertainty sets in different states are independent of each other. We shall now elucidate the crashing model which use and can also be found in Elmaghraby and Girish [2010]; Rudolph and Elmaghraby [2007]; Tereso et al. [2003, 2004a,b], albeit in the context of the TCTP. We adopt the view that the uncertainty of a task's duration is a function of deterministic and stochastic components. The stochastic component is termed work content. We can describe it as an uncertain measure of effort required to complete a task which is exponentially distributed. We denote it as a r.v. $w^v_{xa} \sim \exp(\mu^v_{xa})$ which is a function of the current state x and interdictor's action a. The adversary's choice of $\xi \in \Xi_x$ is the non-stochastic component and given both ingredients, the task's random duration is $c^v_{xa}(\xi) = w^v_{xa}/\xi_v$. It can be shown that $c^v_{xa}(\xi) \sim \exp(\xi_v \mu^\pi_{vx})$. We therefore assume a hyperbolic relationship between the activity's processing rate and resource allocation level.

Theorem 2. *Assume that the interdictor and project manager act according to assumptions A1, A2 and A3. Solving Bellman equations*

$$V(x) = \max_{a \in A_x} \inf_{\xi \in \Xi_x} \left\{ \frac{1 + \sum_{v \in L_x} \xi_v \mu^v_{xa} V\left(y^v_{xa}\right)}{\sum_{u \in L_x} \xi_u \mu^u_{xa}} \right\} \qquad x \in \Omega \backslash \Omega^\infty \qquad (4.11)$$

$$V(x) = 0 \qquad x \in \Omega^\infty \qquad (4.12)$$

maximises the project's mean makespan

Proof. We can use the same argument as for Theorem 1 and in addition use the fact that $\mathbb{E}\left(c^v_{xa}\right) = \left(\xi_v \mu^\pi_{vx}\right)^{-1}$ $\square$

Proposition 4. *The value in (4.11) is optimal for the adversary even if he were able to make decisions at any time, i.e. relaxing assumption A2.*

Proof. Suppose that at time t the game transitions into state x. The interdictor takes action $a_t \in A_x$ and then an optimal ξ^*_t is chosen by the adversary. Let $s > t$ be a time before the next state transition. The tasks in progress have not changed and by the memoryless property every activity's remaining work content $w^v_{xa} \sim \exp(\mu^v_{xa})$ has the same distribution at time s as in the post-decision state at time t, $y^0_{xa_t}$. Therefore, the adversary's optimal crashing decision at time s is unchanged, $\xi^*_s = \xi^*_t$. Likewise, from Proposition 1, the interdictor's null action is optimal. $\square$

Note that the function being minimized in (4.11) is quasilinear (quasi-convex and quasi-concave) in ξ.

We now focus our attention on the adversary's problem in a certain state exclusively and will drop the subscripts a and x since we will assume that they remain fixed. Also we relabel $G_v := \mu^v_{xa} V(y^v_{xa})$ and $\mu_v := \mu^a_{xv}$ and the set of active tasks is $L := L_x$. Finally the uncertainty set is called Ξ. We will define the optimal value of adversary's sub-problem,

$$v_a(x) = \inf_{\xi \in \Xi} \left\{ \frac{1 + \sum_{v \in L} \xi_v G_v}{\sum_{u \in L} \xi_u \mu_u} \right\} \tag{4.13}$$

Proposition 5. *For every state x and action a, $v_a(x)$ equals the optimal objective value of the LP where $A := (a_{kv})_{k=1,\ldots,K, v \in L(x)}$.*

$$\inf_{\substack{z \in \mathbb{R}_+, \\ y \in \mathbb{R}^n_+}} \left\{ z + \sum_{v \in L} G_v y_v \mid Ay - zR \le 0,\ \sum_{v \in L} \mu_v y_v = 1, y \in [zu, zl] \right\} \tag{4.14}$$

and $\xi_v = y_v / z$.

Proof. $v_a(x)$ is the optimal objective value of a linear fractional program. See Boyd and Vandenberghe [2004]; Charnes and Cooper [1962]. $\square$

It is possible to solve the LP (4.14) in every state of the system as part of the dynamic programming algorithm but it will place a tremendous computational burden if we are to use such a direct approach. We will now give a special case where a more efficient method is applicable.

Suppose $K = 1$ i.e. the adversary has only one resource, then the uncertainty set Ξ becomes the intersection of a hyper-rectangle and a closed halfspace. If we remove the upper bounds on ξ_v, then the feasible set becomes a $P+1$ dimensional simplex for which we can solve the problem analytically. We now define the constant $d := R - \sum_v l_v$, for convenience.

Proposition 6. *Assuming there is no upper bound on individual task crashing, that is $u_v = +\infty$ for all v and there is a single resource constraint, the optimal value $v_a(x)$ can be found analytically in time $\mathcal{O}(n)$ with the following expression*

$$v_a(x) = \min_{v \in L} \left\{ \frac{1 + \sum_{u \in L(x)} J_u \left(l_u + \mathbb{1}_{[u=v]}(d - l_u) \right)}{\sum_{w \in L(x)} \mu_w (l_u + \mathbb{1}_{[w=v]}(d - l_u))} \right\} \tag{4.15}$$

Proof. We will first show a technical lemma

Lemma 1. *Let f be a quasiconcave function and x_i for $i = 1, \ldots, n$ be a set of points in* ***dom****f and $z = \sum_{i=1}^n \alpha_i x_i$ where $\alpha \geq 0$ and $e^T \alpha = 1$. It follows that $f(z) \geq \min_{i=1,\ldots,n}(f(x_i))$.*

We can show this by a straightforward induction on n. If $n = 2$ then because f is quasi-concave it follows that $f(\alpha x + (1-\alpha)y) \geq \min(f(x), f(y))$. Let $z = \sum_{i=1}^k \alpha_i x_i$ and define $\gamma := \sum_{i=1}^{k-1} \alpha_i$. We then express $f(z)$ as

$$\begin{aligned} f(z) &= f\left(\gamma \sum_{i=1}^{k-1} \frac{\alpha_i}{\gamma} x_i + \alpha_k x_k \right) \\ &\geq \min \left(f\left(\sum_{i=1}^{k-1} \frac{\alpha_i}{\gamma} x_i \right), f(x_k) \right) \\ &\geq \min_{i=1,\ldots,n} (f(x_i)) \end{aligned}$$

The first inequality follows from $\gamma + \alpha_k = 1$, the second from the induction hypothesis.

We note that the feasible set Ξ is a $P+1$ simplex, by assumption, with vertices $C = \{de_v + l_v \mid v \in L\} \cup \{l\}$. Therefore every feasible point is in the convex hull of C.

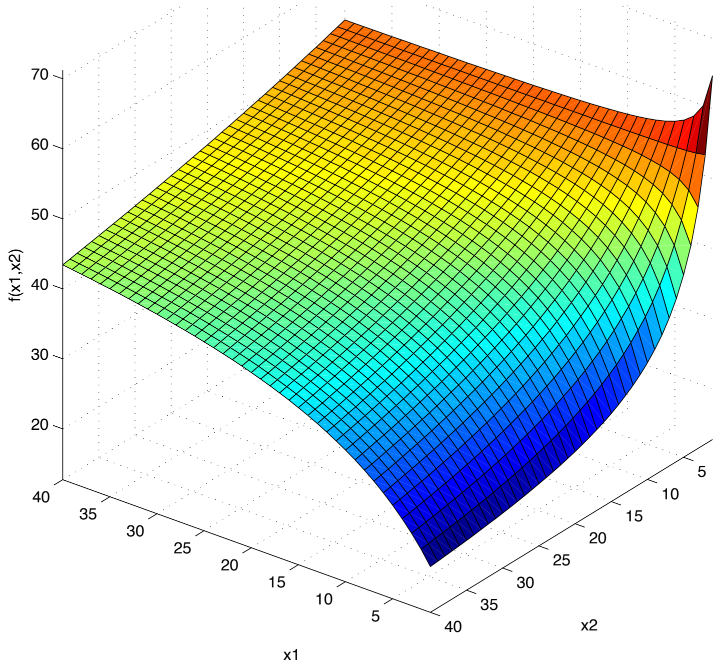
$$\textbf{conv}(C) = \left\{ \sum_{v \in L} \alpha_v x_v \mid x_v \in C, \sum_{v \in L} \alpha_v = 1, \alpha_v \geq 0 \ \forall v \in L \right\} \tag{4.16}$$

Let φ denote the objective function in (4.13). We know that φ is quasi-concave and therefore for any feasible point $z \in \mathbf{conv}(C)$

$$\varphi(z) = \varphi\left(\sum_{v \in L} \alpha_v x_v\right) \geq \min_{v \in L} \left(\varphi(x_v)\right) \tag{4.17}$$

$\square$

Figure 4.5: Illustration of a linear fractional function with of the form in (4.11)



4.7 Implementation uncertainty

We consider a final extension to our model where an interdiction action's effectiveness is random. By that we mean that if we were to interdict a task, unlike before, not only is its remaining processing time random, but also the parameter of that distribution itself is random, i.e. μ_v is a r.v.

First, we look at the simplest case where the residual rate parameter is a binary random variable. The first of the two possible outcomes is that the interdiction succeeds and the residual processing time of task v, $R_v \sim \exp(\nu_v)$, the second is that the interdiction fails and the remaining time $R_v \sim \exp(\mu_v)$. The advantage of this particular model is that we do not need to modify the original state space Ω. If an interdiction fails in a state then the budget is decremented with no other change. The action space $\mathcal{A}$ is also identical. However the state transition probabilities would need to change.

Let $x, y \in \Omega$, $p_a(x,y)$ be the transition probability of the original DTMDP, $p_{u|a}$, the probability that an interdiction succeeds on activities $u \subseteq a \subseteq V$ but fails on the rest [1]. The transition probabilities of the model under consideration $p'_a(x,y)$ are given by

$$p'_a(x,y) := \begin{cases} p_a(x,y) & \text{if } a = \emptyset \\ p_{u|a} & \text{if } N_y = N_x \backslash u, D_y = D_x \cup u, b_y = b_x - |u| \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

The state rewards may also be redefined as

$$r'_a(x) := \begin{cases} r_a(x) & \text{if } a = \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

The Bellman recursion is the same as before but with the new state transition probabilities and state rewards.

$$V'(x) = \max_{a \in A_x} \left\{ r'_a(x) + \sum_{y \in \Omega} p'_a(x,y) V'(y) \right\} \tag{4.20}$$

Corollary 2. *If $p_{a|a} = 1$ and $p_{u|a} = 0$ for all $u \subset a$ then the new model and original are equivalent.*

Proof. Let x be a state, $a \in A_x$ the action taken. If $a = \emptyset$ then by definition of (4.18) the state transition probability is the same as in the original model. If $a \neq \emptyset$ then as $p_{a|a} = 1$, the next state y by (4.18) is with probability 1 identical to the post-decision state y^v_{xa} in the original model. $\square$

4.8 Constrained MDP formulation

In the main dynamic model of this thesis, the one without crashing and with implementation certainty, we used integer costs. It is possible, if needed, to have real valued costs that are discounted at a continuously compounded rate $r \in (0,1)$. The caveat is that

[1]The marginal probabilities need not be independent, they could be correlated since the joint distribution is left to the user's design

the budget constraint will only be obeyed *on average*. The nature of the formulation makes the resultant optimal policy *randomized*.

First we introduce a new state space Ω where each state is a tuple $x = (I_x, N_x, D_x, F_x)$ - so it is the same as before but without the b_x element. The definition of terminal states Ω^∞ is unchanged. Now let's introduce two types of costs: the first is incurred at the end of the project and is some large value C. We will also assign an interdiction cost to each activity $v \in V$, $h_v > 0$. Let $J^\pi(x)$ be the cost-to-go in state x under policy π. Then $J^\pi(x) = M$ for $x \in \Omega^\infty$. For any non-terminal state, the cost-to-go is the discounted expected cost-to-go of any next state y, where the discount factor is determined by the random time spent in the present state τ_x:

$$\begin{aligned} J^\pi(x) &= \mathbb{E}\left(e^{-r\tau_x}\right) \sum_{y\in\Omega} p_\pi(x,y) J^\pi(y) \\ &= \int_0^\infty e^{-rt} t \left(q^\pi(x) e^{-q^\pi(x)t}\right) dt \sum_{y\in\Omega} p_\pi(x,y) J^\pi(y) \\ &= q^\pi(x) \int_0^\infty t e^{-(q^\pi(x)+r)t} dt \sum_{y\in\Omega} p_\pi(x,y) J^\pi(y) \\ &= \frac{q^\pi(x)}{q^\pi(x)+r} \sum_{y\in\Omega} p_\pi(x,y) J^\pi(y) \\ &= \sum_{y\in\Omega} \frac{q_{y|x\pi}}{q^\pi(x)+r} J^\pi(y) \end{aligned}$$

where $q^\pi(x)$ is the rate of the exponential distribution of τ_x and $p_\pi(x,y)$ is the probability of the next state and $q_{y|x\pi}$ is the completion rate under the policy and state of the activity that causes an $x \to y$ transition.

By combining a technique from Dmitri and Edmund [2005]; Kallenberg [1983] and the Bellman equations above, we can solve a constrained MDP

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & C \left(\sum_{x\in\Omega^\infty, a\in A_x} \theta_{xa} \right) \\ \text{subject to} \quad & \sum_{a\in A_x} \theta_{xa} - \sum_{y\in\Omega, a\in A_y} \left(\frac{q_{y|xa}}{q^\pi(x)+r} \right) \theta_{ya} = \mathbb{1}_{[x=x_0]}, \quad x \in \Omega \\ & \sum_{x\in\Omega, a\in A_x} \left(\sum_{v\in a} h_v \right) \theta_{xa} \le b \\ & \theta \ge 0 \end{aligned} \tag{4.21}$$

The randomized optimal policy is then found from

$$p_{xa} = \theta_{xa} / \sum_{u \in A_x} \theta_{xu} \tag{4.22}$$

Where p_{xa} becomes the probability of doing action a in state x.

4.9 Other variations and extensions

There are other ways to extend our primary model. It is possible to introduce different ways of interdicting activities by expanding the action space. Each way can have its own cost, success probabilities and so on. Also it is possible to combine crashing with implementation uncertainty. To be able to continue building ever more sophisticated models it is worth exploring approximate dynamic programming which is the subject of the next chapter.

Chapter 5

Approximate solutions

Our aim in this part is to find algorithms that can solve approximately some dynamic interdiction games but are much more scalable. We will focus on one kind of heuristic method known in the literature as ADP (approximate dynamic programming) or reinforcement learning. These techniques have generated interest for their ability to overcome the curse of dimensionality in some cases and still give high-quality solutions. Another benefit is they give the user more flexibility in modeling. For example, we have in this thesis been tied to strong assumptions about independent exponential distributions.

The common idea behind ADP algorithms is to approximate the value function V (in the Bellman equation) and so avoid having to enumerate the whole state space. To do this they use a mixture of simulation and machine learning. Some ADP algorithms are mostly dependent on simulation, for example, Q-learning or SARSA. Their advantage is they are are relatively model-free. That means that they don't need to make assumptions about Markov chain transition probabilities. Other types of ADP algorithms use state aggregation but remain model-dependent. Finally the algorithms that we will focus on are *parametric* methods which can (loosely) be viewed as a compromise between the model-dependent and simulation approaches. We will mostly be using algorithms from Bertsekas and Tsitsiklis [1996]; Powell [2007].

5.1 Basis functions

One way to approximate the value function is to use a regression model. Let $\tilde{V}^{\pi}(x)$ be the approximate value of being in state $x \in \Omega$ under the policy π. We can compute it from

$$\tilde{V}^{\pi}(x) = \sum_{i=1}^{k} \theta_i \phi_i(x) \tag{5.1}$$

where $\phi_i(x), i = 1, \ldots, k$ are a set of k basis functions and θ_i's are weights. The definition of the basis functions are left up to the algorithm designer but they can be thought of as features of a state. To find a good approximation architecture requires some experimentation and insight into the specific problem under consideration.

We have $2n+2$ basis functions where $\phi_1(x) = 1$, $\phi_2(x) = b_x$, the next $i = 3, \ldots, n+2$ are

$$\phi_i(x) = \begin{cases} 0 & \text{if } i-2 \in I_x \\ 1 & \text{if } i-2 \in N_x \\ 2 & \text{if } i-2 \in D_x \\ 3 & \text{otherwise} \end{cases} \tag{5.2}$$

and the final $i = n+3, \ldots, 2n+2$ are

$$\phi_i(x) = \phi_{i-n}^2 \tag{5.3}$$

5.2 Approximate policy iteration

We will try to approximate the solution to the standard problem (no crashing, certain success) in Chapter 4. This can easily be generalized to the implementation uncertainty problem. Moreover, we can come up with more sophisticated outcomes: for example a task can be delayed by factor that is a continuous random variable rather than be limited to discrete outcomes. This is because in these ADP algorithms, we use simulation to estimate the policy's value without building infinite Markov chains. However, we hypothesize that this will not work for the crashing game because there are two decision makers.

We use a simple state sampling method given below. One of its drawbacks could be that it does not give a uniform distribution over the states sampled and it would be worthwhile to investigate alternatives.

5.3 Regression methods

We have given the outline of a policy-iteration algorithm that estimates the value of being in each state. To give the complete algorithm we will state which regression

Algorithm 4: Approximate policy iteration

```
Input: Project $G = (V, E)$, budget $b$, initial policy $\pi_0$ and value estimate $\tilde{V}^{\pi_0}(.)$
Output: A near-optimal interdiction policy $\pi$
foreach iteration $i := 1, \dots, I$ do
    Let $\pi_i := \text{argmax}_{a \in A_x} \left\{ r_a(x) + \sum_{y \in \Omega} p_a(x, y) \tilde{V}^{\pi_{i-1}}(y) \right\}$
    for $s := 1, \dots, S$ do
        Sample state $x_s$
        Simulate $M$ trajectories starting in state $x_s$, using $\pi_i$
        Let $V_{s1}^{\pi_i}, \dots, V_{sM}^{\pi_i}$ be the simulated project completion times
        $\hat{V}_s^{\pi_i} := \sum_{j=1}^{M} V_{sj}^{\pi_i} / M$
    end
    Find $\tilde{V}^{\pi_i}(.)$ by a regression on $(x_1, \hat{V}_1^{\pi_i}), \dots, (x_S, \hat{V}_S^{\pi_i})$
end
```

Procedure SampleState

```
repeat
    $b_x$ is a random integer from 0 to $b$
    for $v$ :=1,...,$n$ do
        With equal probability place activity $v$ in $N_x$, $I_x$, $C_x$ or $D_x$
until $x$ is admissable
return $x$
```

methods we use. In this we let $\Omega_s \subset \Omega$ be the sampled states, the pairs $(x, \hat{V}^\pi(x)), x \in \Omega_s$ be the sampled states and their estimated values.

5.3.1 Least squares

Our first regression procedure is Least Squares where we use the basis functions given earlier $\phi(.)$ and solve the unconstrained problem for the weights θ

$$\underset{\theta}{\text{minimize}} \sum_{x \in \Omega_s} \left(\hat{V}^\pi(x) - \left(r_\pi(x) + \sum_{y \in \Omega} p_\pi(x, y) \theta^\top \phi(y) \right) \right)^2 \tag{5.4}$$

which has an analytical solution that is efficient to compute. Using the optimal weights θ^* we compute (5.1) for the value estimate.

Definition 1. *For the remainder of this chapter we represent the state x as an integer vector where $x_1 = b_x$ and the remaining elements $(x_i)_{i=2}^{n+1}$ correspond to each activity's state (0 = idle, 1 = active, 2 = delayed active, 3 = completed).*

5.3.2 Artficial neural networks

Our second way is to use Artificial neural networks (ANNs or NNs). Neural networks form a vast research topic and have been used successfully in many applications besides Dynamic Programming. We will only cover the basics to explain our method.

In general a NN is a non-linear function that is used for classification and regression in machine learning. The simplest neural network is just a linear regression model. More complex models are built out of neurons. Each neuron computes a function of the linear combination of its weighted inputs:

$$y = f\left(\sum_i \theta_i x_i\right) \tag{5.5}$$

Where f is a non-linear function such as the popular sigmoid activation function:

$$f(t) = \frac{1}{1+e^{-t}} \tag{5.6}$$

The output of one neuron can be the input to another and is called a signal. We will use only a certain type of NN called a *feed-forward* network where the signals travel in one direction. The NN can then be partitioned into layers where the first is usually called the input layer and receives external input. The last layer is called the output and the ones in between called hidden layers.

We will use our NN to estimate the value of being in state $\tilde{V}^{\pi}(x)$. To do this the idea is that we train it by giving input, output examples. So in our case these would be the pairs $(x, \hat{V}^{\pi}(x)), x \in \Omega_s$. With each training example, the NN uses an algorithm called *backpropagation* to minimize the error of its prediction against the actual output (like in linear regression). The difficulty with backpropagation is that it optimizes a non-convex function and so sometimes either does not converge or finds a local minimum. We therefore need to introduce a limit on the number of iterations and a performance threshold $\epsilon > 0$.

In our appplication each neuron in the input layer will receive each element of the state vector. While the output layer will consist of a single neuron that outputs a value o_x. Because the NN we use employs the sigmoid activation function, whose range is between 0 and 1, we need to scale the output by an amount W, so that our estimate is $\tilde{V}^{\pi}(x) = W o_x$. Similarly we divide the training outputs $\hat{V}^{\pi}(x)$ by W before sending them in. We will choose W to be a large enough number.

Neural networks are useful because they can fit arbitrary functions closely and can

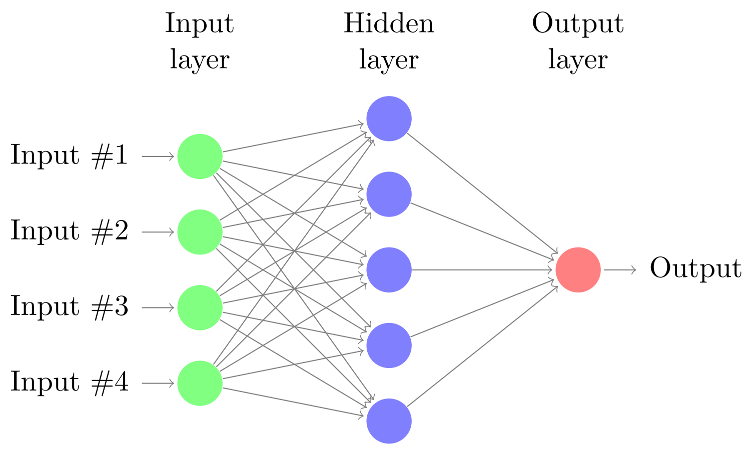


Figure 5.1: A three layer feed-forward NN, similar to the one that we use. Diagram thanks to http://www.texample.net/tikz/examples/neural-network/

be updated recursively. However these models suffer from other difficulties apart from their non-convexity. They can *overfit* the data, that is, they will fit the noise in data thus reducing their prediction accuracy and their predictions can have a high variance. This can be a significant problem with ADP which relies on Monte-carlo simulation.

5.3.3 Support vector regression

With support vector regression Smola and Scholkopf [2004] our goal is to find weights θ that approximate well the value of a state in (5.1). However we also want at the same time to reduce model complexity to address the problem of overfitting (where there is noisy data). The basic SVR problem can be stated as a Quadratic program

$$\begin{aligned} \underset{\theta,\xi,\xi^*}{\text{minimize}} \quad & \frac{1}{2}||\theta||^2 + C\sum_{x\in\Omega_s}(\xi_x+\xi_x^*) \\ \text{subject to} \quad & \theta^\top\phi(x) - \hat{V}^\pi(x) \le \epsilon + \xi_x \quad , x\in\Omega_s \\ & \hat{V}^\pi(x) - \theta^\top\phi(x) \le \epsilon + \xi_x^* \quad , x\in\Omega_s \end{aligned} \tag{5.7}$$

Where the slack variables ξ, ξ^* penalize deviation grater than ϵ from $\tilde{V}^\pi(x)$ the function we are trying fit, the term $\frac{1}{2}||\theta||^2$ penalizes complexity and C is used to trade off the two penalties. With a larger C and smaller ϵ, maximizing the accuracy in fitting the data is favoured over minimizing complexity.

The dual problem is

$$\begin{aligned}
\underset{\lambda,\lambda^*}{\text{maximize}} \quad & -\sum_{x,y\in\Omega}(\lambda_x^*-\lambda_x)(\lambda_y^*-\lambda_y)K(x,y)-\epsilon\sum_{x\in\Omega_s}(\lambda_x+\lambda_x^*) \\
& +\sum_{x\in\Omega_s}\hat{V}^\pi(x)\,(\lambda_x^*-\lambda_x) \qquad (5.8)\\
\text{subject to} \quad & 0\le\lambda_x,\lambda_x^*\le C \qquad ,x\in\Omega_s
\end{aligned}$$

where λ, λ^* are Lagrange mutipliers and $K(x,y) := \phi(x)^\top\phi(y)$ is a kernel function. The advantage of solving the dual is that we need not evaluate the inner product $\phi(x)^\top\phi(y)$ explicitly. Rather, all that is needed is a suitable kernel which lets us enjoy significant computational savings if there are many basis functions. The problem can be solved more efficiently using the SMO (sequential minimization optimization) algorithm Platt [1999]. Once the optimal λ, λ^* are found, the estimate is worked out as

$$\tilde{V}^\pi(x) = \sum_{y\in\Omega_s}\left(\lambda_y-\lambda_y^*\right)K(y,x) \tag{5.9}$$

This is also known as the support vector expansion. Another benefit of the dual problem is that it has only $2|\Omega_s|$ variables and because the problem is convex has a global optimum (if feasible and bounded) that can be found efficiently. Also the parameters C and ϵ give us a way to mitigate overfitting. A disadvantage of SVR based ADP methods is that there is no known way to perform it recursively unlike with least squares and ANNs.

5.3.4 SVR kernels

Our final task is to find a valid kernel $K(.,.)$. Fortunately there is already a test called Mercer's condition that allows us to do that:

The rule is that there exists a vector of basis functions ϕ such that $K(x,y) = \phi(x)^\top\phi(y)$ if and only if for all f where

$$\int f(x)^2dx$$

is finite (f is square-integrable) implies

$$\int K(x,y)f(x)f(y)dxdy \ge 0$$

These are some valid and popular choices of kernel:

- The polynomial kernel: $K(x, y) = \left(x^\top y + 1\right)^p$
- Radial basis function (RBF) kernel: $K(x, y) = \exp\left(||x - y||^2/\gamma\right)$
- Sigmoidal: $K(x, y) = \tanh\left(\kappa x^\top y - \delta\right)$

An interesting feature of the RBF kernel is that the basis functions that it's computed from are infinite dimensional. Intuitively, we can see this as the series expansion of e^x.

5.4 Non-simulation based SVR

We will attempt to apply a recent technique by B. Bethke and J. How and A. Ozdaglar [2008] to approximately solve our standard dynamic problem. This approach is highly model dependent but eliminates the need to perform simulations.

Unlike the techniques provided thus far we do not give state, value training examples. Rather we work with the MDP definition itself. The following function

$$B(x) = \theta^\top \phi(x) - \left(r_\pi(x) + \sum_{y\in\Omega} p_\pi(x, y)\theta^\top \phi(y) \right) \tag{5.10}$$

$$= -r_\pi(x) + \theta^\top \psi(x) \tag{5.11}$$

where $\psi(x) := \phi(x) - \sum_{y\in\Omega} p_\pi(x, y)\phi(y)$ is called the Bellman residual. The aim is to minimize $|B(x)|$ for as many sampled states possible. When $B(x) = 0$ for each sample, then our approximation

$$\tilde{V}^\pi(x) = \sum_{i=1}^{k} \theta_i\phi_i(x) = V^\pi(x)$$

is exact.

Bethke showed that when $B(x)$ is substituted into the nominal SVR it is equivalent to the following problem. Let ϕ_0 denote the 'feature' vector of some terminal state. The problem is stated as

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & \frac{1}{2}||\theta||^2 \\ \text{subject to} \quad & \theta^\top \psi(x) - r_x = 0 \quad x \in \Omega_s \\ & \theta^\top \phi_0 = 0 \end{aligned} \tag{5.12}$$

Note that for our problem we added just one more constraint that states that the value of the terminal state must be zero. Introducing multipliers λ and μ, the Lagrangian is

$$L = \frac{1}{2}||\theta||^2 - \sum_{x\in\Omega_s} \lambda_x(\theta^\top \psi(x) - r_\pi(x)) - \mu\theta^\top \phi_0 \tag{5.13}$$

As (5.12) is a convex problem, the necessary and sufficient optimality conditions are

$$\nabla_\theta L = \theta - \sum_{x\in\Omega_s} \lambda_i \psi(x) - \mu\phi_0 = 0 \tag{5.14}$$

$$\theta^\top \psi(x) - r_\pi(x) = 0 \tag{5.15}$$

$$\theta^\top \phi_0 = 0 \tag{5.16}$$

Using (5.14), (5.15) and (5.16), we solve for λ and μ as follows

$$\begin{pmatrix} \mathbb{K} & \psi \\ \psi^\top & 1 \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix} \tag{5.17}$$

Where $\psi := (\psi(x)^\top \psi_0)_{x\in\Omega_s}$ and $\mathbb{K}$ is the Gram matrix for $\{\psi(x)\}_{x\in\Omega_s}$. Using the solution to (5.17), the value approximation is calculated from

$$\begin{aligned}
\tilde{V}^\pi(x) &= \theta^\top \phi(x) \\
&= \left(\sum_{y\in\Omega_s} \lambda_y \psi(y) + \mu\phi_0 \right)^\top \phi(x) \\
&= \left(\sum_{y\in\Omega_s} \lambda_y \left(\phi(y) - \sum_{z\in\Omega} p_\pi(y,z)\phi(z) \right) + \mu\phi_0 \right)^\top \phi(x) \\
&= \sum_{y\in\Omega_s} \lambda_y \left(K(y,x) - \sum_{z\in\Omega} p_\pi(y,z)K(z,x) \right) + \mu K(0,x)
\end{aligned}$$

Where the first equality follows from (5.14). Bethke showed that when $\Omega_s = \Omega$, ,that is we sample the entire state space, the approximate solution becomes exact.

Chapter 6

Numerical evaluation

6.1 Nuclear project

These experiments are based on the case study from Brown et al. [2009]; Harney et al. [2006]. We will use them to test the algorithms in Chapter 3. Here we have a project to develop a small batch of nuclear weapons.

6.1.1 Case study data

The main milestones in the project are:

1. Diversion of 120 metric tonnes of yellowcake uranium
2. Enrichment plant feed material production
3. Enrichment with a choice of either gas centrifuge, gas diffusion and aerodynamic technologies
4. Conversion of uranium hexaflouride to highly enriched uranium
5. Missile design and assembly

In total there are 155 tasks with about 400 precedences. Each task normally takes a number of weeks to complete and consumes non-renewable resources.

There are 5 types of resource: professional, skilled and unskilled labour, energy and materials. They all have limited availability and also incur a $ cost per unit of use. We calculated from the data that the minimum required financial budget is around $159.8 million. Anything less and the project is unviable. Also with this budget, the project will take (a maximimum of) 412 weeks to complete - a little under 8 years.

Table 6.1: Proliferator's resources

Name	**Units**	**Unit cost (US $)**	**Available**
Energy	MWHr	100	3,100,000
Materials	$ 1000's	1,000	190,000
Professional labour	man-months	48,000	10,000
Skilled labour	man-months	24,000	10,000
Unskilled labour	man-months	6,000	10,000

The project manager crashes tasks and we model the deterministic task duration as

$$d_i = \overline{d}_i - e_i \tag{6.1}$$

where $e_i \in [0, \overline{d}_i - \underline{d}_i]$ is the expedited duration. There is a fixed resource cost for completing a task in its normal time. There is also an additional cost for expediting which is given by a quantity called the acceleration factor a_r for resource r:

$$a_{ir} = \frac{(a_r - 1)c_{ir}}{\overline{d}_i - \underline{d}_i} \tag{6.2}$$

Table 6.2: Acceleration factors

Name	**Factor**
Energy	2.0
Materials	1.2
Professional labour	1.2
Skilled labour	1.5
Unskilled labour	2.0

6.1.2 Data for the interdictor

An interdiction adds a fixed delay to the task duration. There are 12 types of interdictions $A, B, .., L$, each having a specified delay and cost. Every task has an interdiction type assigned to it.

Table 6.3: Cost index for the interdictor. Each task has an upper and lower case later assigned to it. The former is the delay in weeks that the interdiction would cause and the latter is the cost to the interdictor. For example, diffusion barriers has Fa which means that it could delay the project by 24 weeks and costs $ 200,000

Delay index	A	B	C	D	E	F	G	H	I	J	K	L
Delay (weeks)	4	8	12	16	20	24	28	36	40	48	56	60
Cost index	a	b	c	d								
Cost ($M)	0.2	0.45	1.2	1.7								

6.2 Implementation details

All LP/MILPSs were solved using CPLEX 12.1, a commercial software from IBM. The experiments in this section were run on a 2.93 GHz Intel Core i7 processor with 4 GB RAM.

To solve the models with the data from the case study, we produced a project spreadsheet. This could also be generated by exporting from popular project management software such as MS Project or OmniPlan [1]. We then developed a Java program to read the spreadsheet and either generate a `.lp` file, which would be read via the interactive CPLEX terminal, or call the solver directly from the application. It is possible and could be worthwhile to extend the program to read from a database or directly interface with the project management tools but this is beyond the focus of the dissertation.

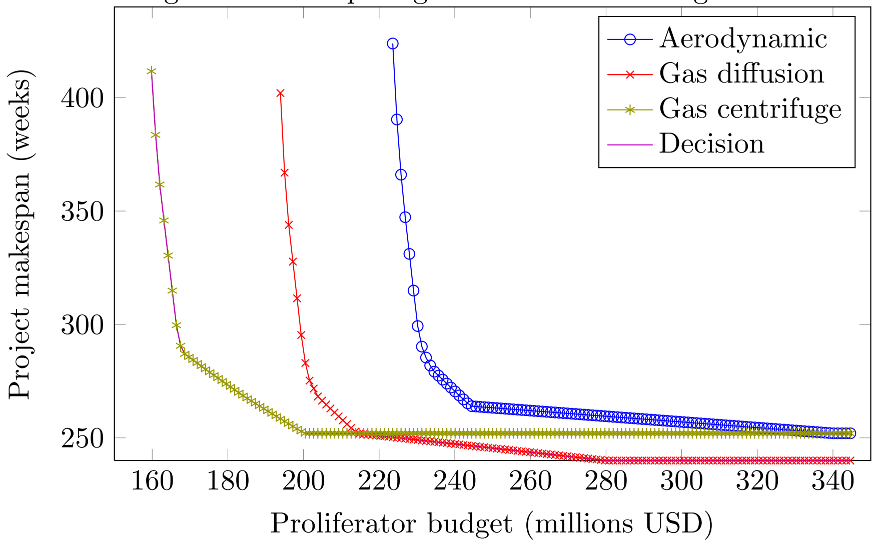
We also experimented with using a domain specific language from IBM called OPL (*optimization programming language*). This is an algebraic modelling language similar to GAMS and AMPL where the idea is to describe the mathematical model in a user-friendly way. The data is then combined with the description and a concrete problem is generated for the solver. However we chose to write our own custom translator as it seemed simpler and made debugging easier.

[1] Omniplan is a project analysis tool for the Mac OS http://www.omnigroup.com/products/omniplan

6.2.1 Discussion of results

We tested the PMIN, DPMIN, FPINT and DPINT models. The results from each model were consistent with each other. We first observed that the three enrichment technologies have different trade-offs. Gas centrifuge was the cheapest with a minimum project cost of \$159.8M and \$200M fully crashed but required 196 weeks. In contrast, the gas diffusion 'sub-project' could be completed in just 192 weeks but had significantly higher costs due to large amounts of energy needed to complete the final task to 'Produce enriched and depleted uranium'. Aerodynamic enrichment is least desirable, with the highest costs and having the same fully crashed duration as gas centrifuge. The results we obtained for DPMIN confirmed that it is the minimizer of PMIN over technology decisions.

Figure 6.1: Comparing enrichment technologies



The results for running DPINT showed that the proliferator's budget had modest impact on the effectiveness of interdictions (Figures 6.2 and 6.3) with a difference of about 40 weeks between a \$190 USD and \$300 USD project budget in Figure 6.2. This is because the proliferator could freely switch to other technologies in the static model and as we saw earlier the project completion time was similar between all three technologies although significantly cheaper for centrifuge enrichment.

In Figure 6.3, the project makespan went up rapidly from an interidction budget of 0 to \$3M. It then tapers off because the interdictor can only attack non-critical tasks. With an unlimited budget the interdictor will trivially delay everything.

Figure 6.2: Efficient frontier for an increasing number of interdictions from 0 to 8

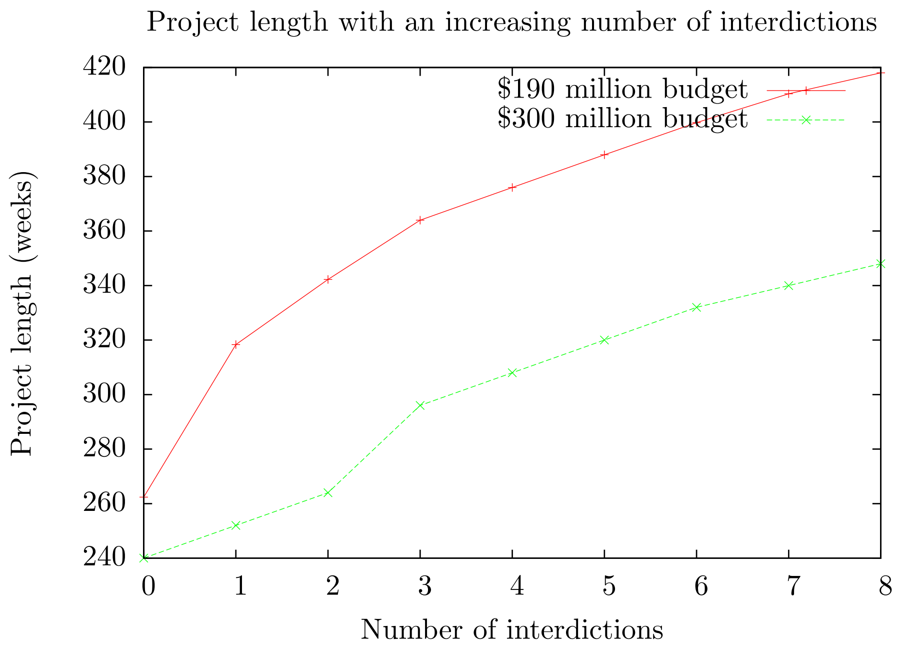
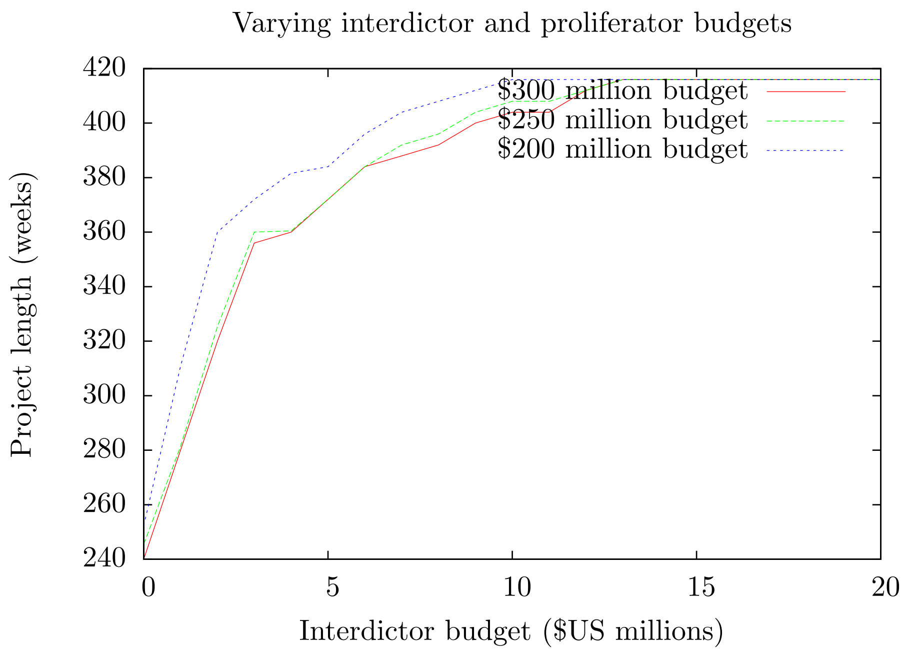
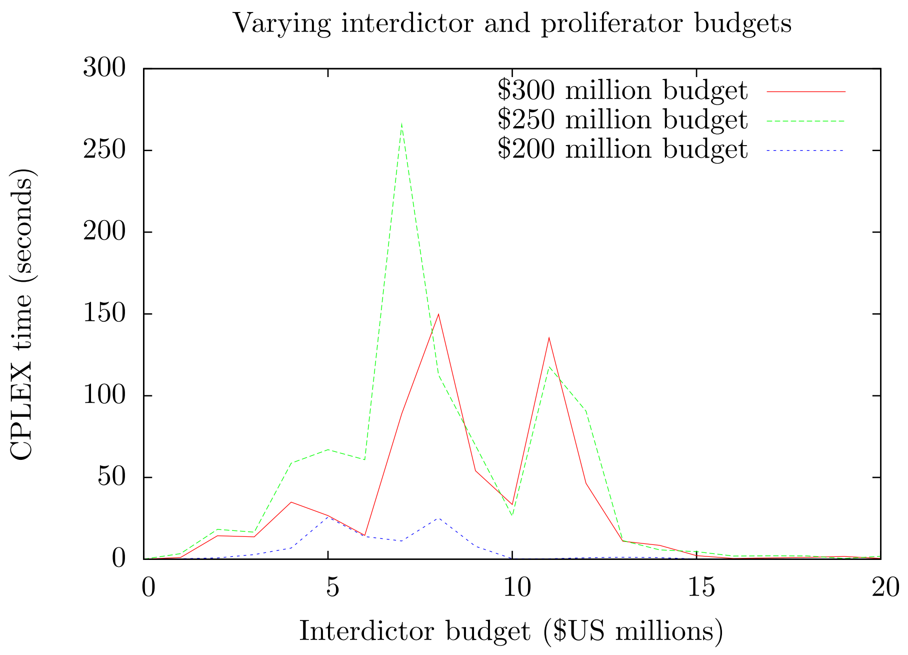


Figure 6.3: Efficient frontier for a varying interdiction budget



As expected and can be seen in Figure 6.4, DPINT has the fastest runtimes for either extremely low interdiction budgets or very high ones. The decisions at these extremes become easier to compute. The longest runtime was found for when the proliferator's budget was $250M and the interdiction budget around $6M. This was the most difficult scenario to find an optimal solution for. The reason is that at that point, the proliferator would fare equally well with either gas diffusion or centrifuge technologies as they had same duration and cost (given no interdictions). Thus he would have the most freedom to jump from one to the other and this made the interdictor's problem harder, as reflected in the spike of the CPU time to just under 5 minutes.

Figure 6.4: Computation time



6.3 Exact dynamic project interdiction evaluation

We now give experimental results of implementing the models in Chapter 4. We demonstrate how scalable the exact dynamic programming algorithms are and how much is theoretically gained from using this dynamic model.

First, we evaluate the simplest problem, where the project manager does *not* crash tasks and interdiction attempts are always successful. Afterwards we look at the problem with crashing allowed and finally at the one with binary implementation uncertainty. It turns out that these three versions are the only ones that can be solved on medium sized problems exactly. To address more elaborate set-ups, such as those where the measure of interdiction success is not necessarily yes or no (it could be continuous) and where activity durations are correlated, we would use approximate dynamic programming. We test the effectiveness of such algorithms in the next section.

6.3.1 Evaluation methodology

To answer the question of how much one gains from solving our dynamic problem over the nominal one, we use the following method due to the lack of historical data to back-test on. We evaluate the optimal dynamic policy by computing the mean time to absorption (equivalently the project makespan) of the Markov process that results from the policy's implementation. Let $q_{j|ia}$ be the rate of the activity that causes a transition $i \to j$.

$$m_k^\pi(i) = \sum_{a \in \mathcal{A}(i)} p_{ia}^\pi \left(\frac{m_{k-1}^\pi(i) + \sum_{j \in \Omega} q_{j|ia} m_k^\pi(j)}{\sum_{j \in \Omega} q_{j|ia}} \right) \tag{6.3}$$

where $m_k^\pi(i)$ is the k^{th} moment of the project makespan from state i, p_{ia}^π is the probability of executing action a in i with policy π and $m_0^\pi(i) := 0$ for all states i. The equations are solved backwards starting from the terminal state. Note that the project makespan has a phase-type distribution (see Appendix 1).

Afterwards, we can compare $m_k^\pi(0)$ to $m_k^{\hat{\pi}}(0)$ given by a sub-optimal policy $\hat{\pi}$, based on heuristics, by computing equation (6.3) in the same way.

There are 3 heuristics that we use. The first is a *pure* static policy where we select a set of 'target' activities $T \subseteq V$ by solving the nominal problem and interdict any task in T as soon as it begins. In real life, a better strategy would be to solve the same deterministic problem in every state but on a different PERT network consisting of the unfinished activities. We call such a strategy an *adaptive* static policy. However we will only use the latter on smaller problem instances because evaluating it on more complex examples becomes too computationally expensive because of the need to solve many MILPs.

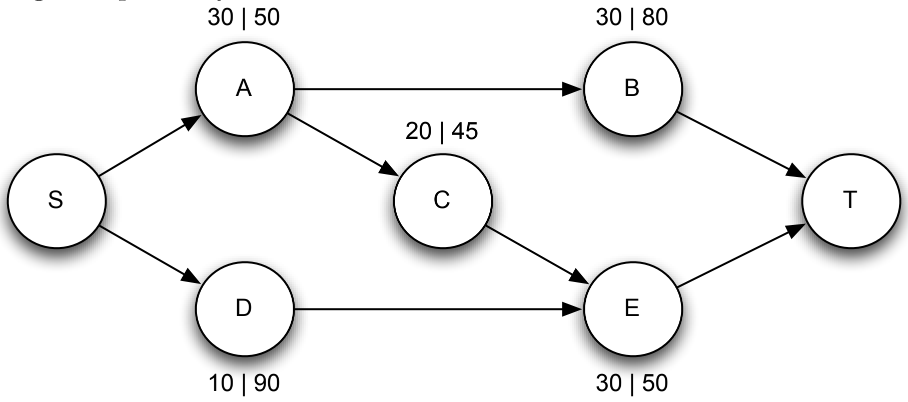
Lastly, we include a naive heuristic, called a greedy policy, which works by always delaying the most time-consuming tasks in any state whenever possible.

Table 6.4: Summary of interdiction policies

Policy	Description
Dynamic	Policy given by the dynamic program solution
Non-strict dynamic	Policy given by the randomized constrained MDP
Pure static	Always delay the same fixed set of tasks
Adaptive static	Chooses tasks to delay in a state by solving a subproblem
Greedy	Perform any admissable interdiction

6.3.2 Preliminary examples

Figure 6.5: Toy PERT network. The mean normal and interdicted durations are to the left and right respectively of the vertical bars



We start our analysis with hand-picked examples to compare, in more detail, the optimal dynamic and heuristic solutions.

Firstly, we consider the 5 activity PERT network from Figure 6.5. We see that with a budget of 3, the longest interdicted $s - t$ path is $S \to A \to C \to E \to T$ which, in the deterministic model, would increase the project makespan to $50 + 45 + 50 = 145$. Therefore, if we were to follow this reccommendation, we would, at the start of the project, interdict activity A. However, as Table 6.5 shows, doing so is sub-optimal (the optimal initial action being to interdict only D). Just this example is enough to show

Budget	Dynamic	Adaptive	Static	Greedy
1	139.76	139.76	139.76	109.35
2	162.24	162.24	158.62	152.51
3	183.56	167.25	149.74	167.16
4	196.74	179.25	149.74	191.20

Table 6.5: Mean project makespans with some policies from Table 6.4

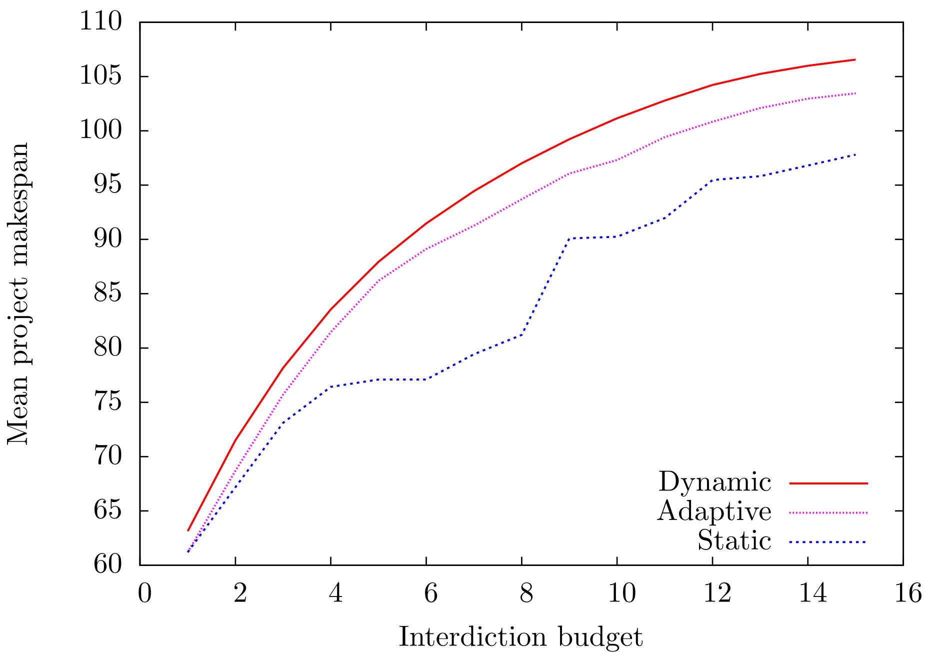
that there is a potential to improve on the nominal problem solution (i.e. the static policies).

As we would expect, we see from Table 6.5 that the mean project makespan is an increasing function of the budget for all policies. Surprisingly however, when the budget equals 3, the greedy policy outperforms the static solution by 22%. Furthermore, as soon as the budget increases to 4, the greedy policy outperforms *both* the static and adaptive policies, the latter by nearly 10%. Although such a finding does not apply to the majority examples (where the greedy policy *is* always dominated by its rivals), it does reveal something about the nature of the MILP solutions. Note how the longest interdicted path in the above network has 3 activites A, C and D. As soon as we increase our budget above 3, the pure static problem is unable to find any further improvements (i.e. any longer paths) and in essence, deteriorates, by suggesting actions that are no more optimal than they would be with a smaller budget.

Table 6.5 confirms that dynamic solution is the best. Indeed with a budget of 3, the dynamic solution exceeds its nearest rival's value, the adaptive static policy's, by almost 10%. This issue, as we will see later, can be more pronounced in networks exhibiting greater parallelism in their tasks' processing.

For the second of our chosen examples, we used a random network with 20 tasks. We made the budget range from 1 to 15 and the efficient frontier is shown in Figure 6.6. We see that the expected makespan, as a function of the budget, is concave for the optimal exact dynamic solution. On average, its value is 3.1% greater than the adaptive static solution's. The greatest difference between the two policies is just over 4% when the budget is 10. This may indicate that a dynamic approach is most valuable relative to the alternatives, when the budget is around half of the size of the network. On the other hand, if the budget is equal to zero or the number of tasks in the network, then all policies perform equally well.

Figure 6.6: Efficient frontiers for a network with 20 activities and order strength of 0.6



6.3.3 Large scale PERT networks

So far we demonstrated empirically that it is advantageous to solve the dynamic problem. In the best case we are able to produce a strategy that extends the mean duration of a project by as much as 10% above its rival policies (Figure 6.5).

The question we now answer is how scalable our dynamic programming algorithm is. We also explore if there is any relationship between characteristics of a PERT network and the value in using our algorithm.

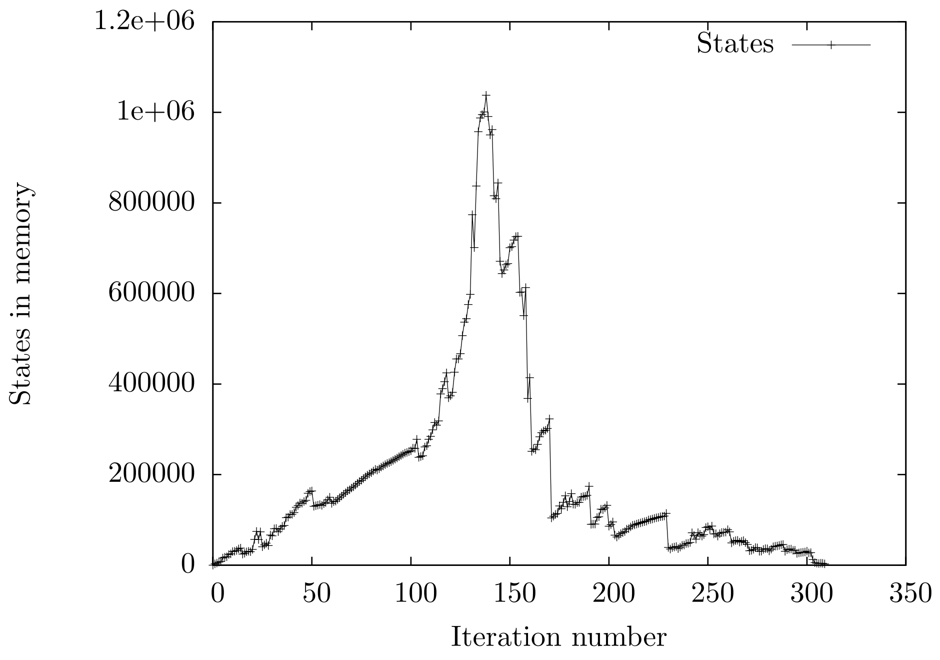
We tested our algorithm on 300 acyclic directed graphs that were produced by the RanGen software Demeulemeester et al. [2003]. This tool allows us to randomly generate a PERT network with a specified number of vertices and order strength. The second parameter is a measure of how difficult a PERT network G is to solve and is computed in the following way. Let n be the number of vertices in G and m the size of $\prec$, the order strength $O(G)$ is given by

$$O(G) := m / \binom{n}{2} \tag{6.4}$$

In other words it is the ratio of the number of transitive precedence relations over the maximum possible number.

6.3.3.1 Decomposition in action

Figure 6.7: Number of stored states with a total of roughly 2,700,000



Before we continue, we show that, as stated in Chapter 4, we reduce the memory requirements of solving the MDP by using the decomposition approach. Figure 6.7 shows the number of states stored at each iteration of the dynamic programming algorithm. On average only about 22% of states need to be stored at any time, while the maximum required peaked at 44% during the 137th iteration. The problem being solved in Figure 6.7 involved a PERT network with 80 tasks and an order strength of 0.8.

6.3.3.2 Performance evaluation on 300 PERT networks

For each order strength and network size we used 30 random instances (this was a similar approach to Creemers et al. [2010]) to work out sample average measures of CPU time, state space size and mean project makespans. To speed up the computation, we spread it between several but nearly identical machines (the `glyph` group in the Labs). Each one had a Dell Optiplex 980 Intel Core i5 650 3.2GHz processor and 8 GB of RAM. When running our Java process, we set the maximum JVM heap size to 2560 MB.

	Budget							
Size	1	2	3	4	5	6	7	8
10	0.125	0.118	0.122	0.122	0.120	0.118	0.123	0.129
20	0.331	0.325	0.332	0.340	0.349	0.355	0.352	0.368
30	0.769	0.793	0.828	0.842	0.886	0.909	0.921	0.951
40	1.439	1.542	1.645	1.775	1.940	2.062	2.187	2.325
50	3.026	3.222	3.566	3.991	4.490	5.043	5.573	6.076
60	5.859	6.456	7.273	8.563	10.17	11.95	13.84	15.74
70	4.351	5.786	9.253	16.61	28.98	46.60	67.51	90.67
80	10.89	14.43	23.06	42.43	80.97	125.4	179.6	244.8
90	16.30	25.37	50.92	110.4	218.5	379.0	580.3	802.7
100	21.23	39.26	107.6	241.4	555.2	1071.0	1809.1	2806.3

Table 6.6: Average CPU solution time in seconds over 30 random instances with 80% density

	Budget					
Size	1	2	3	4	5	6
10	77.0	146.2	217.0	287.8	358.6	429.4
20	348.2	739.7	1169.9	1604.6	2039.4	2474.2
30	1102.7	2560.3	4293.3	6090.6	7895.4	9700.6
40	3608.4	9410.0	17222.3	25850.8	34674.9	43525.0
50	8877.7	24773.9	47827.3	74497.3	102314.0	130355.4
60	20136.1	59572.9	120607.0	194435.6	273175.6	353144.1
70	57325.3	188955.3	422217.2	738365.1	1102313.3	1486371.1
80	114419.5	392637.7	907792.2	1630374.0	2481203.3	3389711.7
90	235703.3	839294.5	1998663.7	3667425.0	5661641.7	7804814.1
100	507836.1	1928983.3	4887529.3	9480422.5	15322732.3	21876561.5

Table 6.7: Average state space size over 30 random instances with 80% density

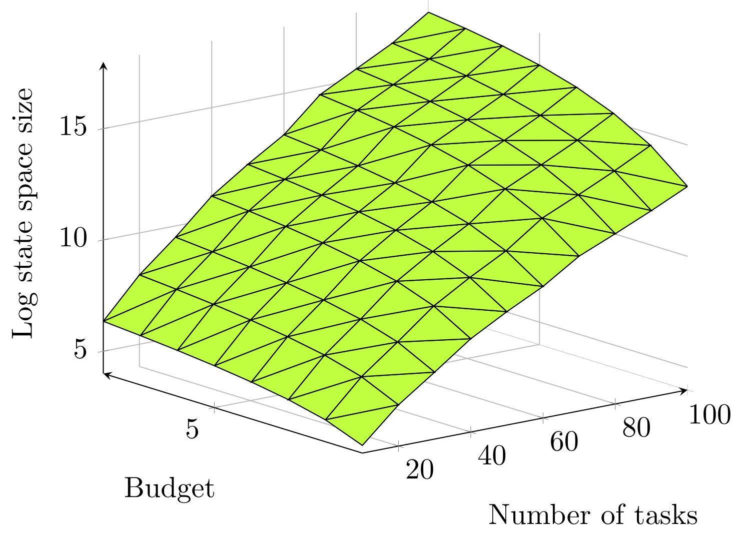


Figure 6.8: Average number of states versus network size and problem budget (on a log scale)

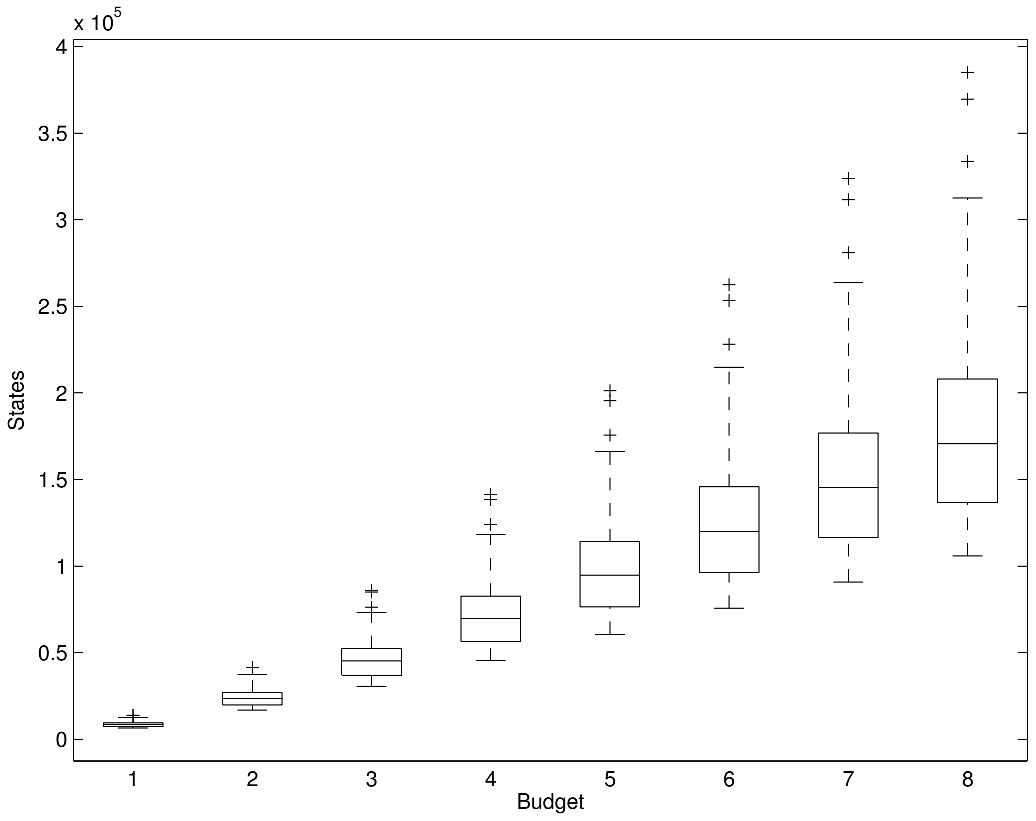


Figure 6.9: State space size with budget for networks with 50 activities and 80% density

	Budget	
Size	7	8
10	500.2	571.0
20	2909.0	3343.8
30	11505.8	13311.0
40	52376.6	61228.2
50	158421.8	186489.4
60	433306.9	513487.4
70	1876642.2	2268295.0
80	4317219.4	5249229.9
90	9997285.3	12201475.5
100	28738368.5	35702364.7

Table 6.8: Average state space size over 30 random instances with 80% density (continued)

	Budget							
Size	1	2	3	4	5	6	7	8
10	0.125	0.115	0.1181	0.126	0.122	0.1352	0.130	0.135
20	0.481	0.515	0.5503	0.613	0.679	0.7629	0.826	0.888
30	1.508	1.790	2.2946	3.190	4.364	5.8955	7.450	9.063
40	4.908	6.383	9.7705	16.58	27.57	41.850	58.10	74.99
50	3.026	3.222	3.566	3.991	4.490	5.043	5.573	6.076

Table 6.9: Average CPU solution time in seconds over 30 random instances with 60% density

	Budget			
Size	1	2	3	4
10	168.1	354.6	558.2	763.4
20	1746.6	4527.5	8240.0	12314.6
30	11990.3	36529.9	75909.8	124903.9
40	56401.3	188719.0	423769.8	739066.8
50	324109.2	1247660.5	3189663.6	6215531.0

Table 6.10: Average number of states over 30 random instances with 60% density

	Budget			
Size 4	5	6	7	8
10	968.6	1173.8	1379.0	1584.2
20	16470.3	20635.8	24801.8	28967.8
30	178014.1	232295.8	286792.1	341311.4
40	1096323.4	1468473.7	1844175.5	2220423.8
50	10057479.7	14346986.6	18817613.9	23342343.9

Table 6.11: Average number of states over 30 random instances with 60% density (continued)

	Budget							
Size	1	2	3	4	5	6	7	8
10	0.155	0.159	0.187	0.169	0.170	0.176	0.196	0.209
20	0.723	0.946	1.366	2.176	3.341	4.843	6.505	8.203
30	4.319	8.380	22.06	57.32	128.8	241.2	394.8	572.3

Table 6.12: Average CPU solution time in seconds over 30 random instances with 40% density

6.3.4 Discussion of results

For all order strengths, we see an exponential growth of both computational resources and state space size with increasing numbers of activities. However, keeping the network size constant, the relationship between the budget and number of states is almost linear (Figure 6.8 and 6.9).

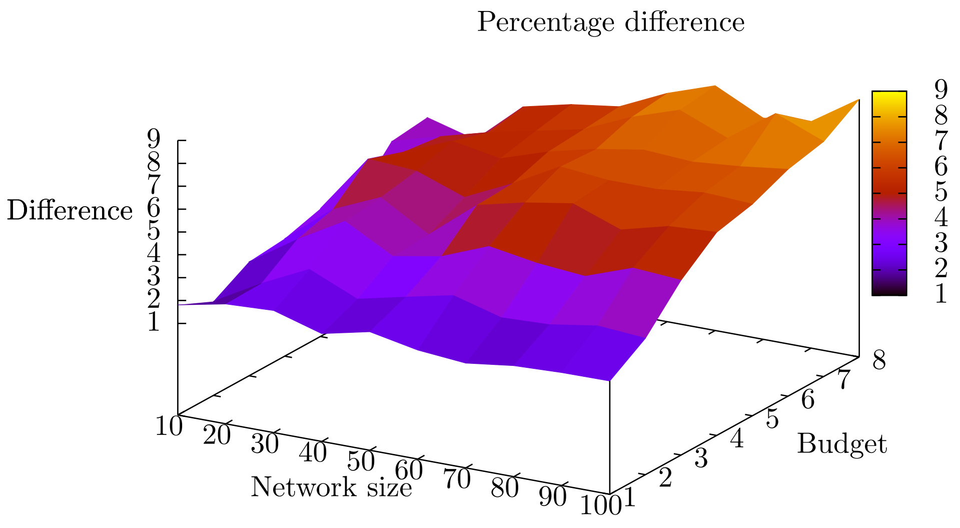
Despite the explosion in the problem complexity, we are able to solve non-trivial sized problems - networks with up to 100 tasks - provided that the order strength is high at 80%. In the best case we were able to solve problems with about 35 million states where the computation took 81 minutes. Beyond that, our program would run out of memory.

When implementing the algorithm, we decided to speed up the computation by using a state template 'cache' as described in Chapter 4. It is quite possible that were we to remove that, slightly larger problem instances could be solved due to the increased space available. However we did not have enough time to test this hypothesis.

We also see that order strength has a great effect on complexity. When the OS is 60% we can only solve [1] networks with up to 50 activities and a budget of 8. With 40% this drops to 40 and a budget of 3. Finally when the network has an OS of 20% , we could only solve all instances with 20 activities and a budget of 4. The reason for

[1]That means solving all of our 30 randomly generated instances

Figure 6.10: Average % improvement in value over a pure static policy



this is that low densities in PERT networks, all else being equal, allow more tasks to execute in parallel creating more possible states. Conversely, a higher order strength enforces a strict order of task execution and so fewer states.

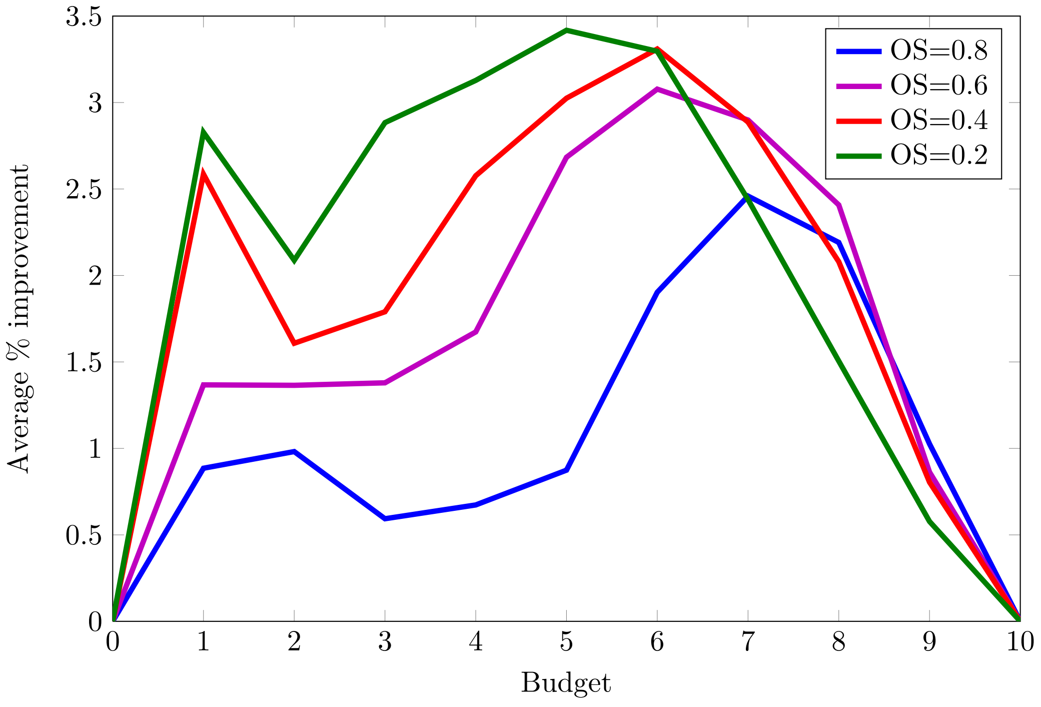
Figure 6.10 shows that the dynamic strategy gives the most advantage with larger networks and budgets for a fixed order strength (80% in this case). This might be because there are more states and therefore more information that the static model loses. The highest average improvement is about 8%.

Finally, we examine the average improvement over an adaptive static policy in Figure 6.11. As we predicted, the highest average improvement of 3.3% occurs when the order strength is lowest at 20% and the budget is around half the network size (5 to 6). Beside that, we see consistent pattern of improvement between all order strengths. The improvement first declines when the budget is 2-3, then sharply increases until the global maximum at 5 and finally drops off.

6.3.5 Interdiction with crashing evaluation

We have formulated and tested a dynamic interdiction problem on PERT networks. We will now evaluate the extension where the project manager is able to crash tasks. Because of the novelty of this problem, as we did in the basic dynamic problem's case,

Figure 6.11: Average % improvement in value over an *adaptive* static policy. Network size is 10.



we explain a suitable method of assessing the model and its solution algorithm.

Because there are now two players; the interdictor and project manager, we will generate a policy for each one. In software engineering terms, a policy is a lookup table. To speak of this in game theoretic terms, we now refer to the interdictor and project manager as the leader and follower respectively. The leader's lookup table is, as before, a mapping of states to a set of activities (to delay in that state), whereas the follower's is a mapping from states to a 'crashing function'. The crashing function maps each active or active delayed task in that state to an amount of each resource allocated to it. Given a policy π for the leader and ρ for the follower, we may compute the expected makespan of the project as a modification to equation (6.3). Let ξ^{ρ}_{ij} be the crashing function on an activity that causes a transition $i \to j$ and $q^{\rho}_{j|ia} := \xi^{\rho}_{ij} q_{j|ia}$, the k^{th} moment of the project makespan is given by

$$m^{\pi,\rho}_{k}(i) = \sum_{a \in \mathcal{A}(i)} p^{\pi}_{ia} \left(\frac{m^{\pi}_{k-1}(i) + \sum_{j \in \Omega} q^{\rho}_{j|ia} m^{\pi,\rho}_{k}(j)}{\sum_{j \in \Omega} q^{\rho}_{j|ia}} \right) \tag{6.5}$$

where $m^{\pi,\rho}_{0}(i) = 0$ as in (6.3).

6.3.6 Game with crashing evaluation

We tested the optimal policy's value against a pure static heuristic on 30 randomly generated networks having 10 tasks. In the best case we see an average improvement of 40% when the budget is 70% of the network size and the project's order strength is 0.4. As we would expect there is never any value gained in using a dynamic approach when the budget is either 0 or the size of the network.

However, in between the two extremes, we see that the lowest order strength networks have the greatest improvement for budgets below 5. We also find a surprising pattern where the improvement curves for the OS's peak at different budgets. Not only that but budgets beyond about 7 actually started to favour 0.4 OS networks over 0.2. This is also what we found with the standard game. It might be due to some law of diminishing returns where as soon as we are able to delay the majority of tasks, it becomes preferable for the task execution order to be *slightly* more rigid.

6.3.7 Interdiction with implementation uncertainty evaluation

To measure the implementation uncertainty game's optimal value over an adaptive static heuristic we used the following method: the deterministic problem would use the

Figure 6.12: Crashing game: Average relative advantage of using the optimal dynamic solution over a heuristic

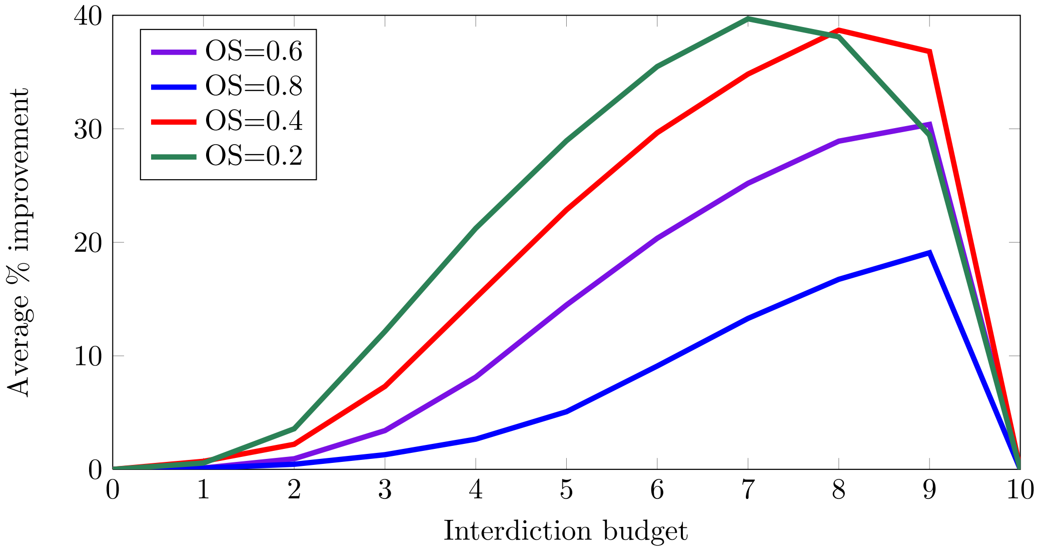
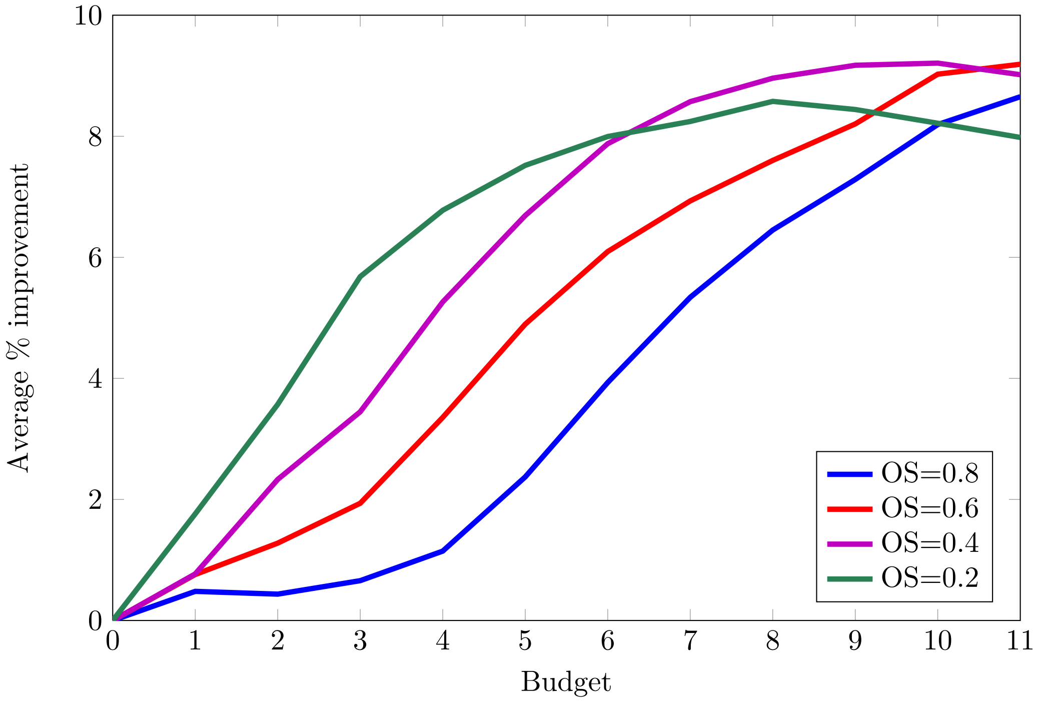


Figure 6.13: Implementation uncertainty game: This shows the average relative percentage improvement of solving the dynamic problem over the adaptive static policy. The PERT networks each had 10 tasks.



expected value of a task's interdicted duration and we would compute the game's value using another modification to (6.3).

The implementation uncertainty game was an interesting case because we could increase the budget *beyond* the size of the network and still see an advantage of using a dynamic policy. The reason being that on average only a fraction of interdiction attempts actually succeed - so there is still room for improvement, so to speak.

Comparing the standard problem with this one reveals that introducing an additional element of uncertainty makes the adaptive static solution even worse, in relative terms. Indeed the average improvement is about 3 times higher with implementation uncertainty than without.

6.3.8 Performance with extensions

The cost of using the crashing [1] and implementation uncertainty extensions is substanitally increased CPU time due to additional calculations done in each state - a hotspot of the basic algorithm in Chapter 4. However the storage requirements aren't affected as the same state space is used throughout.

If we were to do a linear regression [2] on the the points in Figure 6.14, we would find that the gradient is approximately 21 times greater with crashing than without (in the standard version). On the other hand, the implementation uncertainty game's gradient is about 13 times greater.

6.4 Approximate dynamic programming evaluation

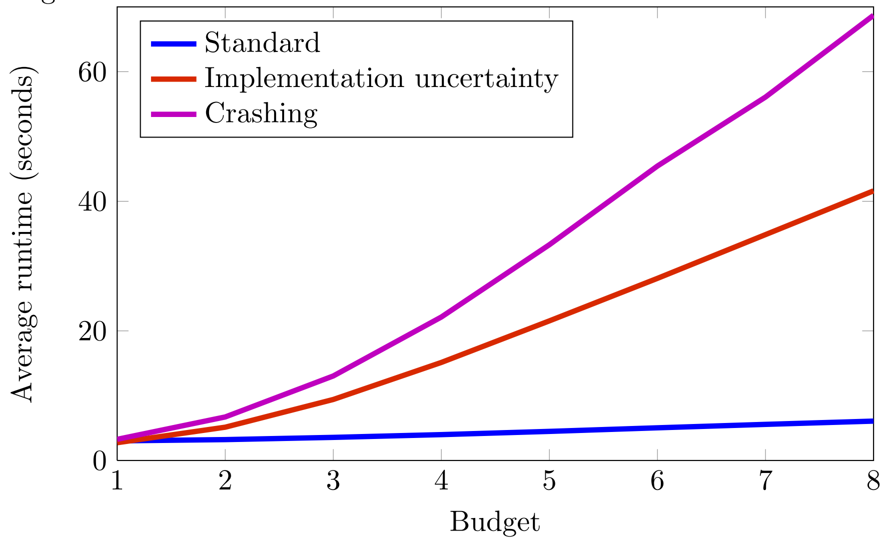
The exact dynamic programming algorithms have proven to be intractable. As soon as the order strength is less than 50%, only small examples can be solved with reasonable computational resources. On the other hand, the dynamic approach is more valuable with harder problem instances (where there is a larger state space). On top of that, the more complicated games with implementation uncertainty and crashing yielded a higher advantage to the optimal policy over other heuristics.

Therefore we now turn our attention to the approximate dynamic programming algorithms we discussed in Chapter 5. We test if they can provide better solutions compared to the nominal problem (the adaptive static policy). Also, we measure how sub-optimal such policies are. Finally, we attempt to understand how parameters influence the quality of ADP solutions. The algorithms we consider are least square

[1] The algorithm with a single crashing resource in Chapter 4

[2] Using the `SLOPE` function in Excel

Figure 6.14: The effect on CPU time of (i) letting the project manager to crash tasks (ii) adding implementation uncertainty. The networks all had 50 activities and the order strength was 80%



policy iteration, neural network-based policy iteration (ANN), support vector regression with simulation (SVR1) and Bellman residual SVR (SVR2).

We set up the ANN as a 3 layer perceptron with $n + 1$ input layer neurons, $2n + 2$ hidden layer neurons and 1 neuron in the output - where $n + 1$ corresponds to the dimension of the state vector (described in Chapter 5). Unfortunately, there was not enough time to test other neural network architectures. SVR1 had $\epsilon = 0.05$ and $C = 10,000$. For each simultion-based algorithm we set the number of simulated trajectories to be $M = 30,000$.

We start with an example to see more closely how the algorithms progress. Figure 6.15 shows the value of each algorithm's policy over 8 iterations on one PERT network. The number of samples taken was relatively large, about half the state space (which was possible with 10 tasks but does not scale). We see all of them shoot up in the first iteration and reach similar values of between 46-47. Then the ANN-based policy stops improving. The remaining policies, SVR1, SVR2 and least squares then proceed to overtake the adaptive static policy's value but never reach the optimum. The only exception to the last statement would be SVR2 because, when the entire state space is sampled, it is exact (because of no simulation noise).

We see that the support vector regression policies are marginally superior to the least square and that SVR1 improves slightly on SVR2. The reason possibly lies in the

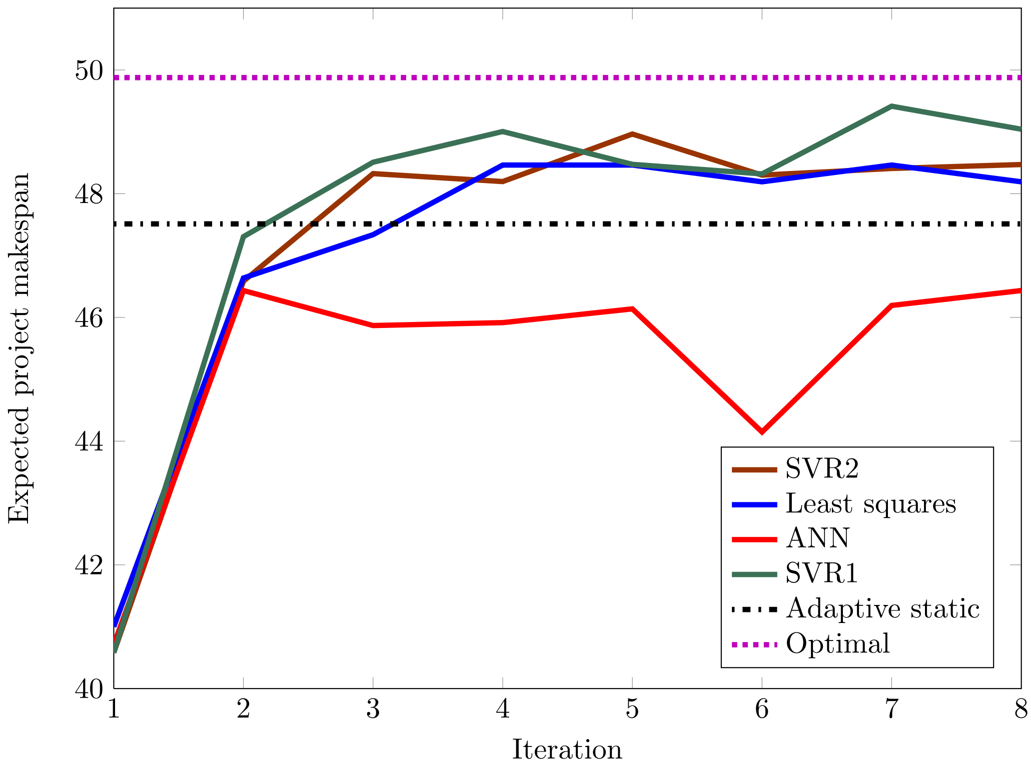


Figure 6.15: Interdiction game value after 8 iterations. The network had 10 tasks and its order strength was 0.4, the number of samples taken by each algorithm was 1000

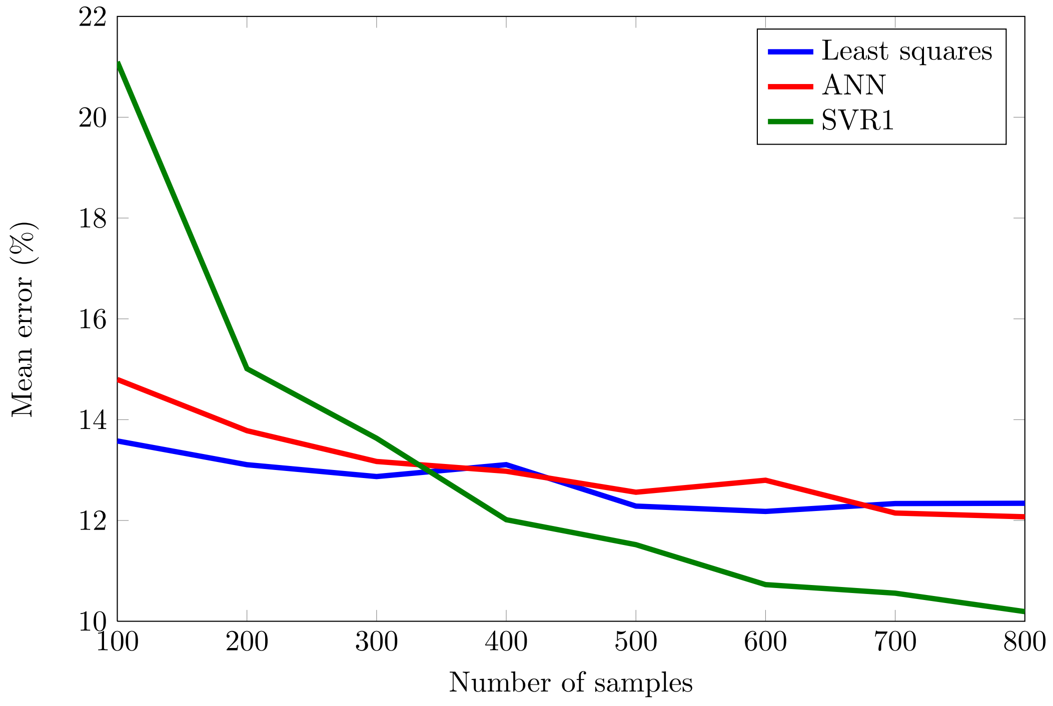


Figure 6.16: Mean absolute percentage error of the estimate $\tilde{V}$ on 10 networks each with 10 tasks and 0.4 order strength

values of the SVR1 parameters. ϵ being positive conferred a lower model complexity to SVR1 as compared to SVR2, which meant that SVR1 could better predict unobserved states' values. On the other hand SVR1 fitted the data more closely than least-square probably because of the implicitly infinite dimension of its data projection (as a result of the radial basis function kernel). Nearly all policies' values in Figure 6.15 converge after 4 iterations and then fluctuate without any improvement.

Next, we measured the *mean absolute percentage error* of the value estimates produced by the four algorithms (Figure 6.16) . The sample sizes constituted roughly 5%, 10% up to 40% of the state space. We did not try larger samples as they would not be used in practice. Unfortunately the SVR2 approach gave very large errors - around 90% when the sample sizes were in the given range. However above 1000 samples, we did consistently decrease the error and its solution was exact, as we predicted, when the whole state space was used. It could be that SVR2 is effective with a more specialized sampling strategy rather than the straightforward randomized one that we tried.

Figure 6.16 demonstrates that least squares has the lowest error with small sample sizes. This suggests to us that it has the most potential in practical applications (as

Table 6.13: Average % improvement over adaptive static policy and average % sub-optimality. The number of sampled states is in brackets and networks had 10 activities.

Algorithm		**Order strength** 0.2 (300)	0.4 (200)	0.6 (100)	0.8 (50)
Least squares	AS improvement	+1.2902	+1.0919	-1.3147	-1.0071
	Sub-optimality	3.2056	2.5346	3.6729	1.0206
ANN	AS improvement	-2.8821	-2.8926	-1.8164	-3.0245
	Sub-optimality	5.0870	4.6197	3.5365	3.7974
SVR2	AS improvement	-13.1498	-8.6819	-10.7159	-8.9035
	Sub-optimality	20.5348	13.6582	14.6776	9.9948

well as being solved more efficiently). Our prediction is confirmed in Table 6.13 where we see that least square is generally closer to the optimum than other policies [1]. It is also the only one which offers any advantage over the adaptive static policy (at least for 0.4 and 0.6 order strengths).

[1]We ran out time before being able to measure the average performance of SVR1. However on a few examples that we tried, it was inferior to least squares

Chapter 7

Conclusion

In this thesis we developed a new model for interdicting projects in continuous time under uncertainty. Our inspiration was an equivalent deterministic problem described in Brown et al. [2009].

First, we formulated the original model as a large scale MILP. Using standard solvers, we showed that it is computationally tractable provided that there are not too many technologies. Our method was faster than in the original paper, taking up to 5 minutes, in the worst case, as opposed to 20. We also were able to solve larger problem instances (using larger interdiction budgets) than Brown et al. [2009] reported. However we believe that their algorithm scales better when there are more technologies while there were just three in the case study that we tested on.

Later we formulated a novel continuous time MDP model of the interdiction game. We simplified it to only allow for a single technology, integer interdiction costs and no crashing. We then showed how it can be solved by reducing it to a discrete time MDP. This approach was based on Creemers et al. [2010]; Kulkarni and Adlakha [1986]. Next, we implemented a dynamic programming algorithm by Creemers et al. [2010] and optimized it. As long as the PERT network order strength was greater than 50%, we were able to solve medium sized problems with up to 60-100 tasks. Our algorithm failed to work for larger problems because of a lack of storage.

Next, we tested the optimal solution generated by our algorithm against other heuristics. It turned out that compared to the best heuristic, the one that solves the deterministic problem in every state, there was an improvement of about 1-3% on average. This figure was strongly correlated with the size of the problem i.e. the number of states, and peaked when the interdiction budget was around half the network size.

Then we extended the vanilla model by making crashing tasks possible. However this differed from Brown et al. because we had renewable as opposed to non-renewable resources whose supply stays constant. We then solved this game as a robust MDP, without affecting the state space and also showed a more efficient way to solve the follower's problem in linear, as opposed to polynomial, time. We reported average improvements over a heuristic as high as 40%.

Finally we developed another extension where an action's success could be uncertain. This involved a relatively small modification to the vanilla problem and also did not affect the state space. We showed that with higher budgets and more states, the mean improvement over the best heuristic is as high as 9%. Finally, we demonstrated that it is possible, in theory, to overcome the limitation of integer non-discounted interdiction costs by using a constrained MDP and solving this alternative model as a large scale LP.

To overcome the intractability of some of the models, we implemented and tested approximate dynamic programming algorithms. With these we made use of simulation and regression analysis. We also implemented an ADP algorithm by B. Bethke and J. How and A. Ozdaglar [2008] which did not rely on simulation, instead using a support vector machine to minimize the Bellman residual. Practically it turned out that LSPI (least square policy iteration) was best in terms of performance and scalability. In other words, even when sampling a small subset of states it yielded the best performing policies.

We argue that our interdiction game model is more realistic than previous approaches because we consider issues such as timing and uncertainty of successful interdiction. However our one has drawbacks: it is intractable, we had to make strong assumptions about independent and exponentially distributed task durations. It is possible to work around the last point by approximating task durations using phase-type distributions, adding the 'sub-Markov processes' into the MDP, and use approximate dynamic programming to mitigate the exponentially increasing computational burden.

7.1 Future work

A good extension is to introduce the multiple technology feature, that currently exists in the static model, to the dynamic game. It would probably would not require a major change of state space but would still be worthwhile as a proof of concept that the nominal problem could be replaced. An interesting exercise would then be to apply the complete model to the case study in Harney et al. [2006], probably using LSPI.

With regards to approximate dynamic programming, the Bellman residual method gave poor approximations of the value function and produced policies that were worse in comparison to other ADP algorithms. It is theoretically attractive however and it would be useful to find out what states need to be sampled to get the best performance out of it. It is our suspicion that the Bellman residual method works best on ergodic MDPs but this would need a proper investigation.

Finally, solving the models presented in this thesis was expensive computationally and it could be promising to experiment with high performance computing found in GPGPUs (general purpose graphics processing units) and other advanced hardware.

Appendix A

7.2 Phase type distribution

A phase type distribution (abbreviated as PH) Bolch et al. [1998] is a continuous distribution of the time until absorption in a transient Markov process. The PH is a generalization of exponential, Erlang and Coxian families of distributions and have been shown to approximate any distribution arbitrarily well (with enough states). There exists a discrete time counterpart of the PH distribution.

Let Q be the infinitisemal generator matrix of a Markov process, defined as

$$Q := \begin{pmatrix} 0 & 0 \\ -Se & S \end{pmatrix}$$

where $S \in \mathbb{R}^{n \times n}$ is called the sub-generator. Let α be a row vector specifying the pmf of the process's initial state. The MP's time to absorption T is said to be phase-type distributed $PH(\alpha, S)$ and its density function is

$$f(t) = -\alpha \exp\left(St\right) Se \tag{7.1}$$

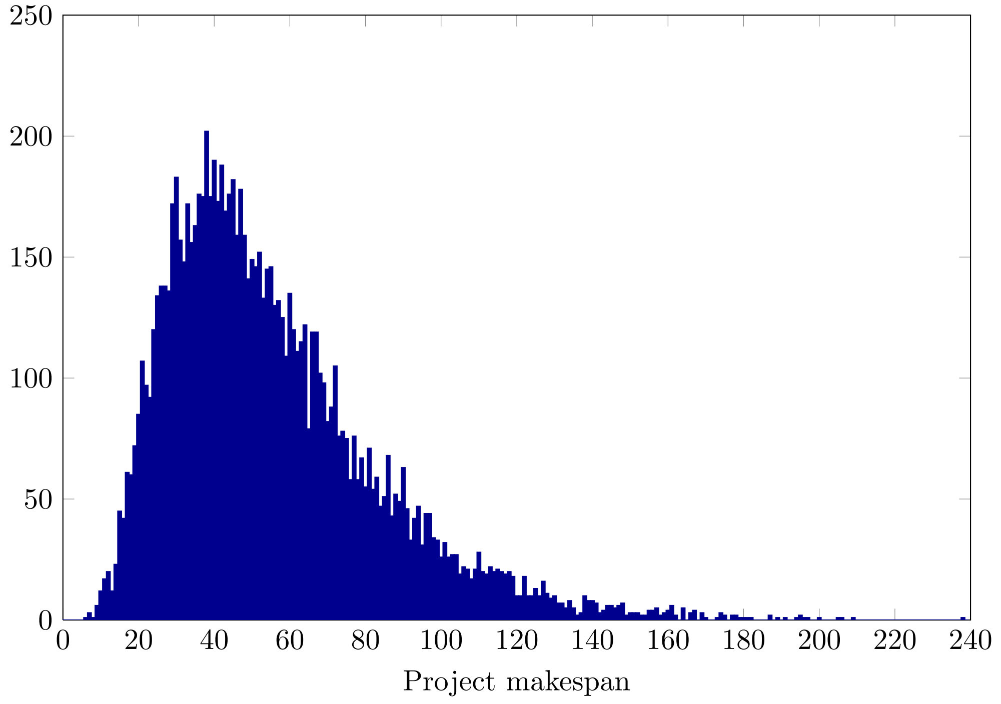
and its cdf is

$$F(t) = 1 - \alpha \exp\left(St\right) e \tag{7.2}$$

where $\exp(St)$ is the matrix exponential

$$\exp(St) = \sum_{k=0}^{\infty} \frac{(St)^k}{k!} \tag{7.3}$$

Figure 7.1: A histogram of 10,000 simulation runs of an interdicted project. The sample mean was 55.4 and the standard deviation was 28.25. The project's makespan is phase-type distributed.



References

B. Bethke and J. How and A. Ozdaglar. Approximate Dynamic Programming Using Support Vector Regression. In *IEEE Conference on Decision and Control (CDC)*, Cancun, Mexico, 2008. 50, 77

Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957. ISSN 0022-2518. 10

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108. 44

Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 2002. 15

Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience, New York, NY, USA, 1998. ISBN 0-471-19366-6. 79

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787. 38

Gerald G. Brown, W. Matthew Carlyle, Johannes Royset, and R. Kevin Wood. On the complexity of delaying an adversary's project. *The Next Wave in Computing, Optimization and Decision Technologies*, pages 3 – 17, 2005. 8, 36

Gerald G. Brown, W. Matthew Carlyle, Robert C. Harney, Eric M. Skroch, and R. Kevin Wood. Interdicting a nuclear-weapons project. *Oper. Res.*, 57:866–877, July 2009. ISSN 0030-364X. doi: 10.1287/opre.1080.0643. URL http://dl.acm.org/citation.cfm?id=1595844.1595851. 1, 8, 9, 17, 18, 19, 21, 36, 52, 76

A Charnes and W W Cooper. Programming with linear fractional functionals. *Naval Research Logistics Quarterly*, 9(3-4):181186, 1962. URL `http://www.springerlink.com/index/10.1007/BF02613374`. 38

Kelly J. Cormican, David P. Morton, and R. Kevin Wood. Stochastic network interdiction. *Oper. Res.*, 46:184–197, February 1998. ISSN 0030-364X. doi: 10.1287/opre.46.2.184. URL `http://dl.acm.org/citation.cfm?id=767679.768152`. 1, 8

Stefan Creemers, R Leus, and M Lambrecht. Scheduling markovian pert networks to maximize the net present value. *Operations Research Letters*, 38:51–56, 2010. 27, 31, 32, 33, 34, 62, 76

Erik Demeulemeester, Vanhoucke M, and Herroelen Willy. Rangen: A random network generator for activity-on-the-node networks. Technical Report urn:hdl:123456789/97356, Katholieke Universiteit Leuven, 2003. URL `http://ideas.repec.org/p/ner/leuven/urnhdl123456789-97356.html`. 61

Dolgov Dmitri and Durfee Edmund. Stationary deterministic policies for constrained mdps with multiple rewards costs and discount factors. In *Proceedings of the 19th international joint conference on Artificial intelligence*, IJCAI'05, pages 1326–1331, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1642293.1642504`. 42

S. Elmaghraby and R. Girish. Optimal resource allocation in activity networks under stochastic conditions. *Internal report. North Carolina State University*, 2010. 37

Robert Harney, Gerald Brown, Matthew Carlyle, Eric Skroch, and Kevin Wood. Anatomy of a project to produce a first nuclear weapon. *Science and Global Security*, 14(2-3):163–182, 2006. doi: 10.1080/08929880600993105. URL `http://www.tandfonline.com/doi/abs/10.1080/08929880600993105`. 22, 23, 52, 77

Frederick S. Hillier and Gerald J. Lieberman. *Introduction to operations research / Frederick S. Hillier, Gerald J. Lieberman.* Holden-Day Inc., San Francisco :, 1967. 6

Eitan Israeli and R. Kevin Wood. Shortest-path network interdiction. *Networks*, 40: 2002, 2002. 8

L.C.M Kallenberg. *Linear Programming and Finite Markovian Control Problems.* Math. Centrum, 1983. 42

James E. Kelley, Jr and Morgan R. Walker. Critical-path planning and scheduling. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '59 (Eastern), pages 160–173, New York, NY, USA, 1959. ACM. doi: http://doi.acm.org/10.1145/1460299.1460318. URL `http://doi.acm.org/10.1145/1460299.1460318`. 4

V. G. Kulkarni and V. G. Adlakha. Markov and markov-regenerative pert networks. *Operations Research*, 34:769–781, 1986. 27, 32, 34, 76

Churlzu Lim and J. Cole Smith. Algorithms for discrete and continuous multicommodity flow network interdiction problems. *IIE Transactions*, 2006. 8

Marco Lopez and Georg Still. Semi-infinite programming. *European Journal of Operational Research*, 180(2):491–518, 2011. 15

Joseph Moder and Cecil Phillips. *Project management with CPM and PERT [by] Joseph J. Moder [and] Cecil R. Phillips.* Reinhold Pub. Corp. New York, 1964. 4, 21

David P. Morton, Feng Pan, and Kevin J. Saeger. Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1):3–14, 2007. doi: 10.1080/07408170500488956. URL `http://www.tandfonline.com/doi/abs/10.1080/07408170500488956`. 1, 8, 9

John C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-19416-3. URL `http://dl.acm.org/citation.cfm?id=299094.299105`. 49

Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007. ISBN 0470171553. 44

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994. ISBN 0471619779. URL `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&amp;path=ASIN/0471619779`. 10

Adam J. Rudolph and S. Elmaghraby. The optimal resource allocation in stochastic activity networks via continuous time markov chains. *Internal report. North Carolina State University*, 2007. 37

Andres Shipani. Cocaine production rise spells trouble for bolivia. http://www.bbc.co.uk/news/10231343, 2010. Accessed: 30/09/2010. 1

Alex J Smola and Bernhard Scholkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004. ISSN 0960-3174. doi: 10.1023/B:STCO.0000035301.49549.88. URL http://dx.doi.org/10.1023/B:STCO.0000035301.49549.88. 48

Matthew J. Sobel, Joseph G. Szmerekovsky, and Vera Tilson. Scheduling projects with stochastic activity duration to maximize expected net present value. *European Journal of Operational Research*, pages 697–705, 2009. 27

Frederik Stork and Marc Uetz. On the generation of circuits and minimal forbidden sets. Open access publications from maastricht university, Maastricht University, 2005. URL http://EconPapers.repec.org/RePEc:ner:maastr:urn:nbn:nl:ui:27-4895. 32

Anabela P. Tereso, Araujo M. Madalena T., and Elmaghraby Salah E. Experimental results of an adaptive resource allocation technique to stochastic multimodal projects. *International Conference on Industrial Engineering and Production Management*, 2003. 37

Anabela P. Tereso, Araujo M. Madalena T., and Elmaghraby Salah E. Adaptive resource allocation in multimodal activity networks. *International Journal of Production Economics*, 92(1):1–10, November 2004a. URL http://ideas.repec.org/a/eee/proeco/v92y2004i1p1-10.html. 37

Anabela P. Tereso, Araujo M. Madalena T., and Elmaghraby Salah E. The optimal resource allocation in stochastic activity networks via the electromagnetism approach. *Project Management and Scheduling*, 2004b. 37

Richard Wollmer. Removing arcs from a network. *Operations Research*, 12(6):pp. 934–940, 1964. ISSN 0030364X. URL http://www.jstor.org/stable/168177. 8

R. K. Wood. Deterministic Network Interdiction. *Mathematical and Computer Modelling*, 17(2):1–18, 1993. doi: 10.1016/0895-7177(93)90236-R. URL http://dx.doi.org/10.1016/0895-7177(93)90236-R. 8