ABSTRACT

Title of Thesis: RESOURCE-BASED SPIKE MITIGATION STOCHASTIC CONTROL PROBLEMS

Keith McCready, for Master of Science in Electrical Engineering, 2013

Thesis directed by: Professor Gilmer Blankenship
Department of Electrical and Computer Engineering

Resource-based spike mitigation stochastic control problems are a class of stochastic control problems, akin to inventory control problems or linear quadratic regulator (LQR) problems. These problems involve consuming a resource to mitigate large, fast losses to a primary state ("spikes"). These properties, included with stochastic elements, draw out a unique behavior where optimal control policies conserve resources during "lucky streaks" and spend the resources during "unlucky streaks." However, these problems often have too many time steps and states to compute an optimal control policy with dynamic programming. This thesis gives examples of such problems and demonstrates how to effectively approximate solutions using suboptimal control methods.

RESOURCE-BASED SPIKE MITIGATION STOCHASTIC CONTROL PROBLEMS

by

Keith McCready

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2013

Advisory Committee:

Professor Gilmer Blankenship, Chair
Professor William Levine
Professor Nuno Martins

ACKNOWLEDGMENTS

Thanks to Professor Steve Marcus for

his help with Stochastic Control

Table of Contents

List of Tables

List of Abbreviations

RBSM: Resource-Based Spike Mitigation

LQR: Linear Quadratic Regulator

CEC: Certainty Equivalence Control

1 Introduction and Motivation

The Resource-Based Spike Mitigation (hereafter RBSM) Stochastic Control Problem Set is a class of discrete time stochastic control problems, akin to inventory control problems or stochastic linear quadratic regulator (LQR) problems, which has interesting control properties when certain features come together into one problem. We will first describe these features in detail, explaining what conditions a problem must meet for it to fit into this class. After that, we will summarize why these features combine to produce interesting results. These resultant properties make this problem class somewhat unlike most other problems generally studied in control engineering.

First, by "resource-based," we mean that some or all control values require and consume a resource when used. Let us call this resource "Energy." In the simplest problems, Energy regenerates over time, potentially stochastically. It may also be generated by control or due to the current state.

Second, "spike mitigation" refers to what we are trying to control and the nature of the cost function. Problems in this class keep track of the amount of some resource lost at each time step (or gained, represented with a negative number). Let us call this resource "Health." The cost incurred at each time step is an exponential function of recent Health lost. That is, the problem has a time window length (e.g. 4 time steps). First, one sums the losses that occurred in a time window. This sum is called a "spike." Next, one applies the exponential function to every spike to compute

the cost for that window, then sums those results over all possible time windows to compute the total cost. The exponential cost means that spikes of higher magnitude are exponentially more costly (e.g. a set of two spikes of 0 and 100 is much more costly than a set of two spikes of 50 each). If it were a linear cost, the problem would not be quite so interesting, as we will see.

The exponential cost on spikes combined with control values that require Energy create a situation where Energy can be "pooled," meaning it can be saved to use on control to mitigate the largest spikes. This is where the "stochastic" part of the title comes in as the third and final important property. When stochastic elements are involved, some stochastic events will contribute to higher costs and some to lower costs. When lower-cost events occur several time steps in a row, this results in smaller spikes. Similarly, when higher-cost events occur several time steps in a row, this results in larger spikes. Stochastic events allow optimal control policies to pool Energy during small spikes and spend it during large spikes in order to reduce total cost.

These three features define the RBSM Stochastic Control Problem Set. As long as the problem follows these guidelines loosely, it will fall into the RBSM control problem class. Problems that follow these guidelines strictly will have more interesting control behavior, as we will show. Problems can have infinite possible control values, but it becomes more difficult to solve, which we also discuss later.

We will define these features mathematically in the next section, establishing a loose model that all RBSM problems fit. Following that, we will examine three

examples of RBSM Stochastic Control Problems. In section four, we will explain why suboptimal control methods are necessary and use them to obtain a reasonable approximation to the solution of one of the examples. We will also see how to extend that knowledge to other problems. Finally, we will summarize the behavior one can expect to see in all RBSM Stochastic Control Problems.

2 RBSM Stochastic Control Problem Model

We will now mathematically describe the properties that all problems in this class share. It is prudent to say here that there are some gaps in the model which cannot be filled in a general model, such as some components of the state evolution or permissible control values. When this occurs, a description of the missing components will be included in curly brackets.

As mentioned in the introduction, we will not actually be tracking Health itself, only the losses to Health as affected by control. Put simply, the rationale for this is that it is not necessary or useful to do so. The cost function of a RBSM Stochastic Control Problem already measures the ability of a control policy to maintain Health at high levels. Observing Health and total Health gains directly actually leads to a control problem to which the solution is either trivial or impossible. More detail can be found in the Appendix section A.1.

To begin, let $t \in \mathbb{Z} \cap [0, T]$, where $T \in \mathbb{Z}$ is the ending time of the system. Let $E[t] \in [0, \text{MaxE}]$ be the Energy of the system at time t, where MaxE is the maximum Energy. MaxE can potentially be positive infinity (in which case $E[t] \in [0, \infty)$ to be precise). The evolution of $E[t]$ can be stochastic or deterministic. Its typical evolution is the current Energy plus any Energy gained minus any Energy lost.

Let $u[t]$ be the control value selected at time t, selected from the set of permissible control values at time t, $U(\{\text{state at time } t\}, t)$. Typically, U is simply the set of control values for which the minimum Energy cost is met. However, in the next

section, we will show that this does not have to be the case. Possible controls vary by problem, but all controls must be causal. If there are a finite number of control values, they may be enumerated.

Let $X[t] = [x_1[t] \quad \cdots \quad x_{\text{WindowSize}}[t]]' \in \mathbb{R}^{\text{WindowSize}}$ be the last WindowSize losses to Health at time t, where WindowSize $\in \mathbb{Z}_+$ is the length of the problem's spike window size, in time steps. The health loss at time t is $x_1[t]$, and values are pushed down the vector every time step. The Health losses can be influenced by controls in various ways, depending on the problem. That is, the probability distribution of the Health losses is determined by controls, including the values of the losses. We define the ways in which controls manipulate the loss of Health to be called "control effects."

It is convenient for many RBSM Stochastic Control Problems to define $L[t] \in \mathbb{R}$ to be a random variable representing the loss of Health that would occur at time $t + 1$ in the absence of some (or all) control effects, as an intermediate variable in the state evolution. This allows the control effects of most (or all) controls on $X[t + 1]$ to be conveniently defined in terms of $L[t]$, and the control effects of remaining control values can be accounted for in the probability distribution of $L[t]$. $L[t]$ is generally selected stochastically each time step, but it can sometimes be deterministic. As is typical in stochastic control, $L[t]$ is unknown for control purposes; it cannot be used in selecting $u[t]$.

Thus, the system state is typically $(E[t], X[t])$. However, some control effects will introduce additional state variables that interact with controls and other state

variables, or possibly even $L[t]$.

Finally, we present the problem model:

$$\text{Minimize expected total cost } V(X[\cdot]) = \mathbb{E}\left[\frac{1}{T}\cdot\sum_{t=1}^{T}(\text{SDF})^{10(y[t]-1)}\right]$$

where

$$y[t] \stackrel{\text{def}}{=} \frac{1}{\text{MaxH}}\sum_{i=1}^{\text{WindowSize}} x_i\,[t], \qquad \forall t;$$

$$\text{SDF} \in (1,\infty);\ \ \text{MaxH} \in \mathbb{R}_+;$$

$$x_1[t+1] = \{\text{Health lost at time } t+1\};$$

$$x_k[t+1] = x_{k-1}[t], \qquad k = 2, 3, \ldots \text{WindowSize};$$

$$E[t+1] = \min(\text{MaxE}, E[t] + \{\text{Energy gained at time } t\} - \{\text{Energy lost at time } t\});$$

$$X[0] \in \mathbb{R}^{\text{WindowSize}};\ \ E[0] \in [0, \text{MaxE}].$$

If $L[t]$ is included, the evolution of x_1 instead becomes

$$x_1[t+1] = \begin{cases} L[t], & \{\text{with no control effects}\} \\ \text{some function } f(L[t]), & \{\text{with control effects}\} \end{cases}$$

where $L[t] \sim \{\text{Distribution depending on problem and possibly control effects}\}$.

Note also that this means the cost incurred at time t is $T^{-1}(\text{SDF})^{10(y[t]-1)}$.

Let us explain the form of the cost function. First, note that $y[t]$ is simply an intermediate variable defined for convenience. It is the spike magnitude at time t divided by MaxH. The scalar MaxH, which stands for "Maximum Health," is a

parameter that scales the size of all spikes proportionally. It is a reference point by which to evaluate the magnitude of a spike. Typically, if a system has a maximum Health, MaxH is set to the maximum Health of the system, hence its name. The scalar SDF, which stands for "Spike Decade Factor," controls the exponential strength of the function. The name comes from the fact that a spike of magnitude a will be SDF times less costly than a spike of magnitude $a + 0.1 \cdot \text{MaxH}$. This is also where the "10" in the cost function originates. For example, suppose the spike magnitude at time t is MaxH. Then $y[t] = 1$, and the cost incurred at time t is $(\text{SDF})^0 = 1$. Suppose instead that the spike magnitude at time t is $1.1 \cdot \text{MaxH}$. Then $y[t] = 1.1$, and the cost incurred at time t is $(\text{SDF})^{10(0.1)} = \text{SDF}$. Note that it does not make sense for SDF to be less than or equal to 1. If SDF is 1, then all spikes have cost 1 regardless of magnitude. If SDF is less than 1, then spikes of higher magnitude actually cost exponentially less. Lastly, the T^{-1} term in front simply normalizes the cost by the ending time of the problem.

In summary, the above cost function exponentially penalizes larger spikes. As mentioned, the spikes here are simply the sum of the last few Health losses. This is similar to a reverse-time exponentially decaying cost function, but places much more emphasis on the last few time steps and much less emphasis on time steps that occurred a long time ago. Other cost functions can actually cause the same properties seen in RBSM Stochastic Control Problems. Another similar cost function would be one that looks Gaussian, which would soften the otherwise sharp cutoff of the time window sum, but still maintain the same general shape.

In stochastic control, control may be chosen with the knowledge of the current state, all past states, and all past controls. Note, though, that the chosen time window cost function does not require infinite information to be stored as time progresses, unlike any functions without a cutoff. Functions without a cutoff also would not behave uniformly over all time steps. The time window cost function is also a direct measurement of the Health lost within a time window, so it measures something of which one can make intuitive sense.

Its main disadvantage, though, is the sharp cutoff at WindowSize time steps. This does not seem to be a major issue in practice, since every time step is still accounted for evenly. However, it does mean that optimal control policies can and usually will change significantly depending on WindowSize.

3 Example RBSM Stochastic Control Problems

In this section, we present examples of RBSM Stochastic Control Problems. We will not solve these problems; rather, this section solidifies understanding of the problem model presented in the previous section. Further, it demonstrates the potential variety of RBSM Stochastic Control Problems. The discussion of how to solve RBSM Stochastic Control Problems is in the next section.

3.1 Example Problem 1 (Storm Damage Prevention and Recovery)

This problem is about managing storm damage to a region. Suppose some infrequent, powerful storms will cause massive physical damage to this region that costs money to repair. Thus, in between storms, it is reasonable that one would want to prepare for the next major storm, perhaps by saving money for repairs or spending money on preventative measures. It is reasonable to assume that a damaged region can receive relief help from other regions, which justifies using the RBSM stochastic control problem model.

Let us state the mathematical definition of the problem. Note that we must now also concretely define U, the set of permissible controls. There are also two new state variables: $P[t], R[t] \in \mathbb{Z}_+, \ \forall$t. The problem is defined as such:

$$\text{Minimize expected total cost } V(X[\cdot]) = \mathbb{E}\left[\frac{1}{T} \cdot \sum_{t=1}^{T} (\text{SDF})^{10(y[t]-1)}\right]$$

where

$$y[t] \stackrel{\text{def}}{=} \frac{1}{\text{MaxH}} \sum_{i=1}^{\text{WindowSize}} x_i\,[t], \qquad \forall t;$$

$$\text{SDF} \in (1, \infty); \;\; \text{MaxH} \in \mathbb{R}_+;$$

$$u[t] \in U(E[t], R[t])$$

$$= \begin{cases} \{\text{CtrlWait}\}, & (E[t] = 0)\text{or}\big((0 < E[t] < \text{CPrevn})\text{and}(R[t] \geq \text{RcvrValid})\big) \\ \{\text{CtrlWait, CtrlRcvr}\}, & (0 < E[t] < \text{CPrevn})\text{and}(R[t] < \text{RcvrValid}) \\ \{\text{CtrlWait, CtrlPrevn}\}, & (E[t] \geq \text{CPrevn})\text{and}(R[t] \geq \text{RcvrValid}) \\ \{\text{CtrlWait, CtrlPrevn, CtrlRcvr}\}, & (E[t] \geq \text{CPrevn})\text{and}(R[t] < \text{RcvrValid}) \end{cases};$$

$$x_1[t+1]$$

$$= \begin{cases} L[t] \cdot (\text{PrevnAmt})\text{^}P[t], & u[t] = \text{CtrlWait} \\ L[t] \cdot (\text{PrevnAmt})\text{^}(P[t]+1), & u[t] = \text{CtrlPrevn} \\ L[t] \cdot (\text{PrevnAmt})\text{^}P[t] - \text{RcvrAmt} \cdot \min(\text{CMaxRcvr}, E[t]), & u[t] = \text{CtrlRcvr} \end{cases};$$

$$x_k[t+1] = x_{k-1}[t], \qquad k = 2, 3, \ldots \text{WindowSize};$$

$$L[t] = \begin{cases} 0, \text{ w.p. Avoid} \\ \text{StormDmg, w.p. } 1 - \text{Avoid} \end{cases};$$

$$P[t+1] = \begin{cases} 0, & L[t] > 0 \\ P[t], & L[t] = 0 \text{ and } u[t] \neq \text{CtrlPrevn} \\ P[t]+1, & L[t] = 0 \text{ and } u[t] = \text{CtrlPrevn} \end{cases};$$

$$R[t+1] = \begin{cases} 0, & L[t] > 0 \\ R[t]+1, & L[t] \leq 0 \end{cases};$$

$$E[t+1] = \min\left(\text{MaxE}, \begin{cases} E[t] + \text{RegenE}, & u[t] = \text{CtrlWait} \\ \text{RegenE}, & (u[t] = \text{CtrlRcvr}) \text{ and } (E[t] < \text{CMaxRcvr}) \\ E[t] - \text{CMaxRcvr} + \text{RegenE}, & (u[t] = \text{CtrlRcvr}) \text{ and } (E[t] \geq \text{CMaxRcvr}) \\ E[t] - \text{CPrevn} + \text{RegenE}, & u[t] = \text{CtrlPrevn} \end{cases}\right);$$

$$\text{CPrevn, CMaxRcvr, RcvrAmt, RegenE, StormDmg} \in \mathbb{R}_+;$$

$$\text{RcvrValid} \in \mathbb{Z}_+; \quad \text{PrevnAmt}, \text{Avoid} \in [0, 1];$$

$$X[0] \in \mathbb{R}^{\text{WindowSize}}; \quad E[0] \in [0, \text{MaxE}]; \quad P[0], R[0] \in \mathbb{Z}_+.$$

Although the above essentially can stand alone, the complexity of the system's evolution likely warrants some intuitive explanation. In this problem, Energy represents money. The regeneration of Energy (at a rate "RegenE") represents the taxation income of the region. Suppose this has low enough variance that estimating it as deterministic at its mean is acceptable. Health losses in this problem ($L[t]$) represent the general destruction of buildings, trees, power lines, and so forth. We could keep track of the status of each item individually, but that problem is much more complicated. Instead, suppose we can generate one number that is representative of the entire region.

As for the controls, one is the ability to spend money on repairing the region after it is damaged, which we call "Recover" ("CtrlRcvr" above, which stands for "Control: Recover"). This only makes sense to do if the region is actually damaged, so for demonstrative purposes, suppose it is only permissible in the next "RcvrValid" time steps following a major storm (non-zero $L[t]$). This is easily done by keeping track of the time since a storm with a new state, $R[t]$, the number of time steps since a major storm. Let "RcvrAmt" be the amount of Health restored by Recover per Energy spent. Suppose only a maximum of "CMaxRcvr" Energy can be spent on this control in a single time step.

Another control is physical preventative maintenance, which we call "Prevention" ("CtrlPrevn" above, which stands for "Control: Prevention"). This reduces the Health loss of the next major storm. In general, this would cost money, but not very much. Let its cost be "CPrevn." This control option represents trimming or removing trees, reinforcing buildings, replacing old or damaged telephone poles, etc. This preventative maintenance would be less effective every time it is used in succession, so suppose it is applied multiplicatively; that is, if using it once reduces the Health lost from the next major storm by $100(1-0.9)\% = 10\%$, suppose using it again before a storm will reduce it by $100(1-0.9^2)\% = 19\%$, and so on. Let us call the number of times preventative maintenance has been used its "stacks," which we will represent and keep track of with a new state, $P[t]$. Suppose the stacks reset to 0 after every major storm, mainly for demonstrative purposes. Let "PrevnAmt" be the percentage by which to multiply hits taken under one stack of Prevention. For example, with four stacks and PrevnAmt $= 0.9$, the normal damage of a major storm would instead be multiplied by $0.9^4 = 0.6561$.

The final control is called "Wait" ("CtrlWait" above, which stands for "Control: Wait"). This control spends no Energy, does not directly affect P or X, and does not introduce any parameters.

Note that all new parameters mentioned thus far have been related to controls or Energy. The specification of $L[t]$ can also introduce new parameters. "Avoid" is the probability that no storm damage will occur at any given time step, and "StormDmg" is the amount of Health lost in the event of a major storm.

This example is remarkable in that the cost of the controls can change the optimal control in interesting ways. Naturally, the general tendency of optimal control policies will be to build up a few stacks of Prevention then save Energy for Recover once Prevention is less Energy efficient at reducing cost. If the cost of Prevention is lowered, then the optimal control will, of course, prefer more stacks of Prevention. If its cost is lowered below RegenE, then there is also a concern of hitting MaxE later if one decides not to use Prevention. This would be unfavorable, since it would waste Energy that otherwise could have been used on Prevention. The optimal control would weigh the probability of that occurring with the probability that it won't, since if MaxE is not reached, that Energy would have been useful for a more powerful Recover. This behavior arises because the system does not always have a way in which to spend excess Energy.

3.2 Example Problem 2 (Video Game Defender)

Some video games present their players with very difficult stochastic control problems as a main component of the game, such as Tetris[1]. The example discussed here involves a player defending against one enemy attacking the player. It is a simplified version of an actual video game. Although it is simplified, the important elements are present so that the interesting control policy behavior can still be observed. The unsimplified version is just much more needlessly cumbersome to present here as an example.

Let us now state the mathematical definition of the problem. As in Example 1, we must again concretely define U, the set of permissible controls. This problem has one additional state variable: $P[t] \in \mathbb{Z}_+, \ \forall$t. The problem is defined as such:

$$\text{Minimize expected total cost } V(X[\cdot]) = \mathbb{E}\left[\frac{1}{T} \cdot \sum_{t=1}^{T} (\text{SDF})^{10(y[t]-1)}\right]$$

where

$$y[t] \stackrel{\text{def}}{=} \frac{1}{\text{MaxH}} \sum_{i=1}^{\text{WindowSize}} x_i\,[t], \qquad \forall t;$$

$$\text{SDF} \in (1, \infty); \ \ \text{MaxH} \in \mathbb{R}_+;$$

$$u[t] \in U(E[t]) = \begin{cases} \{\text{CtrlWait}\}, & E[t] = 0 \\ \{\text{CtrlWait, CtrlRcvr}\}, & 0 < E[t] < \text{CMtgt} \\ \{\text{CtrlWait, CtrlMtgt, CtrlRcvr}\}, & E[t] \geq \text{CMtgt} \end{cases};$$

$$x_1[t+1]$$

$$= \begin{cases} L[t], & u[t] = \text{CtrlWait and } P[t] = 0 \\ \text{MtgtAmt} \cdot L[t], & u[t] = \text{CtrlMtgt or } (u[t] = \text{CtrlWait and } P[t] > 0) \\ L[t] - \text{RcvrAmt} \cdot \min(\text{CMaxRcvr}, E[t]), & u[t] = \text{CtrlRcvr and } P[t] = 0 \\ \text{MtgtAmt} \cdot L[t] - \text{RcvrAmt} \cdot \min(\text{CMaxRcvr}, E[t]), & u[t] = \text{CtrlRcvr and } P[t] > 0 \end{cases};$$

$$x_k[t+1] = \ x_{k-1}[t], \qquad k = 2, 3, \ldots \text{WindowSize};$$

$$L[t] = \begin{cases} 0 \text{ w.p. Avoid} \\ \text{WeakHit w.p. Weak} \\ \text{StrongHit w.p. } 1 - \text{Avoid} - \text{Weak} \end{cases};$$

$$P[t+1] = \begin{cases} 0, & P[t] = 0 \text{ and } u[t] \neq \text{CtrlMtgt} \\ P[t] - 1, & P[t] > 0 \text{ and } u[t] \neq \text{CtrlMtgt} \\ P[t] - 1 + \text{DurMtgt}, & u[t] = \text{CtrlMtgt} \end{cases};$$

$$E[t+1] = \min(\text{MaxE},$$

$$\left\{ \begin{matrix} E[t] + \text{RegenE}, & u[t] = \text{CtrlWait} \\ \text{RegenE}, & (u[t] = \text{CtrlRcvr}) \text{and} (E[t] < \text{CMaxRcvr}) \\ E[t] - \text{CMaxRcvr} + \text{RegenE}, & (u[t] = \text{CtrlRcvr}) \text{and} (E[t] \geq \text{CMaxRcvr}) \\ E[t] - \text{CMtgt} + \text{RegenE}, & u[t] = \text{CtrlMtgt} \end{matrix} \right\} \Bigg);$$

$$\text{CMtgt, CMaxRcvr, RcvrAmt, RegenE, WeakHit, StrongHit} \in \mathbb{R}_+;$$

$$\text{DurMtgt} \in \mathbb{Z}_+; \quad \text{MtgtAmt, Avoid, Weak} \in [0,1]; \quad \text{Avoid} + \text{Weak} \leq 1;$$

$$X[0] \in \mathbb{R}^{\text{WindowSize}}; \quad E[0] \in [0, \text{MaxE}]; \quad P[0] \in \mathbb{Z}_+.$$

Again, the above is the complete problem, but the complexity of the system's evolution warrants intuitive explanation. First, the actual names of Health and Energy in this game are not important, so let us just use those default names. The idea in this problem is that one player of a co-operative team can force enemies to attack him instead of his comrades. The enemy's attacks can be weak, strong, or avoided completely, so $L[t]$ randomly takes one of three values. The largest loss is called a "strong hit," the weaker loss is called a "weak hit," and a zero loss is called an "avoid." These Health losses are also referred to as "damage." The player uses his abilities to reduce the rate at which he takes damage, which is the control.

Comrades are able to heal the damage taken by the player, but they also have other allies that will need healing regularly. Thus, the RBSM stochastic control problem model measures the self-sufficiency of the defending player. There is essentially a cost on how much attention the player requires from healers. That is,

the cost function represents the player's incentive to avoid large Health spikes, which would potentially give the healers too much to heal at once, causing player deaths.

In this problem, the player builds Energy steadily over time (at a rate "RegenE" per time step). The player can spend it to reduce the Health lost for the next few time steps ("Mitigate") or to recover lost Health ("Recover"). Mitigate is "CtrlMtgt" above, which stands for "Control: Mitigate." Recover is "CtrlRcvr" above, which stands for "Control: Recover." To discuss them in detail, Mitigate multiplies the next "DurMtgt" Health losses by "MtgtAmt." In lieu of storing past control values, we store the number of remaining time steps of mitigation in a new state variable, $P[t]$. Mitigate requires and consumes "CMtgt" Energy.

The other important control is Recover. Recover can be used with non-zero Energy and consumes all Energy up to a maximum of "CMaxRcvr" Energy. It adds a negative number onto the next Health loss equal to "RcvrAmt" multiplied by the amount of Energy consumed.

The final control is again called "Wait" ("CtrlWait" above, which stands for "Control: Wait"). This control spends no Energy, does not directly affect P or X, and does not introduce any parameters.

$L[t]$ is slightly more complicated in this example compared to the last, so we have more parameters. "Avoid" is the probability that $L[t] = 0$, and "Weak" is the probability that the player will lose "WeakHit" Health. The player loses "StrongHit"

Health with the remaining probability.

Since Recover and Mitigate are somewhat similar in the effect they have on total cost, the optimal control behavior that arises in this problem depends heavily on the relative magnitudes of parameters. If the balance of parameters is done well, a behavior arises where Mitigate is usually preferred, but Recover is useful when the enemy connects with strong hits several times in a row. Further, Energy is pooled during smaller spikes in order to enable that. This behavior arises because Mitigate has its effect spread over the next few time steps, where Recover's effects occur immediately. Therefore, Recover is more useful to decrease the value of a spike towards the end of its formation by a large amount, but Mitigate potentially affects more spikes (it does not if Avoids occur during its effect).

There are, additionally, other players in this game who have different controls available. For example, some players have the ability to increase $L[t]$'s "Avoid" probability temporarily instead of decreasing the damage taken directly, or similarly, increase its "Weak" probability. Others have the ability to negate the first X damage taken in the next Y time steps. All of these are also RBSM stochastic control problems, but listing separate examples for all of them would be excessive.

3.3 Example Problem 3 (Dam Operation)

Again, it does not have to be $L[t]$ which is stochastic. The regeneration of $E[t]$ alone having stochastic elements can also cause similar behavior (though nothing

prevents both from being stochastic). Here, we have an example of a dam, where the flow through the dam is steady and under our control, but the water above the dam is subject to stochastic elements from storms and droughts. A dam does not as precisely fit the RBSM stochastic control model, since normally we would also worry about the top of the dam overflowing, but this example is mainly for illustrative purposes.

Let us now state the mathematical definition of the problem. This problem does not have any additional state variables. The problem is defined as such:

$$\text{Minimize expected total cost } V(X[\cdot]) = \mathbb{E}\left[\frac{1}{T}\cdot\sum_{t=1}^{T}(\text{SDF})^{10(y[t]-1)}\right]$$

where

$$y[t] \stackrel{\text{def}}{=} \frac{1}{\text{MaxH}}\sum_{i=1}^{\text{WindowSize}} x_i\,[t], \qquad \forall t;$$

$$\text{SDF} \in (1,\infty);\ \ \text{MaxH} \in \mathbb{R}_+;$$

$$u[t] \in U = [0,1], \qquad \forall t$$

$$x_1[t+1] = \text{LossH} - \max(\text{MaxFlow}\cdot u[t], E[t])\,;$$

$$x_k[t+1] = x_{k-1}[t], \qquad k = 2,3,\ldots \text{WindowSize};$$

$$E[t+1] = \min\left(\text{MaxE},\ \begin{cases} E[t] + \text{RegenDrought} - \max(\text{MaxFlow}\cdot u[t], E[t])\,, & \text{w.p. Drought} \\ E[t] + \text{RegenStorm} - \max(\text{MaxFlow}\cdot u[t], E[t])\,, & \text{w.p. Storm} \\ E[t] + \text{RegenNorm} - \max(\text{MaxFlow}\cdot u[t], E[t])\,, & \text{w.p. } 1 - \text{Storm} - \text{Drought} \end{cases}\right);$$

$$\text{LossH, MaxFlow, RegenDrought, RegenStorm, RegenNorm} \in \mathbb{R}_+;$$

$$\text{Storm, Drought} \in [0, 1];\ \ \text{Drought} + \text{Storm} \leq 1;$$

$$X[0] \in \mathbb{R}^{\text{WindowSize}};\ \ E[0] \in [0, \text{MaxE}].$$

This example is easier to grasp immediately than the others, but since this is a new class of problems, let us examine it closely regardless. In this example, "Health" represents the water level below the dam, which will be constantly draining deterministically, and "Energy" represents the water level above the dam, which is regenerated randomly. The Health losses in this problem are deterministic because water below the dam flows away at a constant rate unaffected by the weather. Our goal is to keep the water level below the dam from fluctuating wildly over time despite this constant drain. Since the cost function only encourages lessening the largest spikes, the water level below the dam does not actually matter. This may seem counter-intuitive, but lessening the largest spikes serves to keep the water below the dam close to a mean value, rather than e.g. allowing it to dry out one time step then overflow a few time steps later. The control in this problem is the flow through the dam.

As mentioned, the regeneration of Energy is stochastic in this problem. However, the probability distribution is not complicated. The Energy lost at each time step is determined only by the current Energy and the control. The Energy gained is one of three values. The Energy gain is "RegenDrought" with probability

“Drought,” “RegenStorm” with probability “Storm,” and “RegenNorm” with the remaining probability.

The possible control values are all the possible values the dam can have between “fully closed” and “fully open.” These are represented as control values of 0 and 1, respectively. This means that this problem, unlike the other examples presented so far, has infinite control values. The control determines how much Energy we allow to replenish Health, which is converted at a ratio of one-to-one, since it reflects the movement of water. The parameter “MaxFlow” determines how much Energy is consumed per time step with the dam fully opened.

This problem does not use an $L[t]$ variable. There is little point, since the losses of Health at each time step are deterministic. The loss of Health at each time step is the parameter “LossH.”

Interestingly, it seems that LossH does not have much of an effect on the optimal control compared to RegenE of the other examples. It merely affects the baseline loss of Health: it is a Health loss offset by a static number, which only serves to increase the cost by an exponent of SDF. To check this, note that if LossH were 10 instead of 20, every four-step window would reduce its spike by exactly 40. Suppose MaxH is 100 and SDF is 3. Then note that, due to the design of the cost function, 60 is three times more costly than 50, and 20 is three times more costly than 10. Basically, the overall cost and the cost at each time step would only simply be multiplied by 3^{-4}. Therefore, the cost will change by changing LossH, but the optimal control will not.

4 Solving a RBSM Stochastic Control Problem

In this section, we will study Example 2 from the previous section (the video game defender). It is chosen because it has the most interesting behavior of the examples given. It has two different methods to spend Energy. One is preventative control that lasts multiple time steps, and one is weaker overall but happens instantly. If MaxE is strictly greater than CMaxRcvr and CMtgt, the problem allows for limited pooling of Energy during small spikes and expenditure of Energy during large spikes. The hope is that, with careful tuning of parameters, the optimal control will be Recover after several Strong Hits in a row (some mitigated by Mitigate), but otherwise the optimal control will be to save Energy and use Mitigate around MaxE Energy, except perhaps during very low spikes, such as several Avoids in a row. It is for this behavior that we will scrutinize Example 2. If one could find those thresholds given the parameters, it would be useful in both designing and playing the game.

Most real-world RBSM stochastic control problems, and all the examples given in this paper, run for a very large number of time steps. If they did not run for a large amount of time steps, the problem wouldn't necessarily last long enough to see the Energy-conservation behavior. They also have a large quantity of possible states, if not infinitely many. Even with a small number of control choices, perfect state information, known probability distributions, and a small number of possible stochastic events, it quickly becomes evident that Dynamic Programming or framing as a Markov Distribution Problem are not feasible computations. For example, with

600 time steps, 3 control choices, and 3 stochastic events, a problem would have at most $(3 \cdot 3)^{600} = 3.51 \times 10^{572}$ possible paths to analyze with dynamic programming. Clearly, this is computationally infeasible. It seems that, as also with many real-world problems[1], we are forced to use suboptimal control methods to arrive at a policy that is close to optimal.

Since we will "solve" by suboptimal control, it is useful to have numeric values of all parameters. We will use typical values for parameters found within the game that the problem is based on.

T= 400
MaxE= 5
WindowSize= 4
MaxH= 100
SDF= 3
Avoid= 0.15
Weak= 0.2
StrongHit= 25
WeakHit= 18
RegenE= 1
CMtgt= 3
CMaxRcvr= 3
MtgtAmt= 0.6
RcvrAmt= 5

Enumeration of controls for simulation purposes:

ControlWait= 0
ControlMitigate= 1
ControlRecover= 2

4.1 Certainty Equivalence Control

After assessing the options, the most obvious method to reduce computation was to try Certainty Equivalence Control (CEC). The Certainty Equivalence Control method is to approximate the optimal control of a stochastic control problem by assuming all stochastic elements always take their mean values, then solving for the optimal control of that deterministic control problem. This method turned out to be, for the most part, useless. However, it is interesting why this is true, and we gain a fair bit of intuition from investigating it, including that CEC is not going to be very helpful for any RBSM stochastic control problem. As mentioned in the introduction, stochastic elements are actually necessary to see the Energy conservation behavior, and trying CEC suboptimal control makes that more clear.

Using the "typical" values, Example 2 under CEC means that every time step always has a loss of about 20 Health units. This means that, once initial conditions for $P[t]$ and $X[t]$ have passed, and the Energy reaches 0 for the first time, we are essentially in a steady-state situation. Using the simple policy of "always use Wait," we naturally see a repetitive behavior (see Table 1).

t = …	k	k+1	k+2	k+3	k+4	k+5
x_1[t]	20	20	20	20	20	20

Table 1. CEC with Wait only.

Here, we see the cost at each time step will be:

$$\frac{1}{T} \cdot 3^{\frac{10(80-100)}{100}} = \frac{1}{T} \cdot 3^{-2} \approx \frac{0.111}{T}$$

What would happen if we changed our policy to "use Mitigate whenever possible; otherwise use Wait," which we will call the "Greedy Mitigate" policy? The spikes thus also repeat (see Table 2):

[20 12 12 20], sum 64

[12 12 20 12], sum 56

[12 20 12 12], sum 56

[20 12 12 20], sum 64

etc.

t = …	k	k+1	k+2	k+3	k+4	k+5
E[t]	3	1	2	3	1	2
u[t]	1 (Mtgt)	0 (Wait)	0 (Wait)	1 (Mtgt)	0 (Wait)	0 (Wait)
P[t]	0	1	0	0	1	0
Mitigated?	N	Y	Y	N	Y	Y
x_1[t]	20	12	12	20	12	12

Table 2. CEC with Greedy Mitigate.

We see then that the mean steady-state cost per time step is:

$$\frac{2}{3} \cdot \frac{1}{T} \cdot 3^{\frac{10(56-100)}{100}} + \frac{1}{3} \cdot \frac{1}{T} \cdot 3^{\frac{10(64-100)}{100}} \approx \frac{\left(\frac{2}{3} \cdot 0.00796 + \frac{1}{3} \cdot 0.0192\right)}{T} \approx \frac{0.0117}{T}$$

There are two things to learn from this. First, it's clear that using control is much less costly than not using it at all. Second, even slightly higher spikes are much more costly.

We could try concentrating our mitigated time steps closer together, but that isn't helpful. Let us demonstrate. Take this "Alternate Mitigation" control policy: "Use Mitigate whenever Energy is at 4, then again when it next reaches 3. Repeat." This policy results in the pattern 12x4, 20x2, 12x4, etc. (see Table 3). We see that we have now one spike which is only 12x4 for a total of 48. However, the 20s are now much closer together. Windows summing to 64 now account for three of six window types, which is half of the types, compared to one third of the types with the Greedy Mitigate policy. Indeed, we will see that the cost for the Alternate Mitigation policy is higher:

t = …	k	k+1	k+2	k+3	k+4	k+5	k+6	k+7
E[t]	4	2	3	1	2	3	4	2
u[t]	1 (Mtgt)	0 (Wait)	1 (Mtgt)	0 (Wait)	0 (Wait)	0 (Wait)	1 (Mtgt)	0 (Wait)
P[t]	0	1	0	1	0	0	0	1
Mitigated?	N	Y	Y	Y	Y	N	N	Y
$x_1[t]$	20	12	12	12	12	20	20	12

Table 3. CEC with Alternate Mitigation.

$$\frac{1}{6}\cdot\frac{1}{T}\cdot 3^{\frac{10(48-100)}{100}}+\frac{2}{6}\cdot\frac{1}{T}\cdot 3^{\frac{10(56-100)}{100}}+\frac{3}{6}\cdot\frac{1}{T}\cdot 3^{\frac{10(64-100)}{100}}\approx\frac{0.0128}{T}$$

This is still significantly better than using no control at all, naturally, but concentrating the control into one location was certainly not helpful, because it created large spikes at the same time as small ones. This hints that we should spread our control effects out as much as possible, perhaps unless we generate more small spikes than large spikes.

Let us also consider the "Greedy Recover" control policy, which uses "Recover" every time step (it is never unavailable except possibly at t=0). Under this policy, every time window sums to 60 (see Table 4). We easily have the steady-state cost per time step:

$$\frac{1}{T}\cdot 3^{\frac{10(60-100)}{100}}\approx\frac{0.0123}{T}$$

t = …	k	k+1	k+2	k+3	k+4	k+5
E[t]	1	1	1	1	1	1
u[t]	2 (Rcvr)	2 (Rcvr)	2 (Rcvr)	2 (Rcvr)	2 (Rcvr)	2 (Rcvr)
x_1[t]	15	15	15	15	15	15

Table 4. CEC with Greedy Recover.

It makes intuitive sense that a 60x3 pattern would be slightly more costly than a 56x2, 64x1 pattern, since we increase two spikes and lower one.

To save time, allow me to claim without proof that spreading the uses of

Recover by waiting for 2 Energy is the same cost, and waiting for 3 Energy results in the pattern 65x2, 50x1, which, perhaps surprisingly, is significantly more costly than Greedy Recover or even Alternate Mitigation. Thus, it would again seem that spreading out control effects as much as possible is optimal.

In fact, as it makes no intuitive sense to mix the strategies in the deterministic case, it seems that Greedy Mitigate is the optimal control policy for CEC (with the given typical parameters). Although we have not really proved it, the data is strongly suggestive. It seems unintuitive that a "greedy"-style control policy would be optimal, though. Surely, this cannot possibly be optimal in the stochastic case, as it makes little intuitive sense to use Mitigate after several Avoids in a row just because we have enough Energy to use it (and we will see later that the intuition is correct). Furthermore, note that the conservation of Energy has no purpose in the CEC case, which is a hint that we will not get a particularly useful policy from CEC. A steady state beginning at 4 Energy using Mitigate every 3 time steps (as in Greedy Mitigate) has the exact same spike pattern as Greedy Mitigate, and using that extra one Energy to use Mitigate one time step early has the same effect on total cost no matter when it is used. Note that we also showed saving Energy up to 4 to use for later was detrimental overall, since conserving Energy is detrimental in the short term.

In conclusion, the Greedy Mitigate and Greedy Recover control policies should be decent, according to CEC, but we can almost definitely derive stronger suboptimal control policies by using Energy conversation techniques and mixing Recover and Mitigate strategies. As such, we look to other suboptimal control

methods.

4.2 Rollout Algorithm with Monte-Carlo Simulation

The Rollout Algorithm is a type of Limited Lookahead suboptimal control method. The idea of Limited Lookahead is to fully evaluate the possible system trajectories and their probabilities, from the current state with all available controls, for the next few time steps (usually just for one time step, though, called One-Step Lookahead). Then, the cost-to-go is estimated for the endpoint of each of those trajectories, which yields an estimated total cost for each trajectory. The costs are compared, and the lowest one is selected. Its first control used is the Lookahead estimate for the optimal control. Essentially, one computes the first few steps of dynamic programming, then estimates the remaining cost. Usually, it is computed online using the current state, as most control problems have a large number of possible control values. However, Example 2 does not, so it may be possible to evaluate offline for all possible states for a given time t.

The Rollout Algorithm specifically uses a "Base Policy" to estimate the cost-to-go. That is, the estimated cost-to-go is the cost-to-go of the state's evolution under the Base Policy. The Rollout Algorithm with Monte-Carlo simulation is a method where the cost-to-go under the Base Policy is further estimated by running many simulations of the state evolution under the base policy, then taking the mean over all simulations. Obviously, this method will have some probabilistic variance. One

way the variance can be reduced is by using the same stochastic results for all parallel trajectories[1]; that is, using the same $L[\cdot]$ for all controls for each run of the simulation. This ensures that, in one of the many runs of the simulation, if the first value of $L[\cdot]$ when we used Wait is Avoid, then it will be Avoid for Mitigate and Recover as well.

To express all this mathematically, the number we compare for each control is called the *Q-factor*, defined in general at time k by[1]:

$$Q_k(x_k, u_k) \stackrel{\text{def}}{=} \mathbb{E}\{g_k(x_k, u_k, w_k) + H_{k+1}(f_k(x_k, u_k, w_k))\}$$

where g_k is the cost incurred at time k, f_k is the next state given the current state etc., and H_{k+1} is an estimate of the cost-to-go from a given state. The cost does not depend explicitly on u or w ($L[k]$) for us. Thus, to go back to our notation, we need only compare $\mathbb{E}\{H_{k+1}(f_k(X[k], u[k], L[k]))\}$ between controls. Since we know the variance will decrease by taking the difference of these using the same $L[\cdot]$ results, we will compare

$$H_{k+1}\left(f_k\left(X[k], u_1[k], \vec{\boldsymbol{L}}[\cdot]\right)\right) - H_{k+1}\left(f_k\left(X[k], u_2[k], \vec{\boldsymbol{L}}[\cdot]\right)\right)$$

to zero, where $\vec{\boldsymbol{L}}[\cdot]$ is the shared vector of all values of L.

In fact, we can even do a bit better, if we like. In this problem, the stochastic result that immediately follows the current time seems to be very important. Thus, we include this precision in the estimations of the cost-to-go. That is, we are not just changing the control $u[k]$, we are also exploring each of the three stochastic events that can happen at time $k+1$, which is determined by $L[k]$. That means we are

running nine parallel trajectories rather than three, but if randomness in $L[k]$ affects the total cost by a large enough amount, it may save time by requiring fewer simulation runs to get the same level of confidence. This is not standard in Rollout suboptimal control, but there is no reason one could not do it. To be clear, it is neither One-Step nor Two-Step Lookahead.

The question remains of how to obtain a confidence in our answer. We could run the simulation (with all nine parallel trajectories) only once and get an answer, but it would likely not be very accurate. We need to know how many times to run the simulation to get an accurate result. Since we do not know what the actual distribution should be (as one would do when testing if a coin or die is "fair," for example), we use the theory of confidence intervals to get an answer. We want to know if the mean

$$\mu = \mathbb{E}\big[H_{k+1}\big(f_k(X[k], u_1[k], L[k])\big) - H_{k+1}\big(f_k(X[k], u_2[k], L[k])\big)\big]$$

is positive (u_2 is better), negative (u_1 is better), or zero (both are just as good). The method to discover this empirically can be found in many elementary statistics books. First, we determine the empirical mean. Assuming we run the simulation N times, this is given by

$$\hat{\mu} = \frac{1}{N}\sum_{j=1}^{N} H_{k+1}\left(f_k\big(X[k], u_1[k], \vec{\boldsymbol{L}}_j[\cdot]\big)\right) - H_{k+1}\left(f_k\big(X[k], u_2[k], \vec{\boldsymbol{L}}_j[\cdot]\big)\right)$$

where $\vec{\boldsymbol{L}}_j[\cdot]$ is the vector of all the stochastic results from the j^{th} Monte-Carlo simulation, and the "hat" represents an empirical estimate. Recall again that we use

the same stochastic results for all trajectories, so this vector is used for both u_1 and u_2. Similarly, we can compute the standard deviation $\hat{\sigma}$ (or alternatively, variance $\hat{\sigma}^2$) of the data set. The standard deviation of the mean of the data set is then given by the simple formula:

$$\hat{\sigma}_{\hat{\mu}} = \frac{\hat{\sigma}}{\sqrt{N}}$$

From this, we can establish the size of a confidence interval that $\mu < 0$ if $\hat{\mu} < 0$, or $\mu > 0$ if $\hat{\mu} > 0$, using the error function *erf*, which is, roughly speaking, the integration of the Gaussian Probability Density Function. With some manipulation of the error function, and accounting for the fact that our confidence interval is infinite on one end, the confidence works out to be the following:

$$c = \frac{1}{2}\left(1 + \text{erf}\left(\frac{|\hat{\mu}|\sqrt{N}}{\hat{\sigma}\sqrt{2}}\right)\right) = \frac{1}{2}\left(1 + \text{erf}\left|\frac{\hat{\mu}\sqrt{N}}{\hat{\sigma}\sqrt{2}}\right|\right)$$

With the Rollout Algorithm, we must also decide what to use as our Base Policy. The CEC trials we did help there: we should likely select Greedy Mitigate or Greedy Recover. Let us choose Greedy Mitigate for the time being, since Greedy Mitigate was triumphant in the CEC case, and it only gets stronger during the largest spikes, since it reduces Health losses by larger amounts, whereas Recover's effect is static and independent of $L[t]$.

Building the simulator in MATLAB, we can get the results from given initial conditions at $t = 0$ with 500 simulations in less than one second, with confidence usually above 0.95 and often greater than 0.9999. Note that if we wanted to start at

t=100, it's simply a matter of changing T from 400 to 300, and setting "X0" to X[100].

In Section 3 it was mentioned that the parameters must be tuned relatively tightly to exhibit interesting optimal control policy behavior. Analyzing the simulation results at key points, we can see the results mostly match the hypothesis. However, it was necessary to increase the power of Recover from 5 to 6 (per Energy) to give it more of an impact. It was only showing up as optimal under Rollout in circumstances that would be impossible under Greedy Mitigate steady-state, such as initial conditions of $X[0] = [25 \quad 25 \quad 15 \quad *]'$ with high Energy, for example (recall x_1, the most recent loss, is the leftmost value). It is intuitively clear that x_4 will never matter for deciding control, since once a loss has moved to x_4, its effect on the total cost has been finalized, hence the asterisk. With this change to RcvrAmt, it also shows up in the largest spikes under steady-state, such as $X[0] = [25 \quad 15 \quad 15 \quad *]'$. Thus, with the new set of parameters, we can demonstrate the interesting features of Example 2. Note that this means our rationale for our choice of Base Policy is somewhat invalidated, so we will also try Greedy Recover as the Base Policy later.

As expected, even with the RcvrAmt change, outside of the largest spikes, Mitigate is usually optimal under Rollout at high Energy. Further, the Rollout control policy is more liberal with spending Energy, as opposed to Waiting, as Energy increases and recent damage increases. Naturally, more recent losses were also more important than later ones.

Surprisingly, if the initial conditions had low values for $X[0]$, the optimal Rollout control was sometimes Wait even with maximum Energy, especially if $P[0]$

was not 0. The hypothesis was that Wait would vanish or almost vanish at maximum Energy, which evidently was incorrect. The threshold on how heavy the Health losses had to be before the Rollout optimal control changed to Mitigate was surprisingly high, though still reasonable enough to be believable. For example, the optimal Rollout control is Wait with $E[0] = 5,\ P[0] = 0,\ X[0] = [15 \quad 5 \quad 5 \quad *]$ but Mitigate if the 15 changes to a 20 or 25. By extension, it is not valid to assume for any RBSM Stochastic Control Problem that Energy-conserving control values are suboptimal at maximum Energy.

Seeking to get a more complete picture of the optimal Rollout control, this process was looped over all potential initial conditions, though limiting $P[t]$ to 0 and 1. This turned out to be too computationally intensive. Consider that taking 20 steps between -21 and 25 for x_1 through x_3, that multiplies the runtime by $2 \cdot 5 \cdot 20^3 = 80000$, which means if the simulation for one set of initial conditions ran for one second, running all of them would take a little over 22 hours. This at least also assures us that dynamic programming optimal control was certainly out of the question.

To cut down on runtime, some adjustments were made: First, the span of x_1 through x_3 was reduced to -15 through 25, jumping by 5 each time, which reduces the number of data points along those axes from 20 to 9. The number of simulations per set of initial conditions was also reduced from 500 to 250 (which lowers the confidence) and reduced T from 400 to 100. The reduction of T may not make sense at first, but recall that our Base Policy is Greedy Mitigate, so the effects of saving or

spending Energy in the first time step of the simulation really effect the cost of the next few time steps. In fact, we would likely see few control changes reducing it even further to 50. Even under other Base Policies, the effects of spending or saving Energy would at most last for the next few small and large spikes, which would occur with very high probability within 50-100 time steps.

With these changes, the simulation in entirety took about 20-30 minutes. In the script, each result (the Rollout control and its confidence) was saved into a 6-D matrix, which was saved as a MATLAB ".MAT" file and will be included in the Appendix along with the scripts. There were no additional surprises in the results. It looked largely the same as the few "key point" initial condition simulations that were ran individually. Of course, the new data was more comprehensive.

It also became clearer that the optimal control is a very complicated function of all five relevant initial conditions. Some very general data patterns were already stated in this subsection, but it was difficult to ascertain any more precise patterns by hand. To find precise patterns or thresholds over the whole data set, one would need to run a multi-dimensional curve fit to a discrete-valued function, which would likely take quite some time to analyze. Regardless, such a matrix can be stored as a lookup table to be consulted during online operation of Example 2, and it is calculable in a somewhat reasonable amount of time, given one makes some adjustments and approximations, as we saw in this subsection.

Since it is impractical to compute the optimal control for this problem, we cannot know what the cost would be under the optimal control. However, we can at

least compare the empirical mean of various policies. Rollout algorithms are actually proven to be no worse than the Base Policy[1]. We would like to see, though, if Rollout control is a nontrivial improvement over Greedy control policies. Running a simulation that computes the Rollout optimal control at every time step took a few minutes, but we can still obtain a reasonably small 95% confidence interval for the mean cost of Rollout control by running the simulation a few times. Rollout control is compared to other key control policies in Table 5, below. Note that even after changing RcvrAmt from 5 to 6, Greedy Recover is about on par with Greedy Mitigate. (The mean cost with RcvrAmt=5 was actually around 9.0.) The Rollout policies under different Base Policies are also too close together to determine which would be better. If more computation time were devoted to the problem, it could be determined. Regardless, we can be fairly confident that either of the Rollout policies is a nontrivial improvement over Greedy policies.

Policy:	Rollout (Base: Gr. Mitigate)	Rollout (Base: Gr. Recover)	Greedy Mitigate	Greedy Recover	Wait only
Mean cost **×100** (c=0.95)	1.77±0.28	1.66±0.46	2.25±0.20	2.30±0.19	31.0±1.9

Table 5. Comparison of empirical mean costs of important control policies.

If we accept that the Rollout Algorithm gives a significant improvement over Greedy policies, the question obviously becomes how to apply this knowledge to other problems. First, note that all RBSM Stochastic Control Problems share the

same features of Example 2 that created the Energy conservation behavior found in the optimal Rollout control. Second, although we have done a lot of analysis, keep in mind that applying Rollout is as simple as simulating a few parallel trajectories over future time steps under a base policy. The scripts used to simulate Example 2 (found in the Appendix) are written to be modular enough to be adapted for other problems easily enough. Alternatively, one could slightly alter existing simulators of the system in question. The data from Example 2 and intuition gained from this section suggests that Rollout suboptimal control methods provide a nontrivial improvement over CEC for RBSM Stochastic Control Problems. Further, our analysis of Rollout control suggests that policies that conserve Energy during smaller spikes to use during larger spikes are nontrivially closer to optimal than policies that do not, as our hypothesis predicted.

5 Potential Research Extensions

The following items would extend the ideas of this thesis:

- Multi-dimensional curve fit to Rollout control data grid and thus computation of a function yielding the optimal Rollout control rather than a lookup table
    - Repeating the Rollout experiment using this function as the new Base Policy, and if possible, repeating until the process converges on a local minimum of cost (still not necessarily truly optimal)
- Runtime duration optimization, possibly including re-writing in another language and/or CPU core parallelization
- Multistep Lookahead Rollout control computations, without requiring prohibitively long runtimes (number of parallel trajectories increases exponentially)
- Analysis of Rollout Control or perhaps even Dynamic Programming of a RBSM Stochastic Control Problem with a low number of remaining or total time steps
- Analysis of a RBSM Stochastic Control Problem in which the controls synergize, i.e. using one control increases the potency of one or more other controls
- Analysis of a RBSM Stochastic Control Problem using suboptimal control methods other than the ones presented in this thesis

- Analysis of a problem similar to the RBSM Stochastic Control Problem model that instead has multiple spike states ($X_a[t]$, $X_b[t]$, etc.)
- Analysis of problems similar to the RBSM Stochastic Control Problem model that instead have other cost functions such that they exhibit the same Energy conservation properties as problems following the model in this thesis
- Analysis of a problem similar to the RBSM Stochastic Control Problem model that instead operates in continuous time or otherwise does not fit the discrete time model

6 Conclusion

Resource-Based Spike Mitigation Stochastic Control Problems are interesting in that the optimal control policy develops a behavior that exploits the stochastic elements to reduce total cost. That is, policies save resources during smaller spikes and spend more resources during larger spikes. This behavior manifests only with all three properties present: controls tied to a resource, cost increasing exponentially with larger spikes, and stochastic events.

Without a resource cost/dependence on controls, no conservation behavior can be noticed. To continue to use Example 2, Mitigate's effect is always active if used every other time step, so assuming Recover behaves as if one always consumed CostMaxRecover Energy units, the optimal control either alternates Mitigate and Recover, or if Recover is strong enough, simply uses Recover every time step.

With simple linear costs on Health losses, the time at which damage reduction or negation occurs becomes meaningless as far as the total cost is concerned, so there is no longer any point to save Energy to use during larger spikes. Instead, the optimal policy revolves around lowering the total loss to Health. Example 2 would just devolve into Greedy Mitigate or Greedy Recover policies as optimal, depending on which had the most negative mean effect on $X[t]$. These policies would also be tied for optimality with policies that save Energy, then spend it later arbitrarily, as long as they never cap on Energy.

As we saw in the Certainty Equivalence Control subsection, if the RBSM

problem is deterministic, all spikes are the same size, so there is no point to conversation of resources, except possibly during the effect of initial condition transients. In fact, this also means that conserving resources usually hurts more in the next few time steps than one could gain later, which means spending resources constantly is strongly optimal. It can also mean that certain controls completely dominate other ones, and thus some controls can go completely unused. Thus, the optimal control often just becomes a simple pattern in deterministic RBSM problems, potentially even allowing the problem to be solved by hand.

These problems are often too complex to solve by dynamic programming. However, we can resort to decent suboptimal control methods. It is reasonably feasible to create a lookup table for a Rollout control policy of a RBSM Stochastic Control Problem. The lookup table, though, may be somewhat sparse, due to computation time limits. If more precise results are desired, it does not take long to calculate a Rollout Control "online" during the control of a RBSM Stochastic Control Problem, using the current state. This may or may not be acceptable depending on the real-time duration of the discrete time steps. Therefore, both options have the potential to be useful in approximating the solution to a RBSM Stochastic Control Problem. However, in problems where the quantity of possible control values is infinite, such as Example 3, one must either discretize the control range and/or solve the problem online (i.e. in real time) using only the current state, since solving the problem for every possible state would likely be computationally infeasible.

APPENDIX

A.1 Control with Health and Known Health Gain Distribution

Early in the development of this thesis, a problem similar to the RBSM Stochastic Control Problem was examined, where Health was an actual state and the Health gains from outside sources was, instead of being completely unknown and infinite on-demand, given as a known probability distribution. This was specifically about Example 2, so the reader should notice the similarities and pay careful attention to the differences. Let us first detail the mathematics of the problem.

Parameters:

(All of these are the "typical" values from Example 2)

T= 400

MaxH= 100

MaxE= 5

Avoid= 0.15

Glance= 0.2

FullHit= 25

GlanceHit= 18

RegenE= 1

ControlWait= 0

ControlMitigate= 1

ControlRecover= 2

CostMitigate= 3

CostMaxRecover= 3

MitigateAmount= 0.6

RecoverAmount= 5

Initial conditions:

$E[0] = 0, \qquad H[0] = 100.0, \qquad P[0] = 0$

State Evolution:

$t \in \mathbb{Z} \cap [0, \mathrm{T}]$

$E[t] \in [0, \mathrm{MaxE}], \qquad \forall t$

$L[t], G[t] \in \mathbb{R}, \qquad \forall t$

$H[t] \in [0, \mathrm{MaxH}], \qquad \forall t$

$P[t] \in \mathbb{Z}_{+}, \qquad \forall \mathrm{t}$

$u[t] \in U(E[t], X[t], P[t]) = U(E[t])$

$$= \begin{cases} \{\mathrm{ControlWait}\}, & E[t] = 0 \\ \{\mathrm{ControlWait}, \mathrm{ControlRecover}\}, & 0 < E[t] < \mathrm{CostMitigate} \\ \{\mathrm{ControlWait}, \mathrm{ControlMitigate}, \mathrm{ControlRecover}\}, & E[t] \geq \mathrm{CostMitigate} \end{cases}, \qquad \forall t$$

(Intuitive abbreviations or numerical substitutions of parameters follow to keep

equation size down)

$G[t] \sim \text{Uniform}(5.0, 10.0)$

$$L[t] = \begin{cases} 0 \text{ w.p. Avoid} \\ \text{GlanceHit w.p. Glance} \\ \text{FullHit w.p. } 1 - \text{Avoid} - \text{Glance} \end{cases}$$

$$E[t+1] = \min(\text{MaxE}, \begin{cases} E[t] + \text{RegenE}, & u[t] = \text{CtrlWait} \\ \text{RegenE}, & (u[t] = \text{CtrlRcvr}) \text{ and } (E[t] < \text{CMaxRcvr}) \\ E[t] - \text{CMaxRcvr} + \text{RegenE}, & (u[t] = \text{CtrlRcvr}) \text{ and } (E[t] \geq \text{CMaxRcvr}) \\ E[t] - \text{CostMitigate} + \text{RegenE}, & u[t] = \text{CtrlMitigate} \end{cases})$$

$$P[t+1] = \begin{cases} 0, & P[t] = 0 \text{ and } u[t] \neq \text{ControlMitigate} \\ P[t] - 1, & P[t] > 0 \text{ and } u[t] \neq \text{ControlMitigate} \\ P[t] + 1, & u[t] = \text{ControlMitigate} \end{cases}$$

$$H[t+1]$$

$$= \begin{cases} \min(100, H[t] - L[t] + G[t]), & u[t] = 0 \text{ and } P[t] = 0 \\ \min(100, H[t] - 0.6 * L[t] + G[t]), & u[t] = 1 \text{ or } (u[t] = 0 \text{ and } P[t] > 0) \\ \min(100, H[t] - L[t] + G[t] + 5.0 * \min(3, E[t])), & u[t] = 2 \text{ and } P[t] = 0 \\ \min(100, H[t] - 0.6 * L[t] + G[t] + 5.0 * \min(3, E[t])), & u[t] = 2 \text{ and } P[t] > 0 \end{cases}$$

Problem Goal:

Prevent Health from ever reaching zero. If this is possible, what control policy results in the lowest-ever value of H[t] over the simulation, on average?

The answer to this problem turned out to be trivial. Any control policy must be able to survive a worst-case scenario. The highest possible loss is 25 every time step, the lowest possible gain is 5 every time step, Mitigate changes the 25 to a 15,

Recover changes the 25 to at the least a 10. Therefore, neither control option can prevent reaching zero Health in the worst case scenario. If it could, then even in the worst case scenario, Health would not budge from 100, or would return to 100 at least once every three time steps. Given feasible parameters, it is a simple exercise to find a policy that would do this. Essentially, it would be either Greedy Mitigate or Greedy Recover.

A nontrivial version of the problem would be one where the system has probability 1 to drive $H[t]$ to 0 as $t \to \infty$. The objective would instead be to maximize the end time of the system, like the opposite of a Stochastic Minimum End Time problem. It would be a lot like Tetris, in this case. This kind of system would be strange to have in a co-operative game, though, where combat lasts for 5-15 minutes. The result of combat would be more a factor of luck, rather than skill, and would likely be frustrating to players just attempting the battle until luck went in their favor, finally nabbing them the victory. It would be like something like slot machines, but as if the slot machine took ten minutes to give the result.

A.2 Single-Trajectory Simulation

This MATLAB code is for simulating one simulation run with given initial conditions and a Base Policy. The Base Policy is located deeper into the code (default Greedy Mitigate), but most variable options and parameters can be edited at the top of the code. This code can be downloaded at http://bit.ly/1ba9fmx.

A.3 Rollout Control Simulation, Given Initial Conditions

The MATLAB code for the Rollout Control is for simulating N simulation runs with given initial conditions and a Base Policy, then determining the optimal Rollout control and the confidence level at t=0. The Base Policy is located deeper into the code (default Greedy Mitigate), but most variable options and parameters can be edited at the top of the code. This code can be downloaded at http://bit.ly/1ba9fmx.

A.4 Rollout Control Simulation, Grid Initial Conditions

The MATLAB code for the Rollout Control simulation that runs using a large grid-like set of initial conditions is for simulating N simulation runs with given initial conditions and a Base Policy, then determining the optimal Rollout control and the confidence level at t=0 for every set of initial conditions in the grid. The Base Policy is located deeper into the code (default Greedy Mitigate), but most variable options and parameters can be edited at the top of the code. "Xspan" contains all x_i initial conditions to simulate for, but boundaries for Energy and P are changed deeper into the code, as well. Since the code takes a long time to run, the results of the code as-is below can be downloaded as a ".MAT" file along with the code itself at http://bit.ly/1ba9fmx.

A.5 Full Rollout Simulation

The MATLAB code for the full Rollout Control simulation, which uses Rollout Control as its control policy, runs through one instance of the state evolution of Example 2. It computes the best Rollout Control at every time step, simulating N runs with the current state as its initial conditions and a Base Policy. The Base Policy is located deeper into the code (default Greedy Mitigate), but most variable options and parameters can be edited at the top of the code. This code can be downloaded at http://bit.ly/1ba9fmx.

SCHOLARLY REFERENCES

[1] Bertsekas, Dimitri P. *Dynamic Programming and Optimal Control.* Nashua: Athena Scientific, 2005.