12-13-2014

Bounds for the Maximum-Time Stochastic Shortest Path Problem

Anara Bolotbekovna Kozhokanova

Bounds for the maximum-time stochastic shortest path problem

By

Kozhokanova Anara Bolotbekovna

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2014

Bounds for the maximum-time stochastic shortest path problem

By

Kozhokanova Anara Bolotbekovna

Approved:

Eric Hansen
(Major Professor)

Ioana Banicescu
(Committee Member)

J. Edward Swan, II
(Committee Member)

Edward B. Allen
(Graduate Coordinator)

Jason M. Keith
Interim Dean
Bagley College of Engineering

Name: Kozhokanova Anara Bolotbekovna

Date of Degree: December 13, 2014

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Eric Hansen

Title of Study: Bounds for the maximum-time stochastic shortest path problem

Pages of Study: 30

Candidate for Degree of Master of Science

A stochastic shortest path problem is an undiscounted infinite-horizon Markov decision process with an absorbing and cost-free target state, where the objective is to reach the target state while optimizing total expected cost. In almost all cases, the objective in solving a stochastic shortest path problem is to minimize the total expected cost to reach the target state. But in probabilistic model checking, it is also useful to solve a problem where the objective is to maximize the expected cost to reach the target state.

This thesis considers the *maximum-time stochastic shortest path problem*, which is a special case of the maximum-cost stochastic shortest path problem where actions have unit cost. The contribution is an efficient approach to computing high-quality bounds on the optimal solution for this problem. The bounds are useful in themselves, but can also be used by other algorithms to accelerate search for an optimal solution.

Key words: stochastic shortest path problem, dynamic programming, probabilistic model checking

DEDICATION

To my Mother.

ACKNOWLEDGEMENTS

This thesis wouldn't have been possible without Dr. Eric Hansen and my committee members Dr. Ioana Banicescu and Dr. Edward Swan. I am very grateful to my mentors for their help and invaluable support. I would like to thank my supervisor Dr. Hansen for insightful guidance throughout the process of working on the thesis and his continuous support that encouraged the work.

Special thanks go to my friends and colleagues who made my stay in Starkville more enjoyable (in alphabetical order): Ajay, Arindam, Amin, Bahar, Bota and Viktor, Ibrahim, Jinchuan, Kazi, Maryam, Nadeem, Peng, Proteek, Shirin, Vika and Bryan, Zadia and Zaeem, and all other wonderful people I met during my Fulbright program.

Finally, I'm eternally grateful to my family for their love, support, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis addresses how to compute bounds on the optimal value function for the maximum-time stochastic shortest path problem. This chapter introduces and motivates research on this problem, and explains its relevance to probabilistic model checking.

1.1 Motivation

Model checking is an automated approach to formal verification that builds a model of a system, and analyzes the model to check whether it meets the specification of a property, where the property is usually expressed in some form of temporal logic. Probabilistic model checking is an extension of traditional model-checking techniques for analyzing systems with probabilistic behavior, such as randomized algorithms, computer networks, and communication and security protocols.

Several different probabilistic state-transition models are used in probabilistic model checking. This thesis considers Markov decision processes (MDPs), which are widely-used to model systems that exhibit both probabilistic and nondeterministic behavior. Probabilities are used to model uncertain events or the randomized steps of an algorithm. Nondeterminism models concurrency, as when multiple processes execute in parallel, or else unknown or underspecified behavior.

A probabilistic model checker is used to verify whether a stochastic system satisfies a property with some minimum probability. For systems that are modeled as MDPs, the presence of nondeterminism in the model means that it is usually not possible to compute exact probabilities. Instead, the values computed are the minimum or maximum probabilities that the property is satisfied over all possible resolutions of nondeterminism – a best-case and worst-case analysis. When real-valued costs (or rewards) are associated with an MDP, a probabilistic model checker can also reason about the minimum and maximum expected value of other measures of performance, such as the "expected number of lost messages" or the "expected time until a protocol terminates."

This thesis is concerned with questions of the last kind: the expected time until a protocol (or some other process) terminates. A probabilistic model checker computes both lower and upper bounds on the expected time by solving both a minimum-time Markov decision process and a maximum-time Markov decision process. Both the minimum-time and maximum-time problems correspond to special cases of a Markov decision process called a stochastic shortest path problem.

A stochastic shortest path problem is an undiscounted infinite-horizon Markov decision process with a target or goal state, where the objective is to minimize (or maximize) the expected cost (or time) to reach the target state. In the case of verifying a randomized protocol, for example, the target state corresponds to successful termination of the protocol, and the probabilistic model checker wants to bound the time required for the protocol to terminate.

Algorithms for solving minimum-time stochastic shortest path problems are widely-studied and well-understood. By contrast, the maximum-time stochastic shortest path problem has been studied very little because the objective of maximizing the time to reach a target or goal state has few applications. But it does have this important application in probabilistic model checking.

Hansen, Abdoulahi, and Shi [9] propose a branch-and-bound algorithm for solving Markov decision problems that can be used to solve both minimum-time and maximum-time stochastic shortest path problems. The algorithm requires initial lower and upper bounds that are improved each iteration until convergence to an optimal solution. Methods for generating bounds for minimum-time Markov decision processes are well-understood. But because there has been very little work on maximum-time stochastic shortest path problems, there is no previous work on generating initial bounds for this problem.

The contribution of this thesis is to introduce an approach to computing lower and upper bounds for the maximum-time stochastic shortest path problem. Trivial lower and upper bounds for this problem are 0 and ∞, respectively, but such trivial bounds are of little practical use. The approach developed in this thesis efficiently computes bounds that are not only non-trivial, they are very close to optimal. The bounds can be used for several purposes. Among them, they can be used to initialize a branch-and-bound algorithm for solving the problem.

1.2 Organization of the Thesis

The rest of the thesis is organized as follows. Relevant background information about probabilistic model checking and Markov decision processes is reviewed in Chapter 2. Chapter 3 develops new bounds for the maximum-time stochastic shortest path problem and describes how to compute them. Chapter 4 describes an implementation of these bounds in the probabilistic model checker PRISM, with experimental results. Chapter 5 summarizes the contributions and discusses planned future work.

CHAPTER 2

BACKGROUND

This chapter reviews probabilistic model checking, stochastic shortest path problems, and algorithms for solving stochastic shortest path problems.

2.1 Probabilistic Model Checking

Computer scientists are interested in automatic verification of software and hardware properties in order to both qualitatively and quantitatively prove compliance of systems with requirements [8]. Probabilistic model checking is widely applied in formal verification. For example, it is used to verify communication protocols [6], testing of multi-billion dollars projects such as space-craft control systems, and many other interesting applications in AI, biology, game theory, and others. (For a summary of applications, see [10]).

The model checking process requires two inputs: a model of a system and a property (usually expressed in some temporal logic) to check against that model (Figure 2.1). The resulting output is either true (if the property holds), false (if the property does not hold), or some numerical value. Properties of a system that involve a numerical value instead of a true/false answer include, for example: what is the minimum expected energy consumption of a disk drive's life-cycle, or what is the maximum expected number of attacks before the virus infects the network?

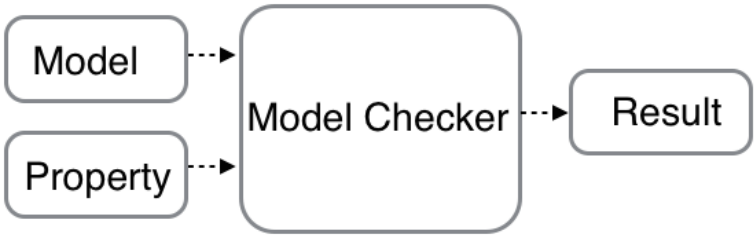

Figure 2.1

Model checking process

Different probabilistic models have been used in probabilistic model checking, including Markov chains, stochastic games, and different forms of probabilistic automata. This thesis applies to probabilistic model checking of systems that are modeled as Markov decision processes. Markov decision processes are used to model systems that have both probabilistic and nondeterministic behavior.

2.2 Stochastic Shortest Path Problem

A discrete-time Markov decision process (MDP) models a sequential decision process where action choices have probabilistic outcomes. In most applications of MDPs, the action choices are made by an intelligent controller. But when MDPs are used in probabilistic model checking, action choices are considered nondeterministic because they reflect the behavior of an outside *scheduler* of distributed processes, or some other uncontrolled behavior that is called an *adversary* or *policy*.

In this thesis, we consider a particular kind of MDP called a stochastic shortest path problem. A stochastic shortest path (SSP) problem is an optimization problem where the objective is to reach a target state with probability 1 while optimizing the total expected

cost [3, 7, 12]. More formally, an SSP problem is a tuple $< S, A, p, \mathcal{T}, g, H >$, where:

S : is a finite set of states;

A : is a function that maps each state $s \in S$ to a finite set of feasible actions $A(s)$;

$p : S \times A \times S \rightarrow [0, 1]$ is a probabilistic transition function that gives the probability $p(s, a, s')$ that taking action $a \in A(s)$ in state $s \in S$ results in a transition to state $s' \in S$;

$\mathcal{T} \in S$: is a finite set of target (or terminal) states that are absorbing and cost-free

$g : S \times A \rightarrow \Re$ is an immediate cost function that gives the expected cost $g(s, a)$ for taking action $a \in A(s)$ in state $s \in S$;

H : Horizon of the problem. For our SSP model the horizon is infinite.

As per L. de Alfaro [5], to compute the minimum or maximum reachability time in an SSP problem "it suffices to equate the cost to the time." Thus we consider the special case of a SSP problem where all actions have unit cost, that is, $g(s, a) = 1$, for all $a \in A(s)$ and $s \in S$. For the minimum-time SSP problem, the objective is to reach the target state while minimizing expected number of stages to reach the target state. For the maximum-time SSP problem, the objective is to reach the target state while maximizing the expected number of stages to reach the target state.

A policy μ is a mapping from states to actions. A policy is said to be *proper* if it ensures that the target state is reached with probability 1 from any initial state. Otherwise, it is said to be *improper*. Bertsekas and Tsitsiklis [3] prove that an optimal policy is proper under the following two assumptions. First, there is a proper policy μ. Second, any improper policy μ must have a state for which the expected total cost is infinite.

Another assumption that ensures that an optimal policy is proper is the assumption that all policies are proper, which is the assumption we consider in this thesis.

2.3 Value Iteration Algorithm

An optimal policy μ^* for the maximum-cost SSP problem can be found by computing the optimal cost vector J^*, which satisfies the following Bellman optimality equation:

$$J^*(s) = \begin{cases} 0 & \text{if } s = \tau, \\ \max_{a \in A(s)} \left[g(s,a) + \sum_{s' \in S} p(s,a,s') J^*(s') \right] & \text{otherwise.} \end{cases} \tag{2.1}$$

Given the optimal cost vector, an optimal policy is determined as follows:

$$\mu^*(s) = arg \max_{a \in A(s)} \left[g(s,a) + \sum_{s' \in S} p(s,a,s') J^*(s') \right]. \tag{2.2}$$

The optimal cost vector for a maximum-cost SSP problem can be found using the *value iteration* (VI) algorithm. VI is an iterative dynamic programming algorithm that is based on the Bellman equation due to R. Bellman (1957). Given an initial cost vector J_0, where $J_0(s)$ is set to some value, usually 0, for all $s \in S$, the cost vector is iteratively improved based on the Bellman equation, as follows:

$$J_k(s) = \max_{a \in A(s)} \left[g(s,a) + \sum_{s' \in S} p_{s,s'}(a) J_{k-1}(s') \right], s \in S. \tag{2.3}$$

As k goes to ∞, the cost vector J_k is guaranteed to converge to the optimal cost vector J^*. But in practice, the VI algorithm can only run a finite number of iterations. Thus it is considered to converge when successive values of J are the same for every $s \in S$, or sufficiently close. An approximate solution has been found when $\max_{s \in S} |J_{k+1}(s) - J_k(s)| < \epsilon$ is satisfied, for some user-specified $\epsilon > 0$.

The value iteration update of Equation 2.3 relies on a *dynamic programming operator*. The dynamic programming operator $T : \Re^n \rightarrow \Re^n$ is defined as

$$(TJ)(s) = \max_{a \in A(s)} \left[g(s,a) + \sum_{s' \in S} p(s,a,s')J(s') \right], \forall s \in S, \tag{2.4}$$

where J is a vector in the n-dimensional normed vector space $\Re^n$. Although the dynamic programming operator for the stochastic shortest path is not a contraction operator in general, Bertsekas proves [2] that it is an m-stage contraction operator with respect to the max norm when all policies are proper. The importance of the dynamic programming operator being a contraction operator is that it ensures convergence and allows the derivation of bounds, which we discuss in Chapter 3.

2.4 Branch and Bound Value Iteration Algorithm

A drawback of the VI algorithm is that it updates the value function for the entire state space each iteration. A novel approach proposed by Hansen et al. [9] combines branch-and-bound search with the VI algorithm (BnBVI). This approach allows an SSP problem to be solved more efficiently by pruning sub-optimal actions, as well as states that are unreachable from the start state under an optimal policy. Although we do not describe the branch-and-bound algorithm in this thesis, it provides the motivation for developing the bounds described in the next chapter. The original BnBVI strategy was applied to the minimum-cost SSP problem, for which bounds are already available. In this thesis, we develop bounds for the maximum-time SSP problem that can be used by the BnBVI algorithm to solve the maximum-time SSP problem more efficiently.

CHAPTER 3

NEW BOUNDS

For the maximum-cost SSP problem, we are not aware of any methods for computing bounds. It is not surprising since almost all work to date focuses on minimum-cost problems.

In this chapter, we describe a method for computing upper and lower bounds for the maximum-time SSP problem, which is a special case of the maximum-cost problem in which all actions have unit cost.

3.1 Preprocessing step

Upper bounds for the maximum-time SSP problem are guaranteed to have finite expectation if and only if all policies for the SSP problem are proper. If there's at least one improper policy (see Section 2.2 for a definition of proper and improper policy), then the maximum time until the target state is reached from some state(s) is unbounded [5].

Given an SPP problem for which not all policies are proper, it is possible to identify a subproblem for which all policies are proper. The subset of states for which all policies are proper, denoted S^1_{min}, is computed in a preprocessing step by the MinProb1 algorithm [5]. A description of the MinProb1 algorithm with examples can be found in V. Forejt et. al's paper [8]. The complexity of the MinProb1 algorithm is linear in the size of the MDP.

3.2 Example

Figure 3.1 shows a very simple example of an MDP that we will use to explain the derivation of bounds. Each node represents a state, where t denotes the target state. For each action, the dashed arcs show the successor state with associated transition probabilities. For states 2 and 3, there is only one available action. For state 1, there is a choice of two actions, one labeled A and the other labeled B. Thus, the problem has two stationary policies and both are proper.

It is well-known that an SSP problem must have an optimal policy that is stationary. For this example, the stationary policy that maximizes the expected number of steps to reach the target is the policy that chooses action A in state 1. (There is no choice of action in the other states.) Under this policy, the expected number of steps to reach the target state from state 1 is

$$\sum_{t=0}^{\infty}(0.99)^t = \frac{1}{(1-0.99)} = 100,$$

the expected number of steps to reach the target state from state 3 is

$$\sum_{t=0}^{\infty}(0.5)^t = \frac{1}{(1-0.5)} = 2,$$

and the expected number of steps to reach the target state from state 2 is

$$1 + 0.5 \times \frac{1}{(1-0.99)} + 0.5 \times \frac{1}{(1-0.5)} = 52.$$

For this example, it is easy to compute these values by hand. For more complex examples, it requires solving a complex optimization problem for an MDP, and initial upper and lower bounds can be very useful.

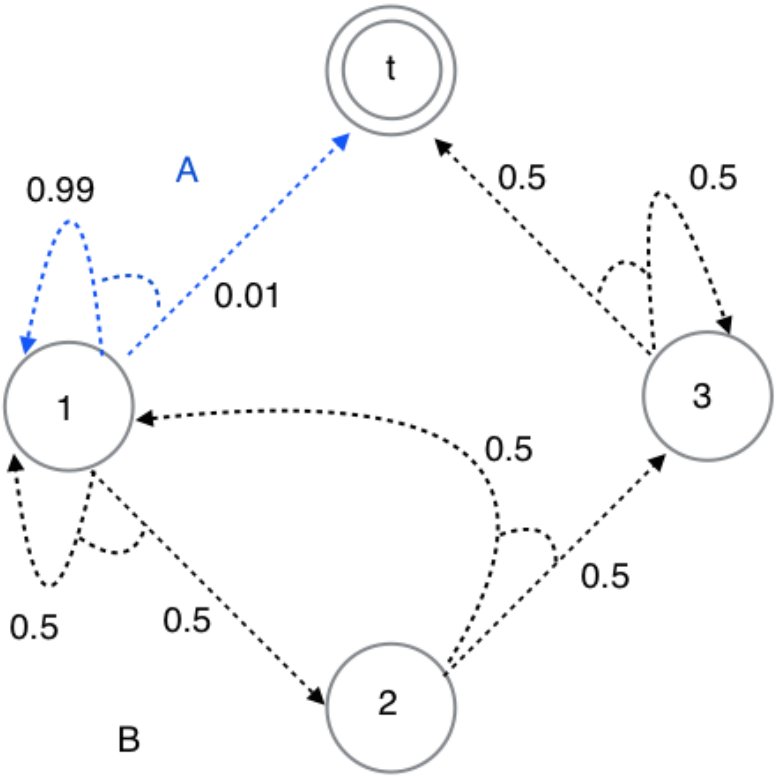

Figure 3.1

Example of a maximum-time SSP problem

3.3 Derivation of bounds

In this section, we explain how to compute bounds for the maximum-time SSP problem. We begin by introducing some notation. Let $N^*(s)$ denote expected number of steps to reach a target state from any state $s \in S$ following the policy μ^* that maximizes the number of steps to reach the target. Note that $N^*(s)$ satisfies the following dynamic programming recursion

$$N^*(s) = \max_{a \in A(s)} \left[1 + \sum_{s' \in S} p_{s,s'}(a) N^*(s') \right], \qquad \text{for all } s \in S. \tag{3.1}$$

Note also, that $N^*(s)$ is finite for all $s \in S$ if and only if all all states in S belong to the set S^1_{min}, which means that all policies are proper. From now on, we assume that S is the same as the set S^1_{min}.

We introduce the notation $\underline{N}(s)$ for a lower bound and $\overline{N}(s)$ for an upper bound on the maximum number of steps to reach the target under any policy. These bounds must satisfy the following inequality:

$$\underline{N}(s) \leq N^*(s) \leq \overline{N}(s), \text{ for all } s \in S. \tag{3.2}$$

The contribution of this thesis is to derive upper and lower bounds for the maximum-time SSP problem that are considerably better than the trivial lower bounds $\underline{N}(s) = 0$ and the trivial upper bounds $\overline{N}(s) = \infty$.

3.3.1 Upper bounds

Our first contribution is the following theorem, which gives an upper bound on the maximum number of steps to reach the target state under any policy.

Theorem 1[1]: *For any SSP problem for which all policies are proper, an upper bound on the expected number of steps to reach the target state from state $s \in S$ under any policy is given by*

$$\overline{N}(s) = m + (1 - p_m(s))\frac{m}{\rho_m}, \qquad \text{for all } s \in S, \tag{3.3}$$

where $p_m(s)$ is the minimum probability of reaching the target state from state $s \in S$ within m stages, and $\rho_m = \min_{s \in S} p_m(s)$ is the minimum non-zero probability of reaching the target state within m stages for all states $s \in S$.

Proof: Since ρ_m is a lower bound on the probability of reaching the target state in m steps beginning from any state, an upper bound on the expected number of steps to reach the target state beginning from any state is given by the infinite geometric sum

$$\sum_{t=0}^{\infty} m(1 - \rho_m)^t = \frac{m}{\rho_m}. \tag{3.4}$$

For any starting state s, the minimum probability of reaching the target state in m steps from s is $p_m(s)$. For the remaining probability, $1 - p_m(s)$, we can use the upper bound given by Equation 3.4 to get an upper bound on the expected number of steps to reach the target state from state s, as given by Equation 3.3. □

For these upper bounds to be finite, m must be large enough to ensure that $p_m(s) > 0$ for all $s \in S$, and thus $\rho_m > 0$. Because all policies are proper, there must be some finite m such that $\rho_m > 0$.

[1]The theorem and its proof is due to Eric Hansen

3.3.2 Lower bounds

Lower bounds for the maximum-time SSP problem are easier to obtain. Although zero is an obvious lower bound, tighter lower bounds can be obtained by evaluating any policy μ for the problem. The worse the policy μ used for the bounds, that is, the longer the policy delays reaching the target state, the tighter bounds we obtain. That suggests using the policy that minimizes the m-stage probability of reaching the target, which is the policy found in computing the upper bound.

Let $N_\mu(s)$ denote the expected number of steps to reach a target state from any state $s \in S$ by following a proper policy μ. Clearly, for any policy μ, we have

$$N_\mu(s) \leq N^*(s), \text{ for all } s \in S, \tag{3.5}$$

which means that $N_\mu(s)$ is a lower bound on the maximum number of steps to reach a target state from any state $s \in S$ for the maximum-time SSP problem.

For any policy μ, we can compute $N_\mu(s)$, for all $s \in S$, by solving the following system of n linear equations in n unknowns:

$$N_\mu(s) = 1 + \sum_{s'} p_{s,s'}(\mu(s)) N_\mu(s'), \qquad \textit{for all } s \in S. \tag{3.6}$$

For a lower bound on the number of steps to reach a target, we do not need to solve these equations exactly. We can start with a lower bound, such as 0 for all states $s \in S$, and update Equation 3.6 some fixed number of iterations. Each iteration produces an improved lower bound. This process is known as a policy evaluation (PE) algorithm because it finds the value function for a given policy.

From now on, we let $\underline{N}(s)$ denote a lower bound computed in this way.

3.4 Algorithms for computing parameters of bounds

The lower bounds, $\underline{N}(s)$, and upper bounds, $\overline{N}(s)$, described above can be computed by policy evaluation (PE) and value iteration (VI) algorithms respectively. We start by computing the upper bounds using the VI algorithm.

3.4.1 VI for the upper bound parameters

The VI algorithm finds the smallest probability $p_m(s)$ of reaching a target state $\tau \in \mathcal{T}$ from each state s in m stages, the smallest probability $\rho_m = \min_{s \in S} p_m(s)$ of reaching a target state from any state within m stages, the steps counter m, and a policy $\mu_m(s)$ for all $s \in S$ that minimizes these reachability probabilities at stage m. These are the parameters needed for our upper bound.

To find the probabilities $p_m(s)$ for some number m of stages, the VI algorithm solves the following dynamic programming recursion:

$$\begin{cases} p_k(\tau) = 1, & \text{for } \tau \in \mathcal{T}, k \geq 0, \\ p_0(s) = 0, & \text{for all } s \in S \backslash \mathcal{T}, \\ p_k(s) = \min_{a \in A(s)} \left[\sum_{s'} p_{s,s'}(a) p_{k-1}(s') \right], & \text{for all } s \in S \backslash \mathcal{T}. \end{cases} \tag{3.7}$$

The values for every state are initialized to 0 except the target states. The target states are initialized to 1. The VI algorithm is then performed.

To produce finite upper bounds, the VI algorithm must run at least until $p_k(s) > 0$ for all $s \in S$, at which point $m = k$. Our pseudocode for the algorithm uses this termination test. But running the VI algorithm longer will result in improved values of $p_k(s)$ and

improved upper bounds. The question of how long to run the VI algorithm is best answered experimentally, and we revisit it in Chapter 4 when we present experimental results.

Table 3.1

VI for the upper bound parameters

Input: $s \in S^1_{min}$, target set $\mathcal{T}$
Output: $p_m(s) > 0$ for every $s \in S$, ρ_m, m, μ

$m \leftarrow 0$
$p_0(\tau) \leftarrow 1$, for $\tau \in \mathcal{T}$
$p_0(s) \leftarrow 0$, for $s \in S$

While $m \geq 0$
    $m++$
    For each state $s \in S$ do
        $p_m(s) \leftarrow \min_{a \in A(s)} \left[\sum_{s'} p_{s,s'}(a) p_{m-1}(s') \right]$
        $\mu_m(s) \leftarrow \arg\min_{a \in A(s)} \left[\sum_{s'} p_{s,s'}(a) p_{m-1}(s') \right]$
    Test for termination: if $p_m(s) > 0$ for all $s \in S$ then
        Return $\rho_m \leftarrow \min_{s \in S} p_m(s)$, $p_m(s)$, for $s \in S$, m, μ_m

3.4.2 PE for the lower bound parameters

The m-stage policy μ_m we computed with the VI algorithm is used for computing the lower bound for the maximum-time SSP problem. The PE algorithm evaluates the m-stage policy μ_m for some number of iterations, which is given by the parameter iterNum in the following pseudocode.

Table 3.2

PE for the lower bound parameters

Input: $s \in S^1_{min}$, target set $\mathcal{T}$, m, a policy μ, iterNum
Output: lower bound value $\underline{N}(s)$, for $s \in S$

1: $m \leftarrow 0$
2: $\underline{N}_0(s) \leftarrow 0$, for all $s \in S$

3: While $m <$ iterNum
4: $\quad m++$
5: $\quad$ For each state $s \in S$ do
6: $\quad\quad \underline{N}_m(s) \leftarrow 1 + \sum_{s'} p_{s,s'}(\mu(s))\underline{N}_{m-1}(s')$

7: Return $\underline{N}(s) \leftarrow \underline{N}_m(s)$, for $s \in S$

3.5 Example revisited

Given the approach to computing upper and lower bounds we have developed, let's revisit the example introduced at the beginning of the chapter. For this example, it is not possible to avoid reaching a target state in 3 steps, and so we compute the minimum probability for each state to reach the target for $m = 3$, and get $p_3(1) = 0.01$, $p_3(2) = 0.375$, $p_3(3) = 0.875$ and $\rho_3 = 0.01$. These are the required parameters for computing the upper bound for the maximum-time SSP problem. After plugging the above parameters into Equation 3.3, the upper bound value for state 1 is 300, for state 2 is 190.5, and for state 3 is 40.5.

For the lower bound we evaluate a policy obtained from computing parameters for the upper bound as described in Section 3.3.2. Iterating over that policy 100 times, will get a

lower bound value for state 1 is 63.4, for state 2 is 33.5, and for state 3 is 2. These bounds are shown in the following table.

Table 3.3

Bounds for the optimal value

	state 1	state 2	state 3
Upper bound $\overline{N}$ (m = 3)	300	190.5	40.5
Optimal N^*	100	52	2
Lower bound $\underline{N}$ (100 iterations)	63.4	33.5	2

Note that the upper bounds could be improved by increasing the value of m and running the value iteration algorithm longer, and the lower bounds could also be improved in this way. Thus the approach allows a tradeoff between the time needed to compute the bounds and the quality of the bounds. We will evaluate this tradeoff in the next chapter.

CHAPTER 4

EXPERIMENTS

This chapter describes experiments in computing bounds for the maximum-time SSP problem.

4.1 Probabilistc Model Checker PRISM

We ran all experiments in the PRISM model checker [11]. PRISM supports four engines that use different data structures for solving problems. We tested examples in the explicit engine which uses explicit data structures. The other engines, sparse, MTBDD, and hybrid, use more sophisticated data structures that make it possible to solve larger problems faster.

4.2 Test examples

The PRISM model checker comes with the range of examples. For our experiments, we verified the maximum expected time [Rmax = ?] properties of the Consensus and Zeroconf models. Both Consensus and Zeroconf are models of communication protocols. The characteristics of the MDPs corresponding to these models are summarized in Table 4.1.

4.2.1 Consensus

Consensus is a randomized distributed protocol that controls access to shared resources between asynchronous processes [1]. All of the consensus models used in our experiments have N=4 asynchronous processes. Each has a different value for the parameter K, which defines the number of coin-flips for a process to make a decision.

4.2.2 Zeroconf

Zeroconf is a protocol that is used to assign IP addresses to devices that join a network (IPv4) [4]. All of the Zeroconf models used in our experiments have N=1000 devices in the network. Each has a different value of the parameter K, which defines the number of ARP packets sent.

Table 4.1

Characteristics of test problems

Consensus (N=4)	# of states	# of actions	# of transitions
K=2	22,656	60,544	75,232
K=4	43,136	115,840	144,352
K=8	84,096	226,432	282,592
K=16	166,016	447,616	559,072
K=32	329,856	889,984	1,112,032
Zeroconf (N=1000,err=0.01)			
K=1	31,954	57,482	73,318
K=2	89,586	164,169	207,825
K=3	179,774	331,425	416,688
K=4	307,768	569,227	712,132

4.3 Experiments

We implemented the algorithms for computing bounds according to Theorem 1 described in Chapter 3, and computed upper and lower bounds for different size examples of the ZeroConf and Consensus models. Examples were tested on machine with 3.5GHz i7 processor and 16GB of RAM, running Ubuntu 12.04.

Tables 4.2 and 4.3 show the results for upper bounds. The running times are averages over five runs of the algorithm for each m separately. To allow the quality of the bounds to be assessed, Tables 4.4 and 4.5 also show the optimal values N^*.

It is important to notice that at the point where m first becomes large enough that $\rho_m > 0$ (highlighted in the tables), the probabilities are extremely small and the upper bounds are very large. However, if we increase the value of m by letting the value iteration algorithm for computing minimum reachability probabilities run longer, the upper bounds very quickly decrease until they are close to optimal values. The convergence of the upper bounds for the ZeroConf examples is especially fast and dramatic. For the Consensus examples, the convergence was periodic and slower, although the reduction in the bounds is still dramatic.

From this result, it is clear that we should not stop the algorithm for computing upper bounds as soon as we have finite bounds, but continue to run it until we have high-quality bounds. For the ZeroConf examples, it only requires running the algorithm 20% longer. For the Consensus examples, on the other hand, we ran the algorithm up to 10 times longer, although good bounds were obtained before then.

Table 4.2

Upper bounds for initial state of ZeroConf problem

K = 1				K = 2			
m	ρ	UB value	UB (s)	m	ρ	UB value	UB (s)
96	3.333E-23	1.208E+17	0.860	102	3.333E-25	6.759E+19	2.401
97	6.657E-21	6.033E+14	0.867	103	6.953E-23	3.081E+17	2.487
98	6.323E-19	6.284E+12	0.891	104	6.944E-21	2.859E+15	2.711
99	3.864E-17	1.026E+11	0.920	105	4.419E-19	4.043E+13	2.789
100	1.668E-15	2.343E+09	0.866	106	2.011E-17	7.716E+11	2.660
101	5.480E-14	7.029E+07	0.971	107	6.970E-16	1.853E+10	2.745
102	1.420E-12	2662610.716	0.885	108	1.911E-14	5.340E+08	2.693
103	2.975E-11	123729.589	0.984	109	4.250E-13	1.786E+07	2.737
107	8.985E-07	109.813	0.940	115	1.280E-06	116.895	2.590
108	7.715E-06	108.265	0.905	116	8.753E-06	116.273	2.822
109	5.606E-05	109.027	0.947	117	5.113E-05	117.046	2.750
K = 3				K = 4			
107	3.333E-27	1.503E+22	5.092	112	3.333E-29	9.469E+23	9.454
108	7.248E-25	6.652E+19	5.150	113	7.544E-27	3.843E+21	9.272
109	7.561E-23	6.048E+17	5.513	114	8.206E-25	3.188E+19	9.046
110	5.037E-21	8.427E+15	5.869	115	5.711E-23	4.011E+17	9.713
111	2.406E-19	1.593E+14	6.166	116	2.856E-21	6.695E+15	9.149
112	8.770E-18	3.800E+12	6.386	117	1.093E-19	1.368E+14	9.461
113	2.536E-16	1.079E+11	5.657	118	3.324E-18	3.310E+12	9.444
114	5.969E-15	3.501E+09	4.825	119	8.253E-17	9.769E+10	9.720
122	1.348E-06	124.570	6.268	129	1.111E-06	130.607	10.212
123	7.550E-06	123.432	6.064	130	5.270E-06	130.316	10.030
124	3.659E-05	124.086	6.022	131	2.204E-05	131.071	10.410

Table 4.3

Upper bounds for initial state of Consensus problem

K = 2				K = 8			
m	ρ	UB value	UB (s)	m	ρ	UB value	UB (s)
41	9.766E-4	41871.25	0.378	113	5.821E-11	1.941E+12	2.912
47	3.662E-3	12773.82	0.425	119	5.675E-10	2.097E+11	3.350
53	0.008362	6295.78	0.472	125	2.987E-9	4.185E+10	3.147
59	0.015076	3880.09	0.562	131	1.126E-8	1.163E+10	3.598
65	0.023636	2722.10	0.491	137	3.410E-8	4.017E+09	3.861
71	0.033801	2076.35	0.582	143	8.811E-8	1.623E+09	4.242
77	0.045312	1677.92	0.622	149	2.017E-7	7.385E+08	4.520
83	0.057918	1413.73	0.677	155	4.197E-7	3.693E+08	4.582
257	0.459196	553.56	1.932	2567	0.461404	5557.44	66.764
263	0.470126	553.43	1.937	2573	0.462485	5557.41	68.428
269	0.480838	553.57	1.873	2579	0.463564	5557.42	71.134
K = 4				K = 16			
65	3.815E-6	1.704E+07	0.999	209	1.355E-20	1.542E22	11.839
71	2.193E-5	3236747.23	1.071	215	2.406E-19	8.938E20	11.934
77	7.176E-5	1072849.70	1.161	221	2.227E-18	9.924E19	12.327
83	1.759E-4	471630.31	1.460	227	1.431E-17	1.586E19	13.163
89	3.600E-4	247162.75	1.382	233	7.180E-17	3.245E18	13.585
95	6.498E-4	146130.98	1.555	239	2.993E-16	7.984E17	12.604
101	0.001070	94329.17	1.511	245	1.079E-15	2.270E17	14.306
107	0.001643	65084.63	1.330	251	3.458E-15	7.257E16	13.828
761	0.461944	1641.35	9.912	9407	0.462093	20351.38	495.442
767	0.465604	1641.33	10.068	9413	0.462388	20351.37	508.060
773	0.469240	1641.39	10.164	9419	0.462682	20351.38	471.312

For the ZeroConf examples, it is interesting to notice that the upper bounds converge to the value of m, while the maximum number of stages is less than m. This observation suggests that the upper bounds of Theorem 1 can be further improved by replacing the quantity m in the formula with a tighter bound on the expected number of stages. We leave this improvement for future work.

Tables 4.4 and 4.5 show lower bounds for the ZeroConf and Consensus examples. Recall that the lower bounds are computed by evaluating the policy that minimizes the m-stage probability of reaching the target state. The results show that the lower bounds computed by this method are very close to optimal.

Finally, we note that the effectiveness of the method for computing lower bounds depends on properly specifying the number of iterations of the policy evaluation (PE) algorithm. In our experiments, we picked a number of iterations for the PE algorithm that seemed to work well, and, for the Consensus examples, increased the number with the size of the problem. Recent work of K. Siddiqui [13] suggests an automatic method for determining the number of iterations of the PE algorithm that may improve performance.

Table 4.4

Lower bounds for initial state of ZeroConf problem

ZeroConf	N^*	LB value	# of iters.	LB (sec.)
K=1	16.125	16.125	100	0.558
K=2	20.961	20.961	100	1.138
K=3	25.850	25.850	100	2.500
K=4	30.748	30.748	100	3.111

Table 4.5

Lower bounds for initial state of Consensus problem

Consensus	N^*	LB value	# of iters.	LB (sec.)
K=2	362.895	97.209	2000	7.096
K=4	1082.056	99.984	4000	14.821
K=8	3664.105	3416.642	8000	110.804
K=16	13321.644	10258.167	16000	428.866
K=32	49420.883	26805.916	32000	1667.853

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we have shown how to derive finite upper bounds for the maximum-time SSP problem. To our knowledge, no one has studied finite upper bounds for this problem before. The approach we have developed allows a tradeoff between the time spent computing the bounds and the quality of the bounds. After the first finite bounds are found, our results show that the bounds improve dramatically with very little additional computation. We also described a related method for computing very good lower bounds.

The maximum-time SSP problem plays an important role in the probabilistic model checker PRISM, and we implemented our algorithm for computing bounds in PRISM. A next step is to use the bounds in a branch-and-bound algorithm for solving the maximum-time SSP problem. We leave it for future work to test the effectiveness of the bounds in this approach.

We briefly mentioned the possibility of tightening the bounds proved in Theorem 1. In particular, the first term m in the bounds of Equation 3.3 could potentially be improved by replacing it with a quantity that represents the expected number of stages to reach the target state when the target state is reached in m or fewer stages. We leave potential improvement of the bounds of Theorem 1 for future work.

Another potential direction for future work is to extend our approach to computing bounds for the maximum-time SSP problem to the more general maximum-cost SSP problem.

REFERENCES

[1] J. Aspnes and M. Herlihy, "Fast Randomized Consensus Using Shared Memory," *Journal of Algorithms*, vol. 11, no. 3, Sept. 1990, pp. 441–461.

[2] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1, 3rd edition, Athena Scientific, Belmont, Massachusetts, 2005.

[3] D. P. Bertsekas and J. N. Tsitsiklis, "An Analysis of Stochastic Shortest Path Problems," *Mathematics of Operations Research*, vol. 16, 1991, pp. 580–595.

[4] S. Cheshire, B. Adoba, and E. Gutterman, "Dynamic configuration of IPv4 link local addresses," Available from http://www.ietf.org/rfc/rfc3927.txt.

[5] L. De Alfaro, "Computing Minimum and Maximum Reachability Times in Probabilistic Systems," *CONCUR99 Concurrency Theory*, vol. 1664, Springer Berlin Heidelberg, 1999, pp. 66–81.

[6] M. Duflot, M. Kwiatkowska, G. Norman, D. Parker, S. Peyronnet, C. Picaronny, and J. Sproston, *FMICS: A Survey of Application*, chapter Practical Applications of Probabilistic Model Checking to Communication Protocols, John Wiley & Sons, Inc., Hoboken, New Jersey, 2012, pp. 133–150.

[7] J. Eaton and L. Zadeh, "Optimal pursuit strategies in discrete-state probabilistic systems," *Journal of Basic Engineering*, vol. 84, no. 1, 1962, pp. 23–29.

[8] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated Verification Techniques for Probabilistic Systems," *Formal Methods for Eternal Networked Software Systems*. 2011, vol. 6659 of *LNCS*, pp. 53–113, Springer.

[9] E. A. Hansen, I. Abdoulahi, and J. Shi, "A Branch-and-Bound Approach to Verification of Markov Decision Processes," unpublished, 2014.

[10] PRISM, "Probabilistic Model Checker,", `http://www.prismmodelchecker.org/casestudies/index.php` (accessed 30-May-2014).

[11] PRISM, "Probabilistic Model Checker,", `http://www.prismmodelchecker.org/` (accessed 30-May-2014).

[12] M. L. Puterman, *Markov Decision Processes: Discrete Dynamic Stochastic Programming*, John Wiley Chichester, Hoboken, New Jersey, 1994.

[13] K. Siddiqui, *A Simple Method to Improve the Performance of Modified Policy Iteration*, undergraduate honors thesis, Mississippi State University, Mississippi State, Mississippi, 2014.