Tarski Fixed Point Computation and Related Problems in Algorithmic Game Theory

Angus Joshi

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2024

Abstract

Tarski's fixed point theorem implies the existence of fixpoints of monotone functions on complete lattices, and can be used in a game-theoretic context where equilibria in certain games manifest as fixpoints of monotone functions. This has motivated the study of the computational complexity of finding so-called tarski fixed points in recent years. In this dissertation I provide an exposition of the TARSKI problem, three related problems in algorithmic game theory, and the state of the art in upper bounds for the TARSKI problem. I also provide an implementation of the aforementioned algorithms applied to the three problems and results from practical testing. The general consensus from practical tests is that the complex, recently developed algorithms with stronger asymptotic bounds perform worse in practice in the included cases than simpler algorithms which have been known for much longer.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Angus Joshi*)

Acknowledgements

Thank you to my supervisor Kousha Etesasmi, my parents, and my rubber duck George for ongoing support.

Table of Contents

Chapter 1

Introduction

1.1 Overview

Fixpoint theorems have proven to be fundamental in the study of equilibria in game theory and economic theory more generally. One such theorem due to Alfred Tarski in [22] proves the existence of fixpoints of monotone functions on complete lattices, and many problems in algorithmic game theory reduce to computing such a fixpoint[9]. Recent years have seen a flurry of results in the computational complexity of tarski fixed-point computation [6, 9, 10, 3], though many problems remain open.
In this dissertation I discuss a variant of tarki fixed-point computation I call the TARSKI problem, three related problems in algorithmic game theory which reduce to TARSKI, and the state of the art in upper bounds for TARSKI. The currently best-known algorithms are also implemented and tested on random instances of the three described problems, reaching the conclusion that the recent improvements in asymptotic upper bounds do not translate to practical performance benefit in the cases tested here.

1.2 Contributions

The main contributions of this dissertation are as follows,

- an exposition of the theoretical development on upper bounds for the TARSKI problem, and reduction from related problems is given with highlights including, proofs of the main lemmas for the so-called inner algorithm in [10] are proven in a simplified context in section 4.2, and a complete polynomial-time reduction from the ARRIVAL problem to TARSKI is given in section 3.1,
- a minor error in [9, Proposition 6.1.] is pointed out, with suggested rectification included in lemma 3.3.6,
- a novel implementation of the recently developed complex algorithms for the TARSKI problem is tested on randomly generated instances of ARRIVAL, simple stochastic games, and shapley's stochastic games, giving evidence that the improved asymptotic upper bounds do not result in a practical performance increase.

Chapter 2

Background

2.1 Lattices, Monotone Functions, and Fixpoints

For the purposes of this dissertation, I'm primarily concerned with an order-theoretic characterization of lattices. I begin with some definitions.

Definition 2.1.1 (Poset)**.** A *partially ordered set*, or *poset* is a set S with a binary relation $\leq$ on S such that the following axioms hold,

- For all $s \in S$, $s \leq s$ (reflexivity),
- for all $s,t,u \in S$, if $s \leq t$ and $t \leq u$ then $s \leq u$ (transitivity),
- for all $s,t \in S$, if $s \leq t$ and $t \leq s$ then $s = t$ (antisymmetry).

Definition 2.1.2 (Least/Greatest Upper/Lower Bounds)**.** Let $(S, \leq)$ be a partially ordered set and $A \subseteq S$. A *greatest lower bound* of A is a $l \in S$ such that for all $a \in A$ $a \geq l$, and whenever $l' \in S$ also satisfies for all $a \in A$ $a \geq l'$ I have $l \geq l'$. When it exists it is denoted $l = \bigwedge A$. A *least upper bound* of A is a $u \in S$ such that for all $a \in A$ $a \leq u$, and whenever $u' \in S$ also satisfies for all $a \in A$ $a \leq u'$ I have $u \leq u'$. When it exists it is denoted $u = \bigvee A$.

Remark 2.1.3. Least upper bounds and greatest lower bounds need not exist in general. Take for example $(\mathbb{Z}, \leq)$ which is claimed to be a partial ordering. $\mathbb{Z}$ is a subset of $\mathbb{Z}$, but clearly $\mathbb{Z}$ has no upper bounds. Boundedness is also in general not sufficient.

Lemma 2.1.4 (Least/Greatest Upper/Lower bounds are unique)**.** *Let L be a lattice, $A \subseteq L$ and u, u' least upper bounds of A. Then $u = u'$. If l, l' are greatest lower bounds of A then $l = l'$.*

Proof. The second part of the definition on a least upper bound gives $u \leq u'$ and $u' \leq u$. Then antisymmetry of a partial order gives $u = u'$. That $l = l'$ follows similarly. ■

Definition 2.1.5 (Complete Lattice)**.** Let $(L, \leq)$ be a partially ordered set. $(L, \leq)$ is a *complete lattice* if for all $A \subseteq L$ there is a least upper bound $u = \bigvee A$ and a greatest lower bound $l = \bigwedge A$.

Definition 2.1.6 (Product Lattice)**.** Given two complete lattices $(L, \leq)$ and $(L', \leq')$ the *product lattice* $(L \times L, \leq \times \leq')$ is the cartesian product of the underlying sets with $(l, l') \leq \times \leq' (m, m')$ if and only if $l \leq m$ and $l' \leq m'$ for all $l, m \in L$ and $l', m' \in L'$.

That a finite product of complete lattices is also a complete lattice is taken as fact. The notion of a complete sublattice will in turn be useful as well.

Definition 2.1.7 (Sublattice)**.** Let $(L, \leq)$ be a lattice. A complete sublattice $(M, \leq)$ is a subset $M \subseteq L$ such that for all $N \subseteq M$ I have $\bigvee N \in M$ and $\bigwedge N \in M$.

For the rest of the dissertation where the ordering is clear from context, I will denote a lattice purely by it's underlying set.

Notation 2.1.8. For $N \in \mathbb{Z}_{\geq 1}$ I use the notation $[N] = \{1, ..., N\}$.

It is clear that any finite totally ordered set is a complete lattice.

Corollary 2.1.9. *For $d \in \mathbb{Z}_{\geq 1}$ let $[N]^d = \prod_{i=1}^{d} [N]$. Then $[N]^d$ is a lattice. The ordering is defined as $(l_1, ..., l_d) \leq (l'_1, ..., l'_d)$ if and only if $l_i \leq l'_i$ for each $i \in \{1, ..., d\}$.*

At times I will also need continuous lattices. I take the result from elementary analysis that all bounded subsets of the real numbers have a least upper bound and greatest lower bound in the real numbers, and closedness, to find $[a, b] \subseteq \mathbb{R}$ is a complete lattice for all $a, b \in \mathbb{R}$ under the usual ordering.

Corollary 2.1.10. *For $d \in \mathbb{Z}_{\geq 1}$ let $[0,1]^d = \prod_{i=1}^{d} [0,1]$. Then $[0,1]^d$ is a lattice. The ordering is defined as $(l_1, ..., l_d) \leq (l'_1, ..., l'_d)$ if and only if $l_i \leq l'_i$ for each $i \in \{1, ..., d\}$.*

The focus will be on finding so-called fixpoints of monotone functions on a lattice.

Definition 2.1.11 (Monotone Function)**.** Let L be a lattice. Then a function $f : L \to L$ is *monotone* if whenever $l, l' \in L$ with $l \leq l'$ I have $f(l) \leq f(l')$.

Definition 2.1.12 (Fixpoint)**.** Let S be a set and $f : S \to S$. Then $s \in S$ is a *fixpoint* of f if $f(s) = s$.

The notion of a monotone point will also be useful.

Definition 2.1.13 (Monotone Point)**.** Let L be a lattice and $f : L \to L$ be monotone. Then $x \in L$ is a *monotone point* if either $f(x) \geq x$ or $f(x) \leq x$. It is *monotone up* if $f(x) \geq x$ and *monotone down* if $f(x) \leq x$.

Notation 2.1.14. Let L be a complete lattice and $a, b \in L$. The notation $[a, b] = \{x \in L : a \leq x \leq b\}$ is sometimes used. The notations $[a, \infty) = \{x \in L : a \leq x\}$ and $(-\infty, b] = \{x \in L : x \leq b\}$ are also used. The reader can verify that these all define complete sublattices of L.

And now for Tarski.

Theorem 2.1.15 (Tarski's Fixed Point Theorem [22])**.** *Let L be a complete lattice and $f : L \to L$ a monotone function. Define $\mathrm{Fix}(f) = \{x \in L : f(x) = x\}$. Then $\mathrm{Fix}(f)$ is a complete lattice under the same ordering as L. In particular, $\mathrm{Fix}(f)$ contains a least fixpoint $l = \bigwedge \mathrm{Fix}(f)$.*

2.2 Matrix Games

Basic theory of matrix games is required for section 3.3 so is included here. Some prior knowledge on game theory is assumed.

Notation 2.2.1. Let k be a field and $n,m \in \mathbb{Z}_{>0}$. $\mathbf{Mat}(k, n \times m)$ denotes the set of all $n \times m$ matrices with entries in k.

Definition 2.2.2 (Matrix Game)**.** A *matrix game* is a zero-sum game played by two players called the maximizer and minimizer respectively with strategy sets $[n]$ and $[m]$ respectively. Players choose an action $i \in [n]$ and $j \in [m]$, then receive payoff $A_{i,j}$ and $-A_{i,j}$ respectively where $A \in \mathbf{Mat}(\mathbb{Q}, n \times m)$. A *mixed strategy* for the maximizer is a probability distribution on $[n]$. That is, a vector $x \in \mathbb{Q}^n$ such that $x_i \geq 0$ for each $i \in [n]$ and $\sum_i x_i = 1$. The set of all mixed strategies for the maximizer is denoted X. A mixed strategy for the minimizer is defined similarly as a probability distrubution $y \in \mathbb{Q}^n$ on $[m]$. The set of all mixed strategies for the minimizer is denoted Y. The *expected payoff* of a matrix game A under mixed strategies $x \in X$ and $y \in Y$ is defined to be $U(x,y) = x^T Ay$ which is easily seen to be the expected payoff the maximizer receives when both players independently and simultaneously choose random strategies according to probability distributions x and y.

They key result in the theory of matrix games is the so-called minimax theorem due to Von Nuemann. A proof can be found in [4, Chapter 15].

Theorem 2.2.3 (Minimax, [4])**.** *Let $A \in \mathbf{Mat}(\mathbb{Q}, n \times m)$ be a matrix game, X and Y the sets of mixed strategies for the maximizer and minimizer respectively. Then,*

$$\max_{x \in X} \min_{y \in Y} x^T Ay = \min_{y \in Y} \max_{x \in X} x^T Ay.$$

This yields a notion of 'value' of a matrix game.

Definition 2.2.4. Let $A \in \mathbf{Mat}(\mathbb{Q}, n \times m)$ be a matrix game, X and Y the sets of mixed for the maximizer and minimizer respectively. The *value* of A is defined $\mathrm{val}(A) = \max_{x \in X} \min_{y \in Y} x^T Ay$.

Fortunately computing the value of a matrix game is computationally not hard.

Proposition 2.2.5. *Let $A \in \mathbf{Mat}(\mathbb{Q}, n \times m)$ be a matrix game. Then* $\mathrm{val}(A)$ *can be computed in time polynomial with the encoding size of A.*

For details on this the reader is again referred to [4, Chapter 15] where it is shown that the problem can be reduced to a sufficiently small linear programming problem, for which polynomial time algorithms are known.

2.3 Basic Algorithms and Lower Bounds

Tarski's fixed point theorem gives rise to a natural computational problem.

Definition 2.3.1 (TARSKI)**.** The problem $\textsc{Tarski}(N,d)$ is, given oracle access to a monotone function $f : [N]^d \to [N]^d$, find a point $x^* \in [N]^d$ such that $f(x^*) = x^*$.

And now for the first basic algorithm for solving TARSKI.

Notation 2.3.2. For $k \in \mathbb{Z}_{\geq 0}$ the notation $\vec{k} = (k, ..., k)$. It is assumed that the dimensionality of this 'vector' is clear from context.

Algorithm 1 Kleene Tarski Iteration. An iterative algorithm for finding fixpoints.

```
1: procedure KLEENETARSKI(monotone f : [N]^d → [N]^d)
2:     x ← 1⃗
3:     while x ≠ f(x) do
4:         x ← f(x)
       return x
```

Correctness of the algorithm if it terminates is clear, so all that is needed it a bound on it's runtime.

Lemma 2.3.3. KLEENETARSKI *always terminates in time* $O(Nd)$.

Proof. By definition of the lattice, for all $x \in [N]^d$ I have $\vec{1} \leq x$, and in particular $\vec{1} \leq f(\vec{1})$. Then by induction for all $i \in \mathbb{Z}_{\geq 0}$, $f^i(\vec{1}) \leq f^{i+1}(\vec{1})$, where the convention that $f^0(x) = x$ is used. So suppose for a contradiction for some $j > Nd$ that for all $i \leq j$, $f^i(\vec{1}) < f^{i+1}(\vec{1})$. By integrality, $\|f^{i+1}(x)\|_1 \geq \|f^i(x)\|_1 + 1$. It follows that $\|f_j(x)\|_1 > Nd$. But this implies that $\|f_j(x)\|_1 > \|\vec{N}\|_1$, which is a contradiction of the definition of f. So for some $j \leq Nd$, $f^j(\vec{1}) = f^{j+1}(\vec{1})$. ■

Theorem 2.3.4. *The query complexity of* TARSKI(N,d) *is* $O(Nd)$.

The fixpoint returned by KLEENETARSKI isn't just any old fixpoint.

Lemma 2.3.5. *Let x be the fixpoint returned by* KLEENETARSKI. *Then x is the least-fixpoint of f. That is, for all other fixpoints* $y \in [N]^d$, $y \leq x$.

Proof. Let $(f^i(\vec{1}))_{i \in \mathbb{Z}_{\geq 0}}$ be the sequence of points generated in KLEENETARSKI. I will show inductively that $f^i(\vec{1}) \leq y$ for all $i \in \mathbb{Z}_{\geq 0}$. For the base case, by construction of the lattice $f^0(\vec{1}) = \vec{1} \leq y$. Suppose $f^{i-1}(\vec{1}) \leq y$. Then by monotonicity of f, $f(f^{i-1}(\vec{1})) = f^i(\vec{1}) \leq f(y)$. But y is a fixpoint so $f(y) = y$ and $f^i(\vec{1}) \leq y$. ■

It should be emphasized that this is algorithm does *not* run in time polynomial with the encoding size of problem instances. For numbers of size 2^n can be represented with n bits of information. At the time of writing it is not known whether or not this problem is solvable in polynomial time. Etessami et al. gave the current best known lower bound on the query complexity of TARSKI, along with other complexity-theoretic results on the problem.

Theorem 2.3.6 ([9])**.** *The query complexity of* TARSKI(N,d) *is* $\Omega(\log^2 N)$.

Dang, Qi, and Ye gave an algorithm for solving the TARSKI problem[6] using a variant of the well known binary search algorithm. The details of their algorithm are instructive to the workings on the improved algorithms detailed later, so they are given below.

Notation 2.3.7. Given a tuple $x = (x_1, ..., x_n)$ for $i \in [n]$ the notation $x_{-i} = (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$. That is, it drops the i-th coordinate of the tuple.

Definition 2.3.8 (Slice)**.** Let $(S_i)_{i\in[d]}$ be totally ordered sets, $L = \prod_{i\in[d]} S_i$ be their product lattice, and $f : L \to L$ be monotone. Then a *slice* of f is a choice of coordinate $i \in [d]$, and a choice of value $x_i \in S_i$, defining a new function $f_s : L_{-i} \to L_{-i}$ with $f_s((l_1, ..., l_{d-1})) = f((l_1, ..., l_{i-1}, x_i, l_i, ..., l_{d-1}))_{-i}$.

Lemma 2.3.9. *Let $f : [N]^d \to [N]^d$ be monotone. Then for any $i \in [d]$, $x_i \in [N]$ the slice $f_s : [N]^{d-1} \to [N]^{d-1}$ at i with value x_i is monotone.*

Proof. Suppose $l, l' \in [N]^{d-1}$ with $l = (l_1, ..., l_{d-1})$, $l' = (l'_1, ..., l'_{d-1})$, and $l \leq l'$. By reflexivity, $x_i \leq x_i$, so $(l_1, ..., x_i, l_i, ..., l_{d-1}) \leq (l'_1, ..., x_i, l'_i, ..., l_{d-1})$, and $f_s(l) \leq f_s(l')$ follows by monotonicity of f. ■

Lemma 2.3.10. *Let $f : [N]^d \to [N]^d$ is monotone, and $x \in [N]^d$ be such that $x \leq f(x)$. Then f restricts to a monotone function $f|_{[x,\infty)} : [x, \infty) \to [x, \infty)$. Similarly, if $x \geq f(x)$ then f restricts to a monotone function $f|_{(-\infty,x]} : (-\infty, x] \to (-\infty, x]$.*

Proof. I need to show that if $x \leq f(x)$ then for all $y \in [x, \infty)$, $f(y) \in [x, \infty)$. By construction, $y \geq x$, and by monotonicity $f(y) \geq f(x)$. But $f(x) \geq x$, so $f(y) \geq x$, and $f(y) \in [x, \infty)$. The second part is similar. ■

Lemma 2.3.11. *Let $f : [N] \to [N]$ be monotone. Then a fixpoint of f can be found in $O(\log N)$ queries of f.*

Proof. Choose $x = \lfloor \frac{N}{2} \rfloor$. $[N]$ is totally ordered, so exactly one of the following hold; $x < f(x)$, $x = f(x)$, $x > f(x)$. If $x = f(x)$ then I'm done. If $x < f(x)$ then by lemma 2.3.10 f restricts to a monotone function $f|_{[x,\infty)}$, and a fixpoint of $f|_{[x,\infty)}$ is clearly also a fixpoint of f. Similarly, if $x > f(x)$ then f restricts to $f|_{(\infty,x]}$. This enables a recursion on the smaller sublattice. Finally, noting that a fixpoint can be found trivially in the one-point set in a constant number of queries, since the search space is halved every recursive call the algorithm terminates in $O(\log N)$ queries. ■

The algorithm of Dang, Qi, and Ye can be seen in algorithm 2.

Lemma 2.3.12. DANGQIYE *returns a fixpoint of f if it terminates.*

Proof. The algorithm only returns if it satisfies the condition on line 11. At this point, $\vec{x_s}$ is a fixpoint of f_s, so it follows that $f(\vec{x})_i = \vec{x}_i$ for $i \in [d-1]$. The condition ensures that also $\vec{x}_d = f(\vec{x})_d$, and $\vec{x}$ is a fixpoint of f. ■

Lemma 2.3.13. DANGQIYE *terminates in at most $O(\log^d N)$ queries to f.*

Proof. By induction. The base case follows from lemma 2.3.11. Suppose DANGQIYE uses at most $O(\log^{d-1} N)$ queries to solve the $d-1$ dimensional case. If the condition on line 11 fails, note that $\vec{x}$ is a monotone point, so lemma 2.3.10 guarantees that the function restricts to the smaller lattice bounded by lines 14 or 16. The d-th dimension is

Algorithm 2 [6]. A binary-search-like algorithm for finding fixpoints.

```
1:  procedure DangQiYe(monotone f : [N]^d → [N]^d)
2:      ⊥ ← 1⃗
3:      ⊤ ← N⃗
4:      return DangQiYeRec(f, ⊥, ⊤)
5:  procedure DangQiYeRec(monotone f : [N]^d → [N]^d, ⊥, ⊤)
6:      while true do
7:          mid_d ← ⌊(⊥_d + ⊤_d)/2⌋
8:          f_s ← the slice of f at d with value mid_d
9:          x⃗_s ← DangQiYe(f_s, ⊥_{-d}, ⊤_{-d})
10:         x⃗ ← ((x⃗_s)_1, ..., (x⃗_s)_{d-1}, mid_d)
11:         if x⃗_d = f(x⃗)_d then
12:             return x⃗
13:         if mid_d < f(x⃗)_d then
14:             ⊥ ← x⃗
15:         if mid_d > f(x⃗)_d then
16:             ⊤ ← x⃗
```

shrunk by a factor of $\frac{1}{2}$ every iteration of the loop, so will require at most $\log N$ recursive calls to the $d-1$ dimensional algorithm. So the algorithm terminates using at most $O(\log N) \cdot O(\log^{d-1} N) = O(\log^d N)$ queries to f. ■

Theorem 2.3.14 ([6])**.** *The query complexity of* $\textsc{Tarski}(N,d)$ *is* $O(\log^d N)$.

Simply combining algorithm 2, and theorem 2.3.6 gives a tight bound on the 2 dimensional problem.

Corollary 2.3.15 ([9])**.** *The query complexity of* $\textsc{Tarski}(N,2)$ *is* $\Theta(\log^2 N)$.

Remark 2.3.16. The reader may notice that there are many other natural fixpoint computation problems for lattices, and wonder what motivates the specific problem detailed above. The problem of computing any fixpoint of a monotone function on a general lattice is known to be intractable. In particular, it was shown in [1] that for any randomized algorithm there are general lattices of N elements which require $\Omega(N)$ queries with high probability to find a fixpoint. Focussing on a specific case such as $[N]^d$ is therefore justified. Further, the case of computing least fixpoints of even one dimensional functions $f : [2^n] \to [2^n]$ is proven to be **NP**-hard in [9] which motivates the problem only requiring an arbitrary fixpoint. As will also be demonstrated in chapter 3, as well as through other examples in [9], the specific problem I detail still has useful applications.

Chapter 3

Related Problems

In this chapter, I discuss three problems problems in algorithmic game theory which are polynomial-time reducible to TARSKI; ARRIVAL, simple stochastic games, and shapley's stochastic games, with the goal of motivating study on the TARSKI problem.

3.1 The Arrival Problem

The arrival problem is, given a directed graph with a particular structure and designated source and target vertex, decide whether or not a particular walk starting at the source ever reaches the target. In [14] a sub-exponential time algorithm for arrival is developed which reduces the problem to TARSKI of some form. It turns out that ARRIVAL is in fact polynomial time reducible to TARSKI. While this is not explicity stated in [14], essentially all of the theory for this reduction is included so credit goes to them. In this section, I give the basic definitions of the ARRIVAL problem as well as a complete proof of polynomial time reduction from ARRIVAL to TARSKI.

Definition 3.1.1 (Arrival Graph)**.** An *arrival graph* is a set of vertices V, a pair of vertices $s,t \in V$, and a pair of maps $s_0, s_1 : V \to V$.

Definition 3.1.2 (Arrival Walk)**.** Let (V,s,t,s_0,s_1) be an arrival graph. The *arrival walk* on this graph is a sequence of vertices $(v_i)_{i\in\mathbb{Z}_{\geq 0}} \in V$ such that $v_0 = s$, and $v_{i+1} = \begin{cases} s_0(v_i), & n_i \text{ even} \\ s_1(v_i), & n_i \text{ odd,} \end{cases}$ where n_i is the number of times v_i has appeared previously in the sequence.

A diagram of an example arrival graph is shown in in *fig*. 3.1. It is clear that the arrival walk for a particular arrival graph is entirely defined by the structure of the graph, which is what lead it to be called a zero player graph game in [7].

Definition 3.1.3 (ARRIVAL)**.** The ARRIVAL problem is, given an arrival graph (V,s,t,s_0,s_1), decide whether or not the arrival walk ever reaches t.

There is an obvious algorithm to solve the ARRIVAL problem; just simulate the walk. Cases of instances where t is not reachable pose a problem however - the walk must

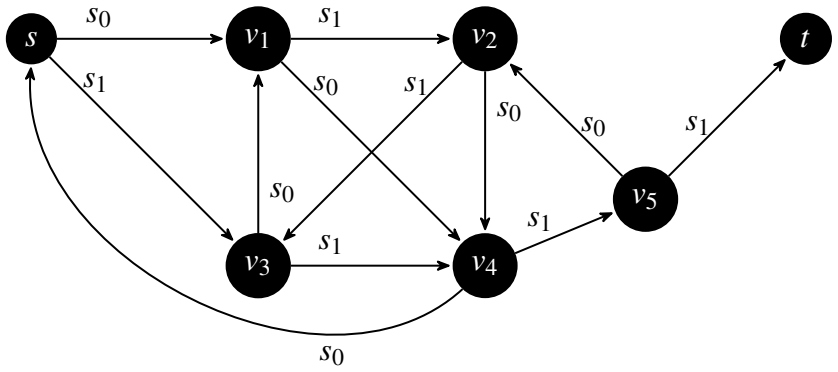

Figure 3.1: The goal of the arrival problem is to decide whether a partiular walk on a directed graph with a particular structure reaches the target. On successive visits to a particular vertex the outgoing edge taken alternates. In this example the walk begins $s \to v_1 \to v_4 \to s \to v_3 \to \ldots$.

cycle infinitely and never terminate! The following content demonstrates that this is a non issue.

Definition 3.1.4 (Hopeful and Desperation)**.** Let (V,s,t,s_0,s_1) be an instance of the ARRIVAL problem. A vertex $v \in V$ is *hopeful* if there is a path $v \to t$ in the directed graph defined with the vertex set V and edge set $E \subseteq V \times V$ with $(u,v) \in E$ if and only if either $s_0(u) = v$ or $s_1(u) = v$. The *desperation* of a hopeful vertex v is the length of the shortest path from v to t.

Lemma 3.1.5 ([7])**.** *Let (V,s,t,s_0,s_1) be an instance of the* ARRIVAL *problem. If $v \in V$ is hopeful, the arrival walk passes through v at most $2^{|V|}$ times.*

Proof. Begin by noting that if a vertex is hopeful, it's desperation is at most $|V|$. I perform an induction on the desperation of v. Suppose the desperation of $v \in V$ is 1. Then either $s_0(v) = t$ or $s_1(v) = t$. If $s_0(v) = t$, t will be reached after passing through v once. If $s_1(v) = t$ and $s_0(v) \neq t$ t will be reached after passing through v twice. In both cases v is passed through at most $2^1 = 2$ times.
Suppose that all hopeful vertices with desperation $d-1$ are passed through at most 2^{d-1} times. Then if $v \in V$ is hopeful with desperation d, for some hopeful $w \in V$ with desperation $d-1$ either $s_0(v) = w$ or $s_1(v) = w$. So at least every second passing of v, the walk will proceed to w. But the walk can pass through w at most 2^{d-1} times, so the walk can pass through v at most $2 \cdot 2^{d-1} = 2^d$ times. ■

Corollary 3.1.6. *Let (V,s,t,s_0,s_1) be an instance of the* ARRIVAL *problem. Then the arrival walk either reaches t, or reaches a vertex which is not hopeful.*

From corollary 3.1.6 it is clear that deciding an instance of the ARRIVAL problem is equivalent to deciding whether or not the arrival walk reaches t or reaches a vertex which is not hopeful.

Definition 3.1.7 (Processed Arrival)**.** Let (V,s,t,s_0,s_1) be an instance of the arrival

problem. It is without loss of generality to assume that the set of unhopeful vertices is non-empty. Let $\sim$ be the equivalence relation on V generated by $u \sim v$ if u and v are both not hopeful. The *processed arrival problem* is a set of vertices $V' = V/\sim$, the canonical projections of s,t,s_0,s_1 into V', and a choice of representative $\bar{t} \in V'$ of all the non hopeful vertices in V.

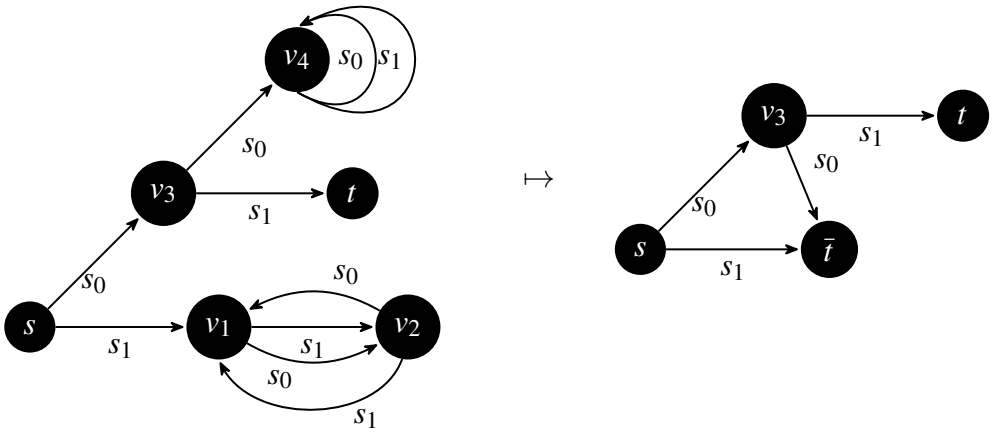

Figure 3.2: Instances of the arrival problem can be preprocessed so that every non-target vertex has a directed path to the target. An extra 'bad target' $\bar{t}$ is added which represents all the vertices with no directed path to the target t, and the problem becomes to decide which of the two targets is reached. From corollary 3.1.6 the walk in the resultant graph must be finite.

Noting that the set of non hopeful vertices can be easily computed in linear time with a breadth first search from t, from this point on I will refer to instances of the ARRIVAL problem exclusively as tuples $(V,s,t,\bar{t},s_0,s_1)$ constructed as above.

Corollary 3.1.8 ([7])**.** *The time complexity of the* ARRIVAL *problem is* $O(n \cdot 2^n)$.

Proof. I reason that the arrival walk on the processed instance $(V,s,t,\bar{t},s_0,s_1)$ has it's walk length bounded by $O(n \cdot 2^n)$. Every vertex $v \in V$ with $v \neq \bar{t}$ is hopeful with desperation at most n, so by corollary 3.1.6 can be passed through at most 2^n times. If the walk reaches t or $\bar{t}$ it terminates, and there are at most n vertices $w \in V$ such that $w \notin \{t,\bar{t}\}$, so the walk can take at most $n \cdot 2^n$ steps. ■

There are in fact instances of the ARRIVAL problem with exponentially long walks - as seen in fig. 3.3 - implying that the worst-case runtime of this algorithm is exponential. Recently a sub-exponential[1] upper bound for ARRIVAL was given in [14]. Interestingly, their algorithm involves a reduction from ARRIVAL to TARSKI. I will not detail the reduction used in the sub-exponential algorithm, but will spend the remainder of the section demonstrating a similar, yet simpler reduction from ARRIVAL to TARSKI.

[1] Specifically an algorithm running in time $O(2^{\sqrt{n}})$.

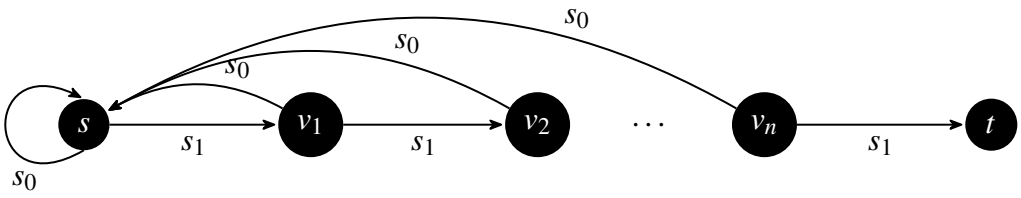

Figure 3.3: There are instances of the arrival problem with exponentially long walks. An induction on the number of steps to get from $s \to v_i$ shows that walk on the above takes at least $\Omega(2^n)$ steps to reach t.

Definition 3.1.9 (Switching Flow)**.** Let $(V,s,t,\bar{t},s_0,s_1)$ be an arrival graph. A *switching flow* is a pair of maps $f_0, f_1 : V \setminus \{t,\bar{t}\} \to \mathbb{Z}_{\geq 0}$ such that the following axioms hold. Let $f_{\text{in}}(v) = \sum_{\substack{w \in V \\ s_0(w)=v}} f_0(w) + \sum_{\substack{w \in V \\ s_1(w)=v}} f_1(w)$.

- For all $v \in V \setminus \{s,t,\bar{t}\}$, $f_{\text{in}}(v) = f_0(v) + f_1(v)$ (flow conservation),
- $f_{\text{in}}(s) = f_0(s) + f_1(s) - 1$ (source flow conservation),
- For all $v \in V$, $f_1(v) \leq f_0(v) \leq f_1(v) + 1$ (switching).

Notation 3.1.10. Throughout this section if $a \in \mathbb{Z}^d_{\geq 0}$ and (f_0, f_1) is the flow corresponding to a, the notation $f_{\text{in}}(v) = \sum_{\substack{w \in V \\ s_0(w)=v}} f_0(w) + \sum_{\substack{w \in V \\ s_1(w)=v}} f_1(w)$ will be used.

It was observed in [7] that the walk on an arrival graph can be characterized by a switching flow.

Lemma 3.1.11 ([7])**.** *Let $(V,s,t,\bar{t},s_0,s_1)$ be an instance of the* ARRIVAL *problem. Define $f_0 : V \setminus \{t,\bar{t}\} \to \mathbb{Z}_{\geq 0}$ by $f_0(v) =$ the number of times $s_0(v)$ is traversed in the arrival walk, and define f_1 similarly. Then (f_0, f_1) is a switching flow.*

Proof. Flow conservation and source flow conservation follow from the fact that the walk must walk out of a vertex if it walks in, minus the initial step it takes from the source. Switching follows from the nature of the walk taking the s_0 edge on even passes, and s_1 edge on odd passes. ■

I next establish a correspondence from arrival instances to monotone functions.

Definition 3.1.12 (Arrival Monotone Function)**.** Let $(V,s,t,\bar{t},s_0,s_1)$ be an instance of the arrival problem, $d = |V \setminus \{t,\bar{t}\}|$ and $(v_i)_{i \in [d]}$ be an enumeration of the vertices in $V \setminus \{t,\bar{t}\}$. The *arrival monotone function* is a function $f : \mathbb{Z}^d_{\geq 0} \to \mathbb{Z}^d_{\geq 0}$ defined coordinatewise as,

$$f((a_1,\ldots,a_d))_i = \begin{cases} \sum_{\substack{j \in [d] \\ s_0(v_j)=v_i}} \left\lceil \frac{a_j}{2} \right\rceil + \sum_{\substack{j \in [d] \\ s_1(v_j)=v_i}} \left\lfloor \frac{a_j}{2} \right\rfloor & v_i \neq s \\ 1 + \sum_{\substack{j \in [d] \\ s_0(v_j)=v_i}} \left\lceil \frac{a_j}{2} \right\rceil + \sum_{\substack{j \in [d] \\ s_1(v_j)=v_i}} \left\lfloor \frac{a_j}{2} \right\rfloor & v_i = s. \end{cases}$$

Lemma 3.1.13. *The arrival monotone function is monotone.*

Proof. Clearly the sum of monotone functions is also monotone, $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ are monotone, composition of monotone functions is monotone, linear functions with non-negative coefficients are monotone, and constant functions are monotone. This encompasses all components of the above function, which is therefore montone. ■

The monotone function was constructed precisely so that the following proposition holds.

Proposition 3.1.14. *Let $f : \mathbb{Z}^d_{\geq 0} \to \mathbb{Z}^d_{\geq 0}$ be an arrival monotone function, and $a = (a_1, ..., a_d) \in \mathbb{Z}^d_{\geq 0}$. Define $g_0(v_i) = \lceil \frac{a_i}{2} \rceil$ and $g_1(v_i) = \lfloor \frac{a_i}{2} \rfloor$. If $f(a) = a$ then (g_0, g_1) is a switching flow.*

Proof. For flow conservation, let $v_i \in V \setminus \{s,t,\bar{t}\}$. Then,

$$\begin{aligned}\sum_{\substack{j\in[d]\\ s_0(v_j)=v_i}} g_0(v_j) + \sum_{\substack{j\in[d]\\ s_0(v_j)=v_i}} g_1(v_j) &= \sum_{\substack{j\in[d]\\ s_0(v_j)=v_i}} \left\lceil \frac{a_j}{2} \right\rceil + \sum_{\substack{j\in[d]\\ s_0(v_j)=v_i}} \left\lfloor \frac{a_j}{2} \right\rfloor \\ &= f(a)_i \\ &= a_i \\ &= \left\lceil \frac{a_i}{2} \right\rceil + \left\lfloor \frac{a_i}{2} \right\rfloor \\ &= g_0(v_i) + g_1(v_1).\end{aligned}$$

Source flow conservation follows similarly. Switching is clear from the definition of $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$.

■

The next proposition draws useful connection between the arrival walk and monotone function.

Proposition 3.1.15. *Let $(V,s,t,\bar{t},s_0,s_1)$ be an instance of the problem, and f be the arrival monotone function. For $a \in \mathbb{Z}^d_{\geq 0}$ let $g_0(a) = \lceil \frac{a}{2} \rceil$ and $g_1(a) = \lfloor \frac{a}{2} \rfloor$. For $i \in \{x \in \mathbb{Z}_{\geq 0} | x < n\}$ where n number of steps to reach t or $\bar{t}$, let $(h^i_0 : [d] \to \mathbb{Z}_{\geq 0}, h_1 a^i : [d] \to \mathbb{Z}_{\geq 0})_{i\in\mathbb{Z}_{\geq 0}}$ be a sequence defined by $h^i_0(j) =$ the number of times the s_0 edge has been taken from v_j after i steps in the walk, and define $h^i_1(j)$ similarly for the s_1 edge. Then for each $j \in [d]$ $g_0(f^i(\vec{0}))_j = h^i_0(j)$ and $g_1(f^i(\vec{0}))_j = h^i_1(j)$.*

Proof. By induction. For the base case where $i = 0$ no edges have been crossed, so for each $j \in [d]$ $h^i_0(j) = h^i_1(j) = 0 = \frac{f^0(\vec{0})_i}{2} = 0$.
So suppose for some $i \in \mathbb{Z}_{\geq 0}$ the statement is true for all $j \in [i-1]$. Let $v_k \in V \setminus \{t,\bar{t}\}$ be the vertex the walk is at after $i-1$ steps, and v_l after $i-2$ steps. Then by the inductive hypothesis $f^{i-2}(\vec{0})_l + 1 = f^{i-1}(\vec{0})_l$ and for each $m \in [d] \setminus \{l\}$, $f^{i-2}(\vec{0})_m = f^{i-1}(\vec{0})_m$. It follows that $f^{i-1}(\vec{0})_k + 1 = f^i(\vec{0})_k$, and the switching property of the monotone function guarantees that the extra unit appears on the correct edge. ■

All of that is really just to say that in the case of monotone functions from the arrival problem, *iteration from the bottom of the lattice as in algorithm 1 is the walk.* This connection gives me some useful corollaries.

Corollary 3.1.16. *Let f be an arrival monotone function, (g_0, g_1) be the switching flow corresponding to an arrival walk as in lemma 3.1.11, and $a \in \mathbb{Z}^d_{\geq 0}$ be the fixpoint corresponding to this switching flow as in proposition 3.1.14. Then a is the least fixpoint of f.*

Proof. Using a similar argument to lemma 2.3.5, iteration from the bottom of the lattice gives the least fixpoint. But proposition 3.1.15 says that iteration from the bottom of the lattice is the walk and will give the fixpoint corresponding to the switching flow of the walk. ■

Lemma 3.1.17. *Let $f : \mathbb{Z}^d_{\geq 0} \to \mathbb{Z}^d_{\geq 0}$ be an arrival monotone function. If $a \in \mathbb{Z}^d_{\geq 0}$ and (f_0, f_1) the flow corresponding to a, then $\sum_{i\in[d]} f(a)_i = 1 + \left(\sum_{i\in[d]} a_i\right) - f_{in}(t) - f_{in}(\bar{t})$.*

Proof. Easily computed from the definitions. ■

Corollary 3.1.18. *Let A be an instance of the arrival problem and (g_0, g_1) a switching flow. Then either $g_{in}(t) = 1$ and $g_{in}(\bar{t}) = 0$, or $g_{in}(t) = 0$ and $g_{in}(\bar{t}) = 1$.*

Proof. The walk only terminates when it reaches either t or $\bar{t}$, so if (f_0, f_1) is the switching flow corresponding I must have exactly one of $f_{\text{in}}(t) = 1$ and $f_{\text{in}}(\bar{t}) = 0$ or $f_{\text{in}}(t) = 0$ and $f_{\text{in}}(\bar{t}) = 1$. By corollary 3.1.16 the walk corresponds to the least fixpoint, and for all other switching flows (g_0, g_1), at least one of $g_{\text{in}}(t) \geq 1$ or $g_{\text{in}}(\bar{t}) \geq 1$. Suppose for a contradiction that $g_{\text{in}}(t) + g_{\text{in}}(\bar{t}) \geq 2$ and let $a \in \mathbb{Z}^d_{\geq 0}$ be the point corresponding to (g_0, g_1). If f is the arrival monotone function corresponding to A then by lemma 3.1.17, $\sum_{i\in[d]} f(a)_i = 1 + \left(\sum_{i\in[d]} a_i\right) - g_{\text{in}}(t) - g_{\text{in}}(\bar{t}) \leq \sum_{i\in[d]} a_i - 1$. Which contradicts a being a fixpoint and (g_0, g_1) being a switching flow. ■

Remarkably, this implies that *any* switching flow is certificate to the walk reaching either t or $\bar{t}$. Further, fixpoints are switching flows so deciding the ARRIVAL problem is reducible to finding a fixpoint of a particular monotone function! I'm not quite finished yet however; the monotone functions of arrival instances considered so far have been on the infinite lattice $\mathbb{Z}^d_{\geq 0}$, but the TARSKI problem I defined is on the finite lattice $[N]^d$. This will turn out to not be an issue.

Notation 3.1.19. For $N \in \mathbb{Z}_{\geq 0}$ notation $[N]_0$ represents $[N] \cup \{0\}$.

Definition 3.1.20 (Bounded arrival monotone function)**.** Let $(V, s, t, \bar{t}, s_0, s_1)$ be an instance of the arrival problem and f be it's corresponding arrival function. Let $n = |V|$ and $N = 2^n$. The *bounded arrival monotone function* is a function $F : [N]^d_0 \to [N]^d_0$ defined coordinatewise as $F(a)_i = \min(f(a)_i, N)$.

Lemma 3.1.21. *Let F be a bounded arrival monotone function. Then F is monotone.*

Proof. Similarly to the proof of lemma 3.1.13, $\min(\cdot, N)$ is clearly monotone. Monotonicity then follows monotonicity of the arrival monotone function, and monotonicity being preserved under composition. ■

Lemma 3.1.22 ([14]). *Let $(V,s,t,\bar{t},s_0,s_1)$ be an instance of the arrival problem, $n=|V|$, $N=2^n$, $d=|V\setminus\{t,\bar{t}\}|$, $f:\mathbb{Z}^d_{\geq 0}\to\mathbb{Z}^d_{\geq 0}$ be the arrival monotone function, and $F:[N]^d_0\to[N]^d_0$ be the bounded arrival monotone function. If $a\in[N]^d_0$ satisfies $F(a)=a$, then $f(a)=a$.*

Proof. Begin by noting that for all $a\in[N]^d_0$, $f(a)\geq F(a)$. Suppose for a contradiction that $f(a)\neq F(a)$. Then $F(a)>f(a)$. From lemma 3.1.17, $\sum_{i\in[d]}f(a)_i\leq 1+\sum_{i\in[d]}a_i$. Since $F(a)=a$ I find $i\in[d]$, $f(a)_j=\begin{cases}a_j+1 & j=i\\ a_j & j\neq i.\end{cases}$. By definition $F(a)_i=\min(f(a)_i,N)$ so $a_i=N$ and $f(a)_i=N+1$. For $\sum_{i\in[d]}f(a)_i=1+\sum_{i\in[d]}a_i$ I require that $f_{\text{in}}(t)=0$ and $f_{\text{in}}(\bar{t})=0$. But corollary 3.1.6 combined with $a_i=N=2^n$ implies that the walk must have terminated. That is, either $f_{\text{in}}(t)>0$ or $f_{\text{in}}(\bar{t})>0$, which is a contradiction. ■

Theorem 3.1.23. ARRIVAL *is polynomial time reducible to* TARSKI.

3.2 Simple Stochastic Games

Simple stochastic games, as defined in [5], are a class of zero-sum games played on graphs with two players called the maximizer and minimizer respectively. For the purposes of this dissertation I will consider only β-stopping simple stochastic games which roughly speaking are simple stochastic games where at every stage the game automatically halts with probability β. Condon shows in [5] that these games necessarily have a rational value for rationally described instances of the problem, and further that the value can be achieved in pure stationary strategies which is to say that players can achieve optimal expected payoff by fixing a deterministic action for every one of their vertices prior to play commencing. The relationship to TARSKI then comes from [9], where Etessami et al. show that computing the *exact* value of (not necessarily β-stopping) simple stochastic games as well as a pure stationary strategy profile to achieve this value is polynomial-time reducible to TARSKI. This section will lay out the required definitions, and describe the aforementioned reduction to TARSKI in the special case of β-stopping simple stochastic games.

Definition 3.2.1 (β-stopping Simple Stochastic Game)**.** A β-*stopping simple stochastic game* is a directed graph $G=(V,V_p,V_{\max},V_{\min},E,v_0,t,\beta)$ with designated start vertex $v_0\in V$, target vertex $t\in V$, $\beta\in(0,1]\cap\mathbb{Q}$, a partition of $V\setminus\{t\}$ into three disjoint subsets $V_p,V_{\min},V_{\max}$, and a mapping $p:V_p\times V\to[0,1]\cap\mathbb{Q}$ such that for all $v_p\in V_p$, $v\in V$ if $(v_p,v)\notin E$ I have $p(v_p,v)=0$, for each $v_p\in V_p$ $\sum_{v\in V}p(v_p,v)=1$, and for every $v\in V\setminus\{t\}$ there is necessarily an edge $(v,w)\in E$ for some $w\in V$. A *play* in a simple stochastic game transpires as follows. A token is placed on the initial vertex of the game v_0. Let v_i be the vertex on which the token currently lies. Then at each step, the game halts with probability β. If it did not halt and $v_i\in V_{\max}$ ($V_{\min}$) then the maximizer (minimzer) chooses and edge $(v_i,v_{i+1})\in E$ for some $v_{i+1}\in V$. If $v_i\in V_p$ then an edge $(v_i,w)\in E$ is chosen randomly with probability $p(v_i,w)$. If $v_i=t$ then the game halts. When the game halts, the maximizer gets a payoff of 1 if the game reached t, and 0 otherwise. The payoff of the minimizer is the negative of the maximizer's.

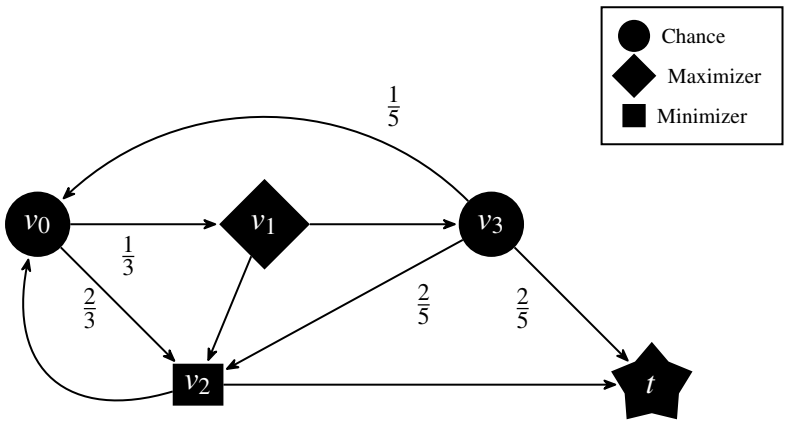

Figure 3.4: In β-stopping simple stochastic games, one of the two players aims to maximize the probability of play reaching the target t, while the other aims to minimize it.

It is clear that since $\beta > 0$ the game eventually halts with probability 1.

Definition 3.2.2 (Pure Stationary Strategy)**.** Let $G = (V, V_p, V_{\max}, V_{\min}, E, v_0, t)$ be a simple stochastic game. A *pure stationary strategy* for the maximizer is a mapping $\sigma : V_{\max} \to V$ with the requirement that for all $v \in V_{\max}$ $(v, \sigma(v)) \in E$. The set of all such pure stationary strategies for the maximizer is denoted S. A pure stationary strategy for the minimizer is a map $\tau : V_{\min} \to V$ such that for all $v \in V_{\min}$ $(v, \tau(v)) \in E$. The set of all such pure stationary strategies for the minimizer is denoted T. A *pure stationary strategy profile* is a pair $(\sigma, \tau) \in S \times T$.

Once you fix a pure stationary strategy profile (σ, τ) in a simple stochastic game, the resultant process is easily seen to be a discrete markov chain with two absorbing states corresponding to the to the game reaching t or halting elsewhere. The probability of reaching t from any particular vertex can then be computed by solving a system of linear equations, leading to our definition of the expected value under a fixed pure stationary strategy profile.

Definition 3.2.3 (Expected Value)**.** Let $G = (V, V_p, V_{\max}, V_{\min}, E, v_0, t)$ be a simple stochastic game. The *expected value* of a particular vertex $i \in V$ under the pure strategy profile (σ, τ) written $v_{\sigma,\tau}(i)$ is the probability of absorption at t in the resulting markov chain after fixing actions according to σ and τ. The *game expected value* is the value of the start vertex $v_{\sigma,\tau}(v_0)$.

The content of the following result is that simple stochastic games have a well defined notion of value that, importantly, can be achieved in *pure stationary strategies*. This is to say that both players can achieve optimal expected payoff in the game by fixing a deterministic action at every node in the game prior to it starting.

Theorem 3.2.4 ([5], lemma 6)**.** *Let $G = (V, V_p, V_{\max}, V_{\min}, E, v_0, t)$ be a β-stopping simple stochastic game and S, T the pure strategy stationary strategy sets for the maxmimizer and minimizer respectively. Then for each $i \in V$,*

$$\max_{\sigma \in S} \min_{\tau \in T} v_{\sigma,\tau}(i) = \min_{\tau \in T} \max_{\sigma \in S} v_{\sigma,\tau}(i).$$

Further if $q_i^* = \max_{\sigma \in S} \min_{\tau \in T} v_{\sigma,\tau}(i)$ then for each $i \in V$, $q_i^* \in \mathbb{Q}$.

The trick is then to construct a monotone function $F : [0,1]^d \to [0,1]^d$ such that $F(x) = x$ if and only if $x = (q_1, ..., q_d)$.

Definition 3.2.5 (β-stopping Simple Stochastic Game Monotone Function)**.** Let $G = (V, V_p, V_{\max}, V_{\min}, E, v_0, t)$ be a β-stopping simple stochastic game, $d = |V|$, and $(v_i)_{i\in[d]}$ some enumeration of the vertices in V. The β-*stopping simple stochastic game monotone function* is a function $F : [0,1]^d \to [0,1]^d$ defined coordinatewise as,

$$F((x_1, ..., x_d))_i = (1-\beta) \cdot \begin{cases} 1, & v_i = t \\ \max\{x_j : (v_i, v_j) \in E\}, & v_i \in V_{\max} \\ \min\{x_j : (v_i, v_j) \in E\}, & v_i \in V_{\min} \\ \sum_{v_j \in V} p(v_i, v_j) \cdot x_j, & v_i \in V_p. \end{cases}$$

It is shown in [5] that for all $x, x' \in [0,1]^d$, $\|F(x) - F(x')\|_\infty \leq (1-\beta)\|x - x'\|_\infty$, which is to say F is a contraction map and has a *unique* fixpoint by the banach fixpoint theorem. It is further shown in [5] that the unique fixpoint $x^* \in [0,1]^d$ satisfies $x^* = (q_1, ..., q_d)$. That is, x^* is the vector of values of the game. The reader can easily verify that F is also monotone in the usual coordinatewise ordering on $[0,1]^d$. The last stage in reduction to TARSKI is to discretize the function $F : [0,1]^d \to [0,1]^d$ to a function $f : [N]_0^d \to [N]_0^d$ such that a fixpoint of f can be converted to an approximate fixpoint of F. This is proved using ideas from [8] and [9] in the following.

Lemma 3.2.6 (Discretized β-stopping Simple Stochastic Game Monotone Function)**.** *Let $G = (V, V_p, V_{\max}, V_{\min}, E, v_0, t)$ be a β-stopping simple stochastic game, $F : [0,1]^d \to [0,1]^d$ it's corresponding monotone function, and $\varepsilon \in \mathbb{R}_{>0}$. Let $M = \left\lfloor \frac{1}{\beta\varepsilon} \right\rfloor$ and define $f : [M]_0^d \to [M]_0^d$ coordinatewise with $f(x)_i = \left\lfloor F(\beta\varepsilon \cdot x)_i \cdot \frac{1}{\beta\varepsilon} \right\rfloor$. If $x^* \in [0,1]^d$ is the unique fixpoint of F and $x = f(x)$ then $\|\beta\varepsilon \cdot x - x^*\|_\infty < \varepsilon$. Further, f is monotone.*

Proof. Let $x \in [M]_0^d$, suppose $x = f(x)$ and $F(x^*) = x^*$. From $x = f(x)$ I have by definition of $\lfloor \cdot \rfloor$ for each $i \in [d]$,

$$1 > F(\beta\varepsilon \cdot x)_i \cdot \frac{1}{\beta\varepsilon} - x_i \geq 0.$$

This implies by definition of $\|\cdot\|_\infty$ and homogoneity of the norm that,

$$\|\beta\varepsilon \cdot x - F(\beta\varepsilon \cdot x)\|_\infty < \beta\varepsilon.$$

So I calculate,

$$\begin{aligned} \|\beta\varepsilon \cdot x - x^*\|_\infty &\leq \|\beta\varepsilon \cdot x - F(\beta\varepsilon \cdot x)\|_\infty + \|F(\beta\varepsilon \cdot x) - x^*\|_\infty \\ &= \|\beta\varepsilon \cdot x - F(\beta\varepsilon \cdot x)\|_\infty + \|F(\beta\varepsilon \cdot x) - F(x^*)\|_\infty \\ &< \beta\varepsilon + (1-\beta)\|\beta\varepsilon \cdot x - x^*\|_\infty. \end{aligned}$$

Rearranging and dividing through by β gives the first part of the claim. That f is monotone easily follows from monotonicity of F, and that $\lfloor \cdot \rfloor$ and multiplication by non-negative constants preserve monotonicity. ■

The culmination of this section is the following result.

Theorem 3.2.7. [2] *Let G be a β-stopping simple stochastic game and $q^* \in \mathbb{Q}$ the value of the game. For all $\varepsilon \in \mathbb{R}_{>0}$ finding a $q \in \mathbb{Q}$ such that $|q - q^*| < \varepsilon$ is polynomial time reducible to* TARSKI *in the encoding size of G and ε.*

Remark 3.2.8. The algorithm of kleene-tarski algorithm 1 applied to the monotone function described above is analagous to a method often called *value iteration*, as described in [2]. There is also a method for solving simple stochastic games often called *strategy improvement* or *policy iteration* as described in [15] which, very broadly speaking, works by iteratively fixing a strategy for the maximizer (or minimizer), computing a best response for the minimizer in the residual markov decision process[3] yielding values $(q_1, ..., q_d)$. A new strategy for the first player chosen by myopically selecting at each node $v_i \in V_{\max}$ an edge $(v_i, v_j) \in E$ which maximizes q_j. The binary search style algorithms from chapter 4 will be compared to value iteration in chapter 5, and strategy improvement is not tested here.

3.3 Shapley's Stochastic Games

Shapley's stochastic games, or stopping stochastic games as described in [21] are a class of zero-sum game played on a set of states with two players called the maximizer and minimizer respectively. At each state the players concurrently choose an action, and then receive a payoff which sums to zero based on the joint action of the two players. The game then halts with some fixed probability. If it didn't halt then a next state is chosen randomly with probability distribution dependent on the joint action chosen. It is shown in [21] that stochastic games necessarily have an optimal expected value, and further that this value can be achieved in stationary strategies. Unlike simple stochastic games however, the value need not be rational or the strategies achieving the value pure. In [9] it is shown that the problem of finding a rational number ε close to the actual value of the game is polynomial-time reducible to TARSKI which will be described in this section.

Definition 3.3.1 (Stopping Stochastic Game)**.** A *stopping stochastic game* $G = (V, A, P, s, q)$ is a set of states of n states $V = (v_1, ..., v_n)$, for each state v_i an $n_i \times m_i$ rational valued matrix $A^i \in A$ called the *payoff matrix*, for each state v_i an $n_i \times m_i$ matrix $P^i \in P$ called the *transition matrix* who's j,k-th entry is an n-vector $P^i_{j,k} = \left((P^i_{j,k})_1, ..., (P^i_{j,k})_n\right)$ such that $\sum_{l \in [n]} (P^i_{j,k})_l = 1$ and for each $l \in [n]$, $(P^i_{j,k})_l \geq 0$. The *starting state* is a state $s \in V$ and the *stopping probability* is a positive rational number $q \in \mathbb{Q}_{>0}$. A *play* is as follows. A token is placed on the initial state $s = v_i$. The maximizer and minimizer choose actions $j \in [n_i]$ and $k \in [m_i]$ respectively, and receive payoffs $p_{\max} = A^i_{j,k}$ and $p_{\min} = -A^i_{j,k}$ respectively. The game then halts with probability q, and if it did not

[2] In [9] Etessami et al. show a stronger result; in a slightly more general model of simple stochastic games that don't necessarily have the β stopping property used above, the problem of finding the exact value of the game q^* is polynomial-time reducible to TARSKI. The simplified stopping game approximation result is shown above instead to simplify exposition, as well as to simplify implementation during the practical testing chapter of this dissertation.

[3] which can be done in polynomial time using linear programming

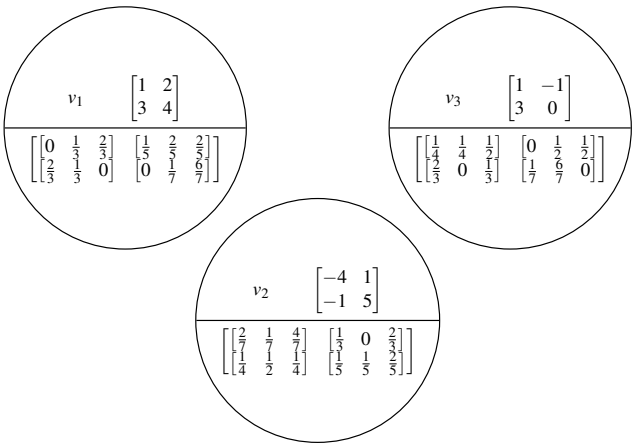

Figure 3.5: An instance of shapley's stochastic games is a set of states, and for each state a pair of actions sets for the two players. Play starts at some state and each player picks an action, and receives payoff equal to the entry in the payoff matrix (in the top half of each circle) corresponding to the joint action. Then play transitions transitions to a new state on a probability distribution according to the transition matrix (in the bottom of each circle). In the stopping variant, after each step the games halts with some positive probability $q \in \mathbb{R}_{>0}$.

transitions to state $l \in [n]$ with probability $(P^i_{j,k})_l$. Play continues until it halts (which happens with probability 1).

I require a more general notion of strategy to the pure strategies introduced in section 3.2 which are as follows.

Definition 3.3.2 (Mixed Stationary Strategy)**.** Let $G = (V,A,P,s,q)$ be a stopping stochastic game. A *mixed stationary strategy* for the maximizer is a vector of real vectors $\sigma = ((\sigma^1_1,...,\sigma^{n_1}_1),...,((\sigma^1_n,...,\sigma^{n_n}_n)))$ such that for each $i \in [n]$, $\sum_{j\in[n_i]} \sigma^j_i = 1$ and for each $j \in [n_i]$, $\sigma^j_i \geq 0$. The set of all mixed stationary strategies for the maximizer is denoted S. A mixed stationary strategy for the minimizer is $\tau = ((\tau^1_1,...,\tau^{m_1}_1),...,(\tau^1_n,...,\tau^{m_n}_n))$ such that for each $i \in [n]$, $\sum_{j\in[m_i]} \tau^j_i = 1$ and for each $j \in [m_i]$, $\tau^j_i \geq 0$. The set of all mixed stationary strategies for the minimizer is denoted T.

Once a stationary strategy profile is fixed, given a starting state, I can define a sequence of random variables $(X_t)_{t\in\mathbb{Z}_{\geq 0}}$ representing the payoff at the t-th step, allowing for the convention that the game continues to 'play' after halting, just giving both players payoff 0 repeatedly. The value of the game then becomes the (necessarily convergent) series $v = \sum_{t=0}^{\infty} \mathbb{E}[X_t]$[4]. Then the value of a state $x \in V$ under stationay strategy profile (σ,τ) is denoted $v_{\sigma,\tau}(x)$. The content of the following result from [21] is that stopping stochastic games necessarily have a minimax value, and that value can be achieved in stationary strategies.

Proposition 3.3.3 ([21])**.** *Let $G = (V,A,P,s,q)$ be a stopping stochastic game and S, T be the mixed stationary strategy sets for the maximizer and minimizer respectively.*

[4]That the expectation is well defined and the series convergent can be seen in [16, Chapter 2].

Then for all $x \in V$,

$$\max_{\sigma \in S} \min_{\tau \in T} v_{\sigma,\tau}(x) = \min_{\tau \in T} \max_{\sigma \in S} v_{\sigma,\tau}(x).$$

The value of the game is then denoted $v(s)$. The value of a particular state $x \in V$ is denoted $v(x)$ and is the maximum expected value if the game were to start at x. Recall the definition of val from definition 2.2.4.

Definition 3.3.4 (Stopping Stochastic Game Monotone Function)**.** Let $G = (V, A, P, s, q)$ be a stopping stochastic game, $d = |V|$ and $M = \max_{i,j,k} |A^i_{j,k}|$. Then the *stopping stochastic game monotone function* is a function $F : [-\frac{M}{q}, \frac{M}{q}]^d \to [-\frac{M}{q}, \frac{M}{q}]^d$ defined coordinatewise as $F(x)_i = \text{val}\left(A^i + (1-q) \cdot T^i\right)$ where T^i is an $n_i \times m_i$ matrix defined $T^i_{j,k} = \sum_{l=1}^{n} (P^i_{j,k})_l \cdot x_l$.

I was perhaps hasty in writing the codomain of this function, but the following lemma shows it to be correct.

Lemma 3.3.5. *Let $G = (V, A, P, s, q)$ be a stopping stochastic game and F as defined in definition 3.3.4. Then for all $x \in [-\frac{M}{q}, \frac{M}{q}]^d$, $F(x) \in [-\frac{M}{q}, \frac{M}{q}]^d$. Further, F is monotone.*

Proof. Note $\text{val}(\cdot)$ is monotone in the coordinatewise ordering for the corresponding set of matrices. For if $\text{val}(B) = k$ for some $B \in \mathbf{Mat}(\mathbb{Q}, n \times m)$ then there is some distribution $x \in \mathbb{Q}^n$ such that for all distributions $y \in \mathbb{Q}^m$, $x^T B y \geq k$. Then if $B \leq B' \in \mathbf{Mat}(\mathbb{Q}, n \times m)$ then since x and y necessarily have all non-negative entries, $k \leq x^T B y \leq x^T B' y$. So $\text{val}(B) \leq \text{val}(B')$. It is then easily verified that F is monotone. Now if $N = \max_{j,k} |B_{j,k}|$ then clearly $-N \leq \text{val}(B) \leq N$. By monotonicity, for all $x \in [-\frac{M}{q}, \frac{M}{q}]^d$, $F(x) \leq F\left(\frac{\vec{M}}{q}\right)$. I denote $[\![k]\!]$ to be the matrix with entries all equal to k with size inferred from context. For each $i \in [d]$,

$$\begin{aligned} F\left(\frac{\vec{M}}{q}\right)_i &= \text{val}(A^i + (1-q) \cdot T^i) \\ &\leq \text{val}\left([\![M]\!] + (1-q)\left[\!\left[\frac{M}{q}\right]\!\right]\right) \\ &= \text{val}\left([\![M]\!] - [\![M]\!] + \left[\!\left[\frac{M}{q}\right]\!\right]\right) = \frac{M}{q}. \end{aligned}$$

Noting that in the case $x = \frac{\vec{M}}{q}$, T^i can clearly be seen to be equal to $\left[\!\left[\frac{M}{q}\right]\!\right]$ for each $i \in [d]$. It can be shown using a similar argument that $F(x) \geq -\frac{M}{q}$ by observing again by monotonicity that $F(x) \geq F\left(-\frac{M}{q}\right)$. ∎

Shapley shows in [21] that for all $x, x' \in [-\frac{M}{q}, \frac{M}{q}]^d$ that $\|F(x) - F(x)'\|_\infty \leq (1-q)\|x - x'\|_\infty$, which is to say F is a contraction map and has a *unique* fixpoint by the banach fixpoint theorem. It is further shown in [21] the unique fixpoint gives precisely the value of the game. That is, if $x^* = F(x^*)$ then for all $i \in [d]$, $v(v_i) = x^*_i$. The last step is a discretization to allow application of the discrete algorithms. I use the notation $\langle M \rangle = \{-M, ..., 0, ..., M\}$.

Lemma 3.3.6 (Stopping Stochastic Game Discretization)**.** *Let $G = (V, A, P, s, q)$ be a stopping stochastic game, F as in definition 3.3.4, and $\varepsilon \in \mathbb{R}_{>0}$. Then if M is the maximum entry in the set of payoff matrices, let $N = \frac{M}{\varepsilon q^2}$ and define a function $f : \langle N \rangle \to \langle N \rangle$ coordinatewise with $f(x)_i = \left\lfloor \frac{1}{\varepsilon q} F(\varepsilon q \cdot x)_i \right\rfloor$. If $f(x) = x$ and x^* is the unique fixpoint of F then $\|\varepsilon q \cdot x - x^*\|_\infty < \varepsilon$.*

Proof. From $f(x) = x$ I have by definition of $\lfloor \cdot \rfloor$ for each $i \in [d]$, $1 > \frac{1}{\varepsilon q} \cdot F(\varepsilon q \cdot x)_i - x_i \geq 0$, which is to say $\|\frac{1}{\varepsilon q} \cdot F(\varepsilon q \cdot x) - x\|_\infty < 1$. Then by homogeneity of the norm, $\|F(\varepsilon q \cdot x) - x\|_\infty < \varepsilon q$. I calculate,

$$\begin{aligned} \|\varepsilon q \cdot x - x^*\|_\infty &\leq \|\varepsilon q \cdot x - F(\varepsilon q \cdot x)\|_\infty + \|F(\varepsilon q \cdot x) - x^*\|_\infty \\ &= \|\varepsilon q x - F(\varepsilon q \cdot x)\|_\infty + \|F(\varepsilon q \cdot x) - F(x^*)\|_\infty \\ &< \varepsilon q + (1-q)\|\varepsilon q \cdot x - x^*\|_\infty. \end{aligned}$$

Rearranging and dividing through by q gives the claim. ■

It should be noted that in [9, Proposition 6.2.] a similar result to lemma 3.3.6 is proven. There is however a mistake at the end of the proof in defining the discretized function, which the above shows how to rectify. I leave out details on encoding sizes, but these can be found in [9]. Putting it all together,

Theorem 3.3.7 ([9])**.** *Let G be a stopping stochastic game and $v \in \mathbb{R}$ the value of the game. Then for all $\varepsilon \in \mathbb{Q}_{>0}$ computing an $x \in \mathbb{Q}$ such that $|x - v| < \varepsilon$ is polynomial-time reducible to* TARSKI *in the encoding size of ε and G.*

3.4 Open Problems

The key open problems relating to this chapter are whether or not polynomial-time algorithms exist for any of the described problems.

Open problem 3.4.1. *Is* ARRIVAL *in* **P***?*

Open problem 3.4.2. *Is computing the exact value of a simple stochastic game in* **P***?*

Open problem 3.4.3. *Is approximating the value of shapley's stochastic games to any $\varepsilon \in \mathbb{R}_{>0}$ in* **P***?*

All of these problems have seen a significant amount of study in years gone by. That answers have not been found, and they are all reducible to TARSKI places TARSKI in an interesting position in the complexity landscape of algorithmic game theory, and perhaps motivates further study on the problem.

Chapter 4

State of the art Algorithms

4.1 Overview

Recent developments have been made in upper bounds for the TARSKI problem. The critical component to the algorithmic improvements is the surprising result from [10] that TARSKI$(N,3)$ can be solved in at most $O(\log^2 N)$ queries. The idea is roughly as follows; suppose I have a monotone function $f : [N]^3 \to [N]^3$ and an algorithm which given a coordinate $i \in \{1,2,3\}$ and a value $v \in [N]$ returns a monotone point $x \in [N]^3$ with the guarantee that $x_i = v$. If this algorithm takes $q(N)$ queries in the worst case, then a fixpoint of f can be found in $O(\log N \cdot q(N))$ in the same fashion as algorithm 2.

Definition 4.1.1 (FINDMONOTONE)**.** The problem FINDMONOTONE(N,d) is, given oracle access to a monotone function $f : [N]^d \to [N]^d$, a coordinate $i \in \{1,...,d\}$, and value $v \in [N]$, find a monotone point $x \in [N]^d$ such that $x_i = v$.

The breakthrough in [10] was in detailing a so-called inner-algorithm which leads to the following result.

Theorem 4.1.2 ([10])**.** *The query complexity of* FINDMONOTONE$(N,3)$ *is* $O(\log N)$.

Fearnley et al. also gave a method of decomposition, allowing for TARSKI in higher dimensions to be decomposed into a product of sorts of lower dimensional problems which I call *fixpoint decomposition* and will be detailed in section 4.3.

Theorem 4.1.3 (Fixpoint Decomposition, [10])**.** *For positive integers $a,b \in \mathbb{Z}_{>0}$, given an algorithm A which can solve* TARSKI(N,a) *in $p(N,a)$ queries, and an algorithm B which can solve* TARSKI(N,b) *in $p(N,b)$ queries, the problem* TARSKI$(N,a+b)$ *can be solved in $O(p(N,a) \cdot q(N,b))$ queries.*

Chen and Li take this one step further; instead of decomposing the TARSKI problem they show that it is possible to make a decomposition on FINDMONOTONE using a method I will call *monotone decomposition* which will be detailed in section 4.4¿

Theorem 4.1.4 (Monotone Decomposition, [3])**.** *For positive integers $a,b \in \mathbb{Z}_{>0}$, given an algorithm A which can solve* FINDMONOTONE(N,a) *in $p(N,a)$ queries, and an*

algorithm B which an solve FINDMONOTONE(N,b) *in* $q(N,b)$ *queries, the problem* FINDMONOTONE$(N,a+b)$ *can be solved in* $O((b+1)\cdot p(N,a)\cdot q(N,b))$ *queries.*

The purpose of this chapter is to describe the state of the art algorithms so that the reader can get a sense of possible directions for future study on the problem. In particular, I believe it will be worthwhile from a theoretical perspective to work on generalizing the inner algorithm from section 4.2 to higher dimensions.

4.2 The Inner Algorithm

In the interest of saving space, I will not detail the inner algorithm in it's entirety. Instead I will introduce the main invariant, proving lemmas showing how the inner algorithm can make progress in the main cases.[1] In contrast to algorithm 2 where two opposing monotone points are maintained essentially defining the top and bottom of a sublattice on which the monotone function naturally restricts (and hence contains a fixpoint), the inner algorithm maintains an invariant so that only a monotone point is guaranteed within the current search space.

Definition 4.2.1 (Witness, [10])**.** Let $f : [N]^3 \to [N]^3$. A *down set witness* is a pair of points $(d,b) \in ([N]^3)^2$ such that $d_3 = b_3$ and for some $i, j \in \{1,2\}$ with $i \neq j$,

- $d_i = b_i$, $b_j \leq d_j$, $f(b)_j \leq b_j$, if $b \neq d$ then $f(d)_j \geq d_j$, else $f(b)_i \leq b_i$,
- $f(d)_3 \geq d_3$ and $f(b)_3 \geq b_3$.

An *up set witness* is a pair of points $(a,u) \in ([N]^3)^2$ such that $a_3 = u_3$ and for some $i, j \in \{1,2\}$ with $i \neq j$,

- $a_i = u_i$, $a_j \leq u_j$, $f(a)_j \geq a_j$, if $u \neq a$ then $f(u)_j \leq u_j$, else $f(a)_i \geq a_i$,
- $f(d)_3 \geq d_3$ and $f(b)_3 \geq b_3$.

I use a notational convenience from [10].

Definition 4.2.2 (Up/Down Set)**.** Let $f : L \to L$ be a monotone function. Then the *up-set* of f is defined $\mathrm{Up}(f) = \{x \in L : x \leq f(x)\}$ and the *down-set* of f is defined $\mathrm{Down}(f) = \{x \in L : x \geq f(x)\}$. At times an abuse of notation is used, where elements of the lattice $x \in [N]^d$ are said to be in the up set or down set of the slice f_s of a monotone function $f : [N]^d \to [N]^d$. The meaning is assumed to be clear from context.

Now for the main invariant of the inner algorithm.

Definition 4.2.3 (Inner algorithm invariant)**.** Let $f : [N]^3 \to [N]^3$ be a monotone function. The *inner algorithm invariant* is an up set witness (d,b) and a down set witness (a,u) such that $u \leq d$.

The reader is directed to fig. 4.1 to aid in digesting the witness and invariant definitions. Throughout whenever the inner algorithm invariant is satisfied, I fix f_s be the slice of f

[1] The key missing cases are when the current search space is 'narrow', in that the width in some dimension is ≤ 1. The reader is directed to [10] for these.

at coordinate 3 with value $b_3 = a_3 = d_3 = u_3$ (where all of these equalities follow from the witness definitions).

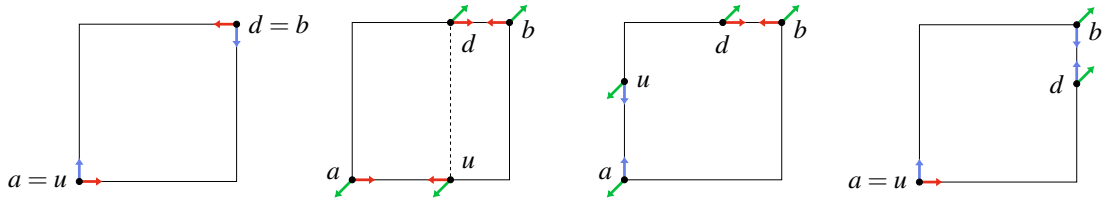

Figure 4.1: There are many possible configurations of witnesses that satisfy the invariant. All of the above are representations of witness pairs which satisfy the invariant using the useful diagramming notation from [10] that should be read as follows; the outer square represents some 'slice' of the three dimensional cube in the third coordinate. An arrow extending from a point $x \in [N]^3$ in direction $i \in \{1,2,3\}$ in the positive (negative) direction should be read as $f(x)_i \geq (\leq) x_i$. For example in the rightmost square if 1 is the horizontal direction and 2 is the vertical direction, and 3 is forwards dimension then $f(b)_2 \leq b_2$ and $f(b)_3 \geq b_3$. Diagram source: [10]

Lemma 4.2.4. *Let $f : [N]^3 \to [N]^3$ be a monotone function with (a,u), (d,b) up and down set witnesses respectively satisfying the inner algorithm invariant. Then there is a point $x \in [N]^3$ with $a \leq x \leq b$ and $x \in \mathrm{Up}(f)$ or $x \in \mathrm{Down}(f)$.*

Proof. If $a = u$ and $b = d$ then by the witness definition $a \in \mathrm{Up}(f_s)$ and $b \in \mathrm{Down}(f_s)$, so lemma 2.3.10 guarantees fixpoint of f_s in the range $[a,b]$, which is then necessarily a monotone point of f.
By the witnesses definition, there are $j, j' \in \{1,2\}$ such that $d_j = b_j$ and $a_{j'} = u_{j'}$. Then let $i, i' \in \{1,2\}$ with $i \neq j$ and $i' \neq j'$. If $a \neq u$ then lemma 2.3.10 gives a point $c \in [N]^3$ with $a \leq c \leq u$ and $f(c)_{i'} = c_{i'}$. By monotonicity and $c \leq u$ I have $f(c)_3 \leq c_3$. If $f(c)_1 \leq c_{j'}$ then $c \in \mathrm{Down}(f)$ and I'm done. If $f(c)_{j'} \geq c_{j'}$ then I consider b and d. Using a similar argument there is a $v \in [N]^3$ with $d \leq v \leq b$ such that $f(v)_3 \geq v_3$, and $f(v)_i = v_i$. If $f(v)_j \geq v_j$ then $v \in \mathrm{Up}(f)$, and if $f(v)_j \leq v_j$ then by the invariant, $v \geq d \geq u \geq c$ gives $v \geq c$, and lemma 2.3.10 guarantees a fixpoint of f_s in the range $[c,v]$, which is neccessarily a monotone point of f. ■

The key is then to half the current search space in a constant amount of time by finding a new set of points satisfying the inner algorithm invariant. There are many distinct cases to be handled here and I have not covered them all. In particular, special cases are required for instances which are narrow (that is, for some $i \in \{1,2\}$ $b_i - a_i \leq 1$). The reader is directed to [10] for these. The first case is under the assumption that u and d are not past the midpoint of the line they are on. Throughout I set $\mathrm{mid}_i = \left\lfloor \frac{a_i + b_i}{2} \right\rfloor$ for each $i \in \{1,2,3\}$, and take f_s to be the slice of f at the 3rd coordinate with value $a_3 = b_3$.

Lemma 4.2.5 ([10])**.** *Let $f : [N]^3 \to [N]^3$ and suppose for each $i \in \{1,2\}$ that $u_i \leq \left\lfloor \frac{a_i+b_i}{2} \right\rfloor$ and $d_i \geq \left\lfloor \frac{a_i+b_i}{2} \right\rfloor$. Then either a pair of witnesses $(\overline{a},\overline{u})$, $(\overline{d},\overline{b})$ satisfying the invariant such that for some $j \in \{1,2\}$ $\overline{b}_j - \overline{a}_j \leq \left\lceil \frac{a_j - b_j}{2} \right\rceil$, or a point $x \in \mathrm{Up}(f) \cup \mathrm{Down}(f)$ can be found using a constant number of queries.*

Proof. Evaluate $f(\text{mid})$. If $\text{mid} \in \text{Up}(f)$ or $\text{mid} \in \text{Down}(f)$ then the inner algorithm can return immediately. If $\text{mid} \in \text{Up}(f_s)$ and $\text{mid} \notin \text{Up}(f)$ then $f(\text{mid})_3 \leq \text{mid}$ and I can set $\overline{a} = \overline{u} = \text{mid}$. By assumption $u \geq \text{mid} \geq d$ so the pair $(\overline{a}, \overline{u})$, (d, b) will do. The case $\text{mid} \in \text{Down}(f_s)$ is similar. Suppose for some $i, j \in \{1,2\}$ with $i \neq j$ that $f(\text{mid})_i \leq \text{mid}_i$ and $f(\text{mid})_j \geq \text{mid}_j$. If $f(\text{mid})_3 \geq \text{mid}_3$ then put $p_j = b_j$, $p_i = \text{mid}_i$ and $p_3 = \text{mid}_3$ and evaluate $f(p)$. By monotonicity $f(p)_3 \geq p_3$, so if $f(p)_j \leq p_j$ then put $\overline{u} = \text{mid}$ and $\overline{a} = p$ and I'm done. If $f(p)_j > p_j$ then by monotonicity and $b_j = p_j$ I have $f(b)_j > b_j$. It follows by definition of witnesses that if $b \neq d$ then $b_i \neq d_i$ and $b_j = d_j$. Then again by monotonicity and $d \leq p$ I have $d_j \leq f(d)_j$, and by definition of a witness $d_i \leq f(d)_i$ and $d_3 \leq f(d)_3$, so $d \in \text{Up}(f)$ and can be returned immediately. The case $f(\text{mid})_3 \leq \text{mid}_3$ is similar except I consider $p_i = a_i$, $p_j = \text{mid}_j$, $p_3 = \text{mid}_3$ and find either (p, mid) is an up set witness, or $u \in \text{Down}(f)$. ■

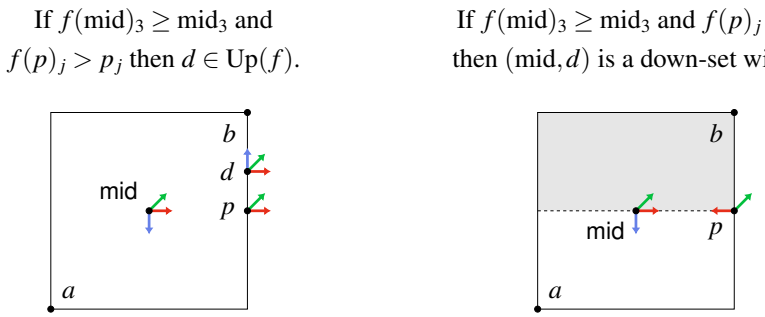

Figure 4.2: In the main case where $u \leq \text{mid}$, $d \geq \text{mid}$, and $a_i - b_i \geq 2$ for $i \in \{1,2\}$, the search space can be halved as in the proof of lemma 4.2.5. mid is queried, and the only non-trivial case is when $\text{mid} \notin \text{Up}(f_s) \cup \text{Down}(f_s)$. The diagrams show the case when $f(\text{mid})_3 \geq \text{mid}_3$. The case $f(\text{mid})_3 \leq \text{mid}_3$ then is similar. The horizontal dimension is taken to be dimension j from the proof, and vertical is i. Diagram source: [10]

.

I must now justify the assumption in lemma 4.2.5 that $u \leq \text{mid}$ and $d \geq \text{mid}$.

Lemma 4.2.6 ([10])**.** *Suppose for some $i \in \{1,2\}$ that $u_i \not\leq \text{mid}_i$. Then a point $p \in [N]^3$ can be found such that $p_i \leq \text{mid}_i$ and (a, p) is a valid down-set witness, or (p, u) is a valid down-set witness. If $d_i \not\geq \text{mid}$ then a point $q \in [N]^3$ can be found such that $q_i \geq \text{mid}$ and (q, b) is a valid up-set witness, or (d, q) is a valid up-set witness.*

Proof. Suppose for some $i \in \{1,2\}$ that $u_i > \text{mid}_i$. Then by definition of a witness, if $j \in \{1,2\}$ and $i \neq j$ then $u_j = a_j$. Put $p_j = u_j$, $p_i = \text{mid}_i$. Then since $u \geq p$ by monotonicity and definition of a witness I have $f(u)_3 \geq u_3$. If $f(p)_i \leq p_i$ then (a, p) is a valid down set witness that satisfies $p_i \leq \text{mid}_i$. If $f(p)_i \geq p_i$ then (p, u) is a valid down-set witness.
The case with $d_i < \text{mid}_i$ is similar; I take $p_j = d_j$, $p_i = \text{mid}_i$ and $p_3 = \text{mid}_3$ and find either (p, b) is a down-set witness or (b, p) is a down-set witness. ■

Combining the previous two lemmas, a 'normal' iteration of the inner algorithm is as follows. Check if $u < \text{mid}$ or $d > \text{mid}$. If so, using lemma 4.2.6 either find a new set of witnesses with a search space that is half the size of the previous, or a new of witnesses such that $u \leq \text{mid}$ and $d \geq \text{mid}$ and continue to the next iteration. If $u \geq \text{mid}$ and $d \leq \text{mid}$ then using lemma 4.2.5 find a new set of witnesses such that the new

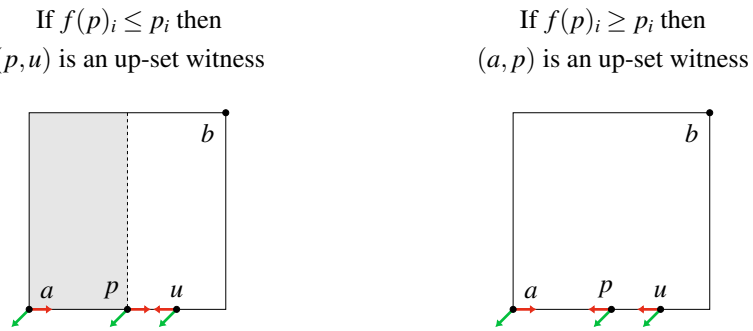

Figure 4.3: The main case described in lemma 4.2.5 breaks down if $u \not\leq \text{mid}$ or $d \not\geq \text{mid}$. This can be rectified via lemma 4.2.6. The horizontal dimension is taken to be i from the proof of lemma 4.2.6, and vertical j. Diagram source: [10]

search space is at most half the size of the previous. With the additional fact from [10] that the search space can be halfed in a constant amount of queries when the search space as a width of at most 1, this gives an $O(\log N)$ query algorithm for the FINDMONOTONE$(N,3)$ problem.

4.3 Fixpoint Decomposition

Throughout I will fix $a,b \in \mathbb{Z}_{>0}$, $d = a+b$, and $f : [N]^a \times [N]^b \to [N]^a \times [N]^b$. Suppose I have an algorithm A which can solve TARSKI(N,a), and an algorithm B which can solve TARSKI(N,b). A naive approach for finding a fixpoint of f would be to define a function on the right hand side of the lattice $f_r : [N]^b \to [N]^b$ where given $x_r \in [N]^b$ the value of $f_r(x_r)$ is computed by defining the slice $f_l : [N]^a \to [N]^a$ such that $f_l(x_l) = f((x_l,x_r))_{-r}$, finding a fixpoint $x_l^* \in [N]^a$ of f_l, then using $f(x_l^*,x_r)_{-l}$ as the result of $f_r(x_r)$. The punchline is that if x_r is a fixpoint of f_r, and x_l^* was the fixpoint of f_l associated with x_r then the point (x_l^*,x_r) is a fixpoint of f. This does not work however - there is no guarantee that f_r is monotone. That is, if points $x_r, x_r' \in [N]^b$ are queried by algorithm B with $x_r \leq x_r'$ it is not necessarily the case that the associated fixpoints of f_l x_l^* and $x_l'^* \in [N]^a$ satisfy $x_l^* \leq x_l'^*$, so monotonocity of f does not carry over. The trick will thus be to find a way to guarantee that x_l^* and $x_l'^*$ satisfy $x_l^* \leq x_l'^*$ whenever $x_r \leq x_r'$. Fortunately this is achieveable; in algorithm 3 the issue is solved by carefully choosing bounds of the sublattice to search in based on previously computed points. The correctness of which will be the concern of the remainder of this section. An abuse of notation x_{-r} or x_{-l} is often used to denote the first a coordinates, or last b coordinates of x respectively.

Lemma 4.3.1. *If* $(p_l,p_r),(p_l',p_r') \in \text{prev}$ *with* $p_r \leq p_r'$ *then* $p_l \leq p_l'$

Proof. Suppose without loss of generality that p_r was queried by algorithm B before p_r'. Then when p_r' is queried, p_l is an element of $\{pl : (p_l,p_r) \in \text{prev} : p_r \leq p_r'\}$ whence $\bot_l \geq p_l$, so the fixpoint found by algorithm A satisfies $x_l^* \geq p_l$. ■

Lemma 4.3.2. *At line 7 of algorithm 3* $\bot_l \leq \top_l$.

Algorithm 3 [10]. An algorithm for decomposing fixpoint computation problems.

1: **procedure** FIXPOINTDECOMPOSITION(
monotone $f : [N]^a \times [N]^b \to [N]^a \times [N]^b$,
algorithm A for solving TARSKI(N,a),
algorithm B for solving TARSKI(N,b))
2: prev $\leftarrow \emptyset$
3: **procedure** $f_r(x_r \in [N]^b)$
4: **procedure** $f_l(x_l \in [N]^a)$ **return** $f((x_l,x_r))_{-r}$.
5: $\bot_l \leftarrow \bigvee\{p_l : (p_l,p_r) \in \text{prev},\ p_r \leq x_r\}$
6: $\top_l \leftarrow \bigwedge\{p_l : (p_l,p_r) \in \text{prev},\ p_r \geq x_r\}$
7: $x_l^* \leftarrow A(f_l)$ with bounds $\bot_l$, $\top_l$
8: prev $\leftarrow$ prev $\cup \{(x_l^*,x_r)\}$
9: **return** $f((x_l^*,x_r))_{-l}$.
10: $x_r^* \leftarrow B(f_r)$
11: **return** (x_l^*,x_r^*) where x_l^* is the fixpoint of f_l found when evaluating $f_r(x_r^*)$.

Proof. By lemma 4.3.1 I have for all $p_l \in \{p_l : (p_l,p_r) \in \text{prev} : p_r \leq x_r\}$ and $p_l' \in \{p_l : (p_l,p_r) \in \text{prev} : p_r \geq x_r\}$ that $p_l \leq p_l'$. Then by definition of $\wedge$ and $\vee$ $\bot_l \leq \top_l$. ■

Lemma 4.3.3. *At line 7 of algorithm 3 I have $f_l(\bot_l) \geq \bot_l$ and $f_l(\top_l) \leq \top_l$.*

Proof. I prove the first case and the second is similar. Suppose not. That is, for some $i \in [a]$ I have $f_l(\bot_l)_i < (\bot_l)_i$. If $L = \{p_l : (p_l,p_r) \in \text{prev} : p_r \leq x_r\}$ is empty then $\vee L = \vec{1}$ and I contradict the definition of f. If L is non-empty then by definition of finite joins, there is some $(p_l,p_r) \in \text{prev}$ such that $(p_l)_i = (\bot_l)_i$ and $(p_l,p_r) \leq (x_l,x_r)$. But $f(((p_l,p_r))_{-r})_i = (p_l)_i$ so I contradict monotinicity. ■

Lemma 4.3.4. *At line 7 of algorithm 3 f_l is a monotone function on the lattice $[\bot_l,\top_l]$.*

Proof. That f_l is monotone follows from an inductive application of lemma 2.3.9. Then the rest follows from lemma 4.3.3 and lemma 2.3.10. ■

Proposition 4.3.5. *The point (x_l^*,x_r^*) returned by algorithm 3 is a fixpoint of f.*

Proof. lemma 4.3.1 guarantees that f_r is monotone, and hence on line 10 a fixpoint x_r^* can be found. Then by construction (x_l^*,x_r^*) is clearly a fixpoint of f. ■

Proof of theorem 4.1.3. Suppose algorithm A takes at most $p(N,a)$ queries and algorithm B takes at most $q(N,a)$ queries to find a fixpoint. Then every query of f_r by algorithm A makes at most $q(N,b)$ queries to f to find a fixpoint of f_l, so given algorithm A makes a most $p(N,a)$ queries, the entire algorithms makes at most $p(N,a) \cdot q(N,b)$ queries to f. ■

4.4 Monotone Decomposition

The algorithm described in section 4.3 makes a decomposition of TARSKI by decomposing into smaller fixpoint computation problems, with the asymptotic improvement coming from the fact that there is an algorithm to efficently solve the TARSKI$(N,3)$ problem. I can do better however; Chen and Li describe in [3] a method of decomposing the FINDMONOTONE problem instead. To save space, this section will show the algorithms used but for proofs of correcteness the reader is referred to [3]. It will be simpler to give an overview after a definition of an auxiliary problem.

Definition 4.4.1 (TARKSI*, [3])**.** The problem TARKSI$^*(N,d)$ is given oracle access to a function $f : [N]^d \to \{-1,0,1\}^d$ such that,

1. for each $x \in [N]^d$ and $i \in [d]$, $x_i + f(x)_i \in [N]^d$,
2. for each $x,y \in [N]^d$ with $x \leq y$, $(x,0) + f(x) \leq (y,0) + f(y)$,

find a point $x \in [N]^d$ such that $f(x) \geq 0$ or $f(x) \leq 0$.

This function f is designed to be an indicator of a monotone function $F : [N]^d \to [N]^d$; that is if I define f coordinatewise as $f(x)_i = \begin{cases} 1, & F(x)_i > x_i \\ 0, & F(x)_i = x_i \\ -1, & F(x)_i < x_i \end{cases}$ then f satisfies the conditions in definition 4.4.1. The first condition is the requirement that the codomain of F is correct, and the second is monotonicity. It is also clear that FINDMONOTONE(N,d) trivially reduces to TARKSI$^*(N,d)$. Chen and Li then wish to make decompositions from TARKSI$^*(N,a+b)$ into TARKSI$^*(N,a)$ and TARKSI$^*(N,b)$. The fixpoint decomposition method algorithm 3 does not obviously apply here; it is not clear what to do with the extra coordinate in the codomain, and further if I split into f_l and f_r similarly to algorithm 3, if $(x_l,x_r) \in [N]^a \times [N]^b$, are found as monotone points of f_l and f_r, x_l can be a monotone-down point, and x_r a monotone up point so (x_l,x_r) is not necessarily a monotone point of f. Chen and Li's first step towards a solution is defining a refinement of the TARKSI* problem.

Definition 4.4.2 (REFINEDTARSKI*, [3])**.** Given a function $f : [N]^d \to \{-1,0,1\}^{d+1}$ as in definition 4.4.1, find a pair of points $\bot, \top \in [N]^d$ such that $\bot \leq \top$, for each $i \in [d]$, $f(\bot)_i \geq 0$, $f(\top)_i \leq 0$, and at least one of the following hold,

1. $f(\bot)_{d+1} = 1$,
2. $f(\top)_{d+1} = -1$,
3. $f(\bot) = f(\top) = 0$.

Interestingly, Chen and Li show that this problem is no harder than TARKSI* in the sense that it can be solved in at most two queries to a TARKSI* orcacle. Their algorithm for doing this is shown in algorithm 4.

Algorithm 4 [3]. An auxiliary algorithm for monotone decomposition.

procedure REFINEDTARSKI*(
$f : [N]^d \to \{-1,0,1\}^{d+1}$ as in definition 4.4.1,
algorithm A for solving TARKSI$^*(N,d)$)
  $(\bot, \top) \leftarrow (\vec{1}, \vec{N})$
  Let $g^+ : [N]^d \to [N]^{d+1}$ with $g^+(x)_i = \begin{cases} 1, & i = d+1 \text{ and } g(x)_{d+1} \geq 0 \\ g(x)_i, & \text{otherwise} \end{cases}$
  $x \leftarrow A(g^+)$ with bounds $\bot$, $\top$

  if $g^+(x) = -1$ **then** $\top \leftarrow x$ and **return** $(\bot, \top)$
  $\bot \leftarrow x$
  Let $g^- : [N]^d \to [N]^{d+1}$ with $g^+(x)_i = \begin{cases} -1, & i = d+1 \text{ and } g(x)_{d+1} \leq 0 \\ g(x)_i, & \text{otherwise} \end{cases}$
  $y \leftarrow A(g^-)$ with bounds $\bot$, $\top$
  if $g^-(y)_{d+1} = 1$ **then** $\bot \leftarrow y$ **else** $\top \leftarrow y$
  return $(\bot, \top)$

Correctness is relatively simple to verify and is omitted for brevity. Note g^+ and g^- also need to be verified to satisfy definition 4.4.1. Now for the main algorithm in algorithm 5.

Algorithm 5 [3]. An algorithm for decomposing monotone point computation problems.

procedure MONOTONEDECOMPOSITION(
$f : [N]^{a+b} \to \{-1,0,1\}^{a+b+1}$ as in definition 4.4.1,
algorithm A for solving TARSKI(N,a),
algorithm B for solving TARSKI(N,b))
  prev $\leftarrow \emptyset$
  procedure $f_r : [N]^b \to \{-1,0,1\}^{b+1}(x_r \in [N]^b)$
    $\bot_l \leftarrow \bigvee\{p_l : (p_l, p_r) \in \text{prev},\ p_r \leq x_r\}$
    $\top_l \leftarrow \bigwedge\{p_l : (p_l, p_r) \in \text{prev},\ p_r \geq x_r\}$
    for j from $a+1$ to $a+b+1$ **do**
      procedure $f_l^j : [N]^a \to \{-1,0,1\}^{a+1}(x_l \in [N]^a)$
        $y \leftarrow f((x_l, x_r))$
        return $(y_1, ..., y_a, y_j)$.
      $(\bot_l^j, \top_l^j) \leftarrow$ REFINEDTARSKI$^*(f_l^j, A)$ with bounds $\bot_l$, $\top_l$
      $(\bot_l, \top_l) \leftarrow (\bot_l^j, \top_l^j)$
    prev $\leftarrow$ prev $\cup \{(\bot_l, x_r)\}$
    $x \leftarrow f((\bot_l, x_r))$
    return $f(x_{a+1}, ..., x_{b+1})$.
  $x_r^* \leftarrow B(f_r)$
  let $(\bot_l, \top_l)$ be the bounds found when evaluating $f_r(x_r^*)$
  if $f_r(x_r^*) \geq 0$ **then return** $(\bot_l, x_r^*)$ **else return** $(\top_l, x_r^*)$

Lemma 4.4.3. *algorithm 5 gives a correct solution to* TARKSI*.

Proof sketch. The proof that all intermediate functions restrict to the transient bounds $(\bot_l, \top_l)$ is similar to the proof of correctness of algorithm 3 with a few extra cases. The crux of the proof for algorithm 5 is [3, Lemma 6] where it is shown that at line 16, for each $i \in \{a+1, ..., b+1\}$, and every $p, p' \in [\bot_l, \top_l]$, $f((p, x_r^*))_i = f((p', x_r^*))_i$. This is because inductively from the loop on line 6 I find $\bot_l \geq \bot_l^i$ and $\top_l \leq \top_l^i$, so if $f((\bot_l, x_r^*))_i = 1$, then since $(\bot_l^i, \top_l^i)$ was a solution to REFINEDTARSKI$^*(f_l^i, A)$ I find $f((p, x_r^*)) \geq f((\bot_l, x_r^*))_i \geq f_l^i(\bot_l^i)_i = 1$. The cases with $f((\bot_l, x_r^*))_i \in \{-1, 0\}$ are similar.
When the algorithm returns on line 17, $(\bot_l, \top_l)$ was a solution to REFINEDTARSKI* in the slice at x_r^*, so for each $i \in [a]$, $f((\bot_l, x_r^*))_i \geq 0$ and $f((\top_l, x_r^*))_i \leq 0$, so the condition on line 17 ensures that the returned point is a solution to TARKSI*. ■

For complete proof of correctness of algorithm 5 see [3]. It is clear that algorithm 5 gives the asymptotic bound in theorem 4.1.4.

4.5 Open Problems

Many problems on the algorithmic complexity of TARSKI remain open. The most impressive upper bound on query complexity would be the following.

Open problem 4.5.1. *Is the query complexity of* TARSKI(N, d) $O(\text{poly}(\log N) \cdot \text{poly}(d))$*?*

The implications of a result in the positive for open problem 4.5.1 would be significant. It would imply a polynomial-time algorithm for computing the exact value of simple stochastic games as described in section 3.2, which has remained an open problem since 1992 when studied by Condon in [5]. It would also imply a polynomial-time algorithm for the ARRIVAL problem as described in section 3.1, the complexity of which has seen a significant amount of study in recent years [14, 13, 7, 18]. It therefore seems reasonable to consider weaker conjectures.

Open problem 4.5.2. *Is the query complexity of* TARSKI(N, d) $O(\log^2 N)$ *for fixed d? That is, is* TARSKI *fixed-parameter tractable?*

Recent results such as theorem 4.1.2 from [10] make this seem plausible. Perhaps the notion of a 'witness' as in definition 4.2.1 can be generalized to higher dimensions, and the search-space halved as in lemma 4.2.5 in a potentially exponential number of queries? A useful intermediate result towards this problem could be the following.

Open problem 4.5.3. *Is the query complexity of* TARSKI$(N, 4)$ $O(\log^2 N)$*?*

Improved lower bounds for the TARSKI problem would also be an interesting result. A first step could be the following.

Open problem 4.5.4. *For some* $d \in \mathbb{Z}_{>2}$ *is the query complexity of* TARSKI(N, d) $\omega(\log^2 N)$*?*

Although in [9] inclusion of TARSKI in **PPAD** $\cap$ **PLS** is shown, it remains an open problem if there are any natural complexity classes for which TARSKI is complete.

Open problem 4.5.5. *Is* TARSKI *complete for any natural complexity classes?*

Chapter 5

Testing the Algorithms

5.1 Overview

While the algorithms of previous sections are of great theoretical interest, questions remain on their practicality. To address this, I have implemented the algorithms and tested their performance on randomly generated instances of ARRIVAL, simple stochastic games, and shapley's stochastic games. The notable conclusions are roughly speaking as follows,

- the most basic algorithms of value iteration, and simulating the ARRIVAL walk outperform all of the binary search style algorithms in almost all cases in terms of query count and time,
- algorithm 3 of Fearnley, Pálvölgyi, and Savani tends to be the most performant of the binary search style algorithms on the problems tested here,
- the fixpoint decomposition method described in section 4.3 performs better than the asymptotically superior monotone decomposition method described in section 4.4.

5.2 Method

5.2.1 Algorithm Implementation Detail

5.2.1.1 Implementation

I implemented these algorithms in the progamming language C++. Complete source code can be found here including all algorithms in chapter 4, and solvers for all problems in chapter 3 using TARSKI algorithms. Compilation and linking was done with clang version 14.0.3 with the C++20 standard and `-O3` optimization settings. The tests were on my laptop with a 10-core Apple M2 CPU and 16GB of memory. Soplex[11] was used as a dependency to solve linear programs as part of the shapley's stochastic games solver. The shapley's stochastic game solver uses multithreading so performance

may differ if run on CPUs with less cores.

5.2.1.2 Performance Improvements

There are some performance optimizations that could be made. For simplicity of implementation[1], `std::vectors` are shuffled and appended to unnecessarily, and I believe performance could be gained by changing this. `std::function` is the main abstraction for passing the monotone functions around the system and are shown to be particularly innefficient in [20], so I believe performance can be gained by changing this to something like the `function_view` described in [20]. In running in the profiler, I found `soplex` was a bottleneck in solving shapley's stochastic games. Perhaps `soplex` is not optimized for solving a large number of very small LPs and a better alternative could be found. There is also perhaps scope for using sensitivity analysis as described in [23] to reuse values from previous function queries to improve solver performance; although this could be incredibly complex and not worth the effort.

5.2.2 Random Problem Generation

Instances of all three problems were generated randomly to facilitate testing. The method of randomization used for each instance is detailed in this subsection. Throughout random numbers were generated using the mersenne-twister[19] 19937 PRNG found in the `<random>` header in the C++ standard library.

5.2.2.1 ARRIVAL

Recall from section 3.1 that an instance of the arrival problem consists of a directed graph with a designated target vertex such that every vertex has exactly two labelled outgoing edges. This leads to a natural notion of a random arrival instance on n vertices $v_1, ..., v_n$. Simply choose for each vertex v_i the successors $s_0(v_i)$ and $s_1(v_i)$ uniformly at random from the set of vertices, and note that it is without loss of generality to fix the target to be v_n. Random instances for various fixed sizes of the ARRIVAL problem were generated thusly for testing.

5.2.2.2 Simple Stochastic Games

Simple stochastic games do not have as natural a notion of random problem instances as ARRIVAL for the following reasons,

- vertices can be one of three types,
- vertices can have different numbers of successors,
- chance vertices can have arbitrary probability distributions on their successors.

For simplicity, I generate a random simple stochastic game on n vertices $v_1, ..., v_n$ as follows,

[1] particularly in implementing slices of functions as described in definition 2.3.8

- the type of each vertex is chosen uniformly at random from the three possibilities,
- all vertices have exactly two successors,
- the probability distribution on the two successors of a chance node is chosen by partitioning the interval $[0,1]$ with a number chosen uniformly at random from the range $[0,1]$.
- v_n is fixed to be the target for the maximizer.

5.2.2.3 Shapley's Stochastic Games

The degrees of freedom for defining an instance of shapley's stochastic games are as follows,

- action sets can have arbitrary size at each state,
- for each joint action at each state, an arbitrary probability distribution on the all the states in the game can be chosen,
- payoffs for each joint action for each state can be chosen arbitrarily.

In order, these are addressed as follows,

- both players have three actions at every state,
- payoff and successor matrices are all 3×3 (which follows from the above item),
- every entry in every successor matrix is a probability distribution on exactly two vertices. That is to say that at every state when a joint action is chosen the transition is chosen to be one of two states,
- every probability distribution in the successor matrix is chosen as a u.a.r. partition of $[0,1]$ as in the simple stochastic game case,
- all entries of the payoff matrices are chosen to be u.a.r. integers in the range $[-10,10]$.

5.2.2.4 Limitations

Testing with random instances in this fashion is necessarily limited; the results shown later give evidence that random generation in this fashion does not tend to generate 'hard' instances. For instance, as will be shown in fig. 5.2, the length of the walk in a random instance of the ARRIVAL problem as described above seems to scale linearly with the size of the problem despite the fact that in the worst case the walk can have an exponential length.

5.2.3 Testing Protocol

Separate tests were carried out for the three problems detailed in chapter 3 as follows. For all problems, all of the algorithms were tested with varying instance sizes. For ARRIVAL all algorithms were also tested on instance with the longest possible walk as in fig. 3.3. For simple stochastic games and shapley's stochastic games all the algorithms

were also tested with a fixed problem size and varying approximation constant ε. In all tests, the number of queries to the monotone function is measured, and the time to run the algorithm is measured. The measured time is precisely the time between the function to run the algorithm is called, and the function returning with a fixpoint, so preprocessing and other miscellaneous actions do not have an effect. All tests were repated 20 times with the mean values recorded. Different sizes were used for different algorithms in the same test to ensure tests terminated in a reasonable amount of time.
The Kleene, Tarski algorithm 1 was not tested directly on any of the monotone functions. Instead, for shapley's and simple stochastic games, the continuous function is iterated on directly, and for ARRIVAL the walk is simulated directly. All of these are essentially the same as algorithm 1 but are slightly more performant due to bypassing unnecessary scaling in stochastic games, and not storing the number of times each node has been passed through in ARRIVAL.
From here on I will denote the Fearnley, Pálvölgyi, Savani algorithm described in algorithm 3 as FPS, the Dang, Qi, and Ye algorithm descbribed in algorithm 2 as DQY, Chen and Li described in section 4.4 as CL, and Kleene, Tarski described in algorithm 1 as KT.

5.3 Results

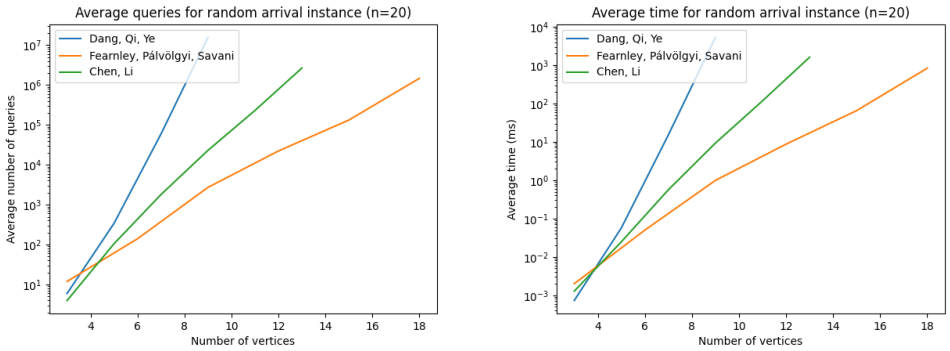

Figure 5.1: The binary search style algorithms take an approximately exponential time and amount of queries on random arrival instances in practice. FPS is the fastest.

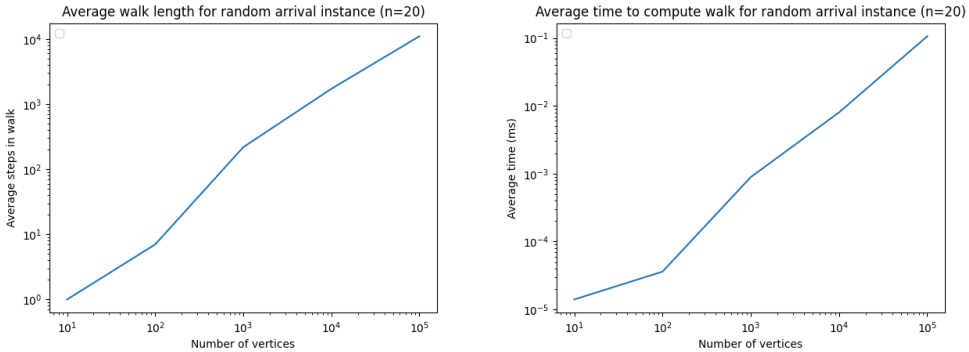

Figure 5.2: Time and number of steps taken to simulate the arrival walk scales roughly linearly with the size of the problem in practice. Simulating the walk is vastly more performant than binary search style algorithms for random arrival instances.

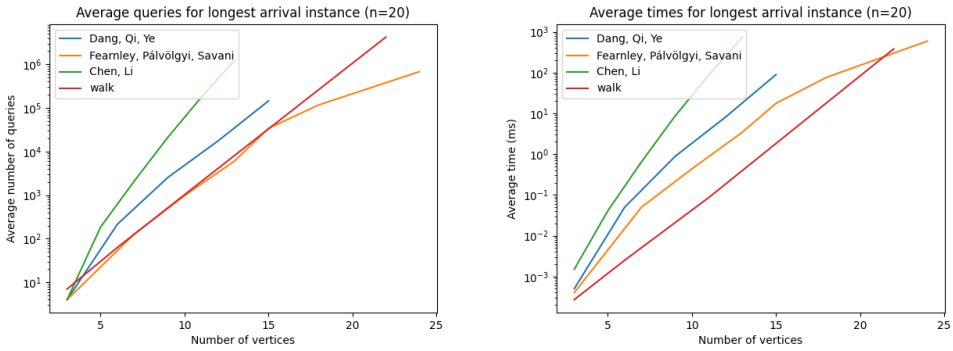

Figure 5.3: On the arrival instance with the longest possible walk as in fig. 3.3, FPS achieves similar performance to just simulating the walk.

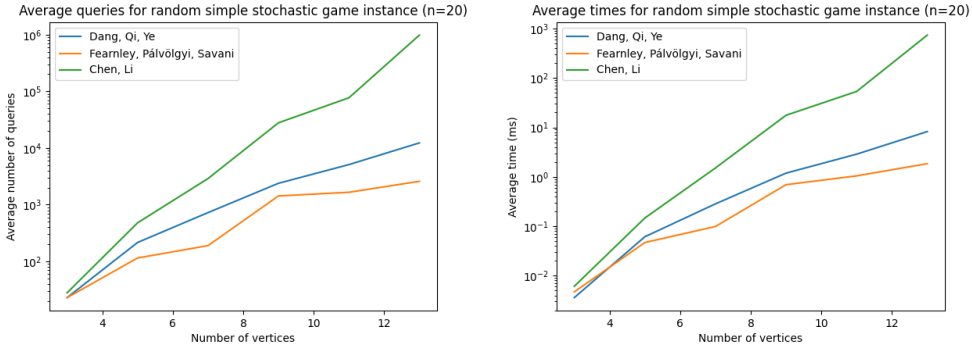

Figure 5.4: The binary search style algorithms take an approximately exponential amount of time and queries in the number of nodes to solve random simple stochastic games with fixed stopping probability and approximation constant.

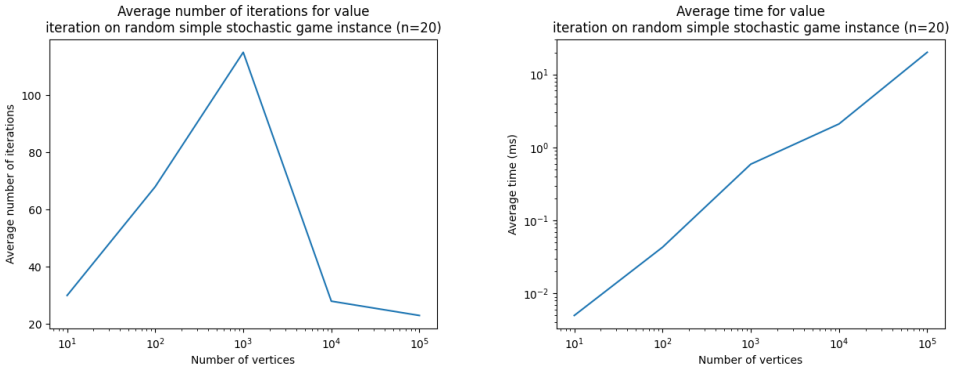

Figure 5.5: Value iteration takes approximately linear time and oddly downward scaling queries in the number of nodes to solve random simple stochastic games with fixed stopping probabilities and approximation constant. It is drastically more performant than the binary search style algorithms.

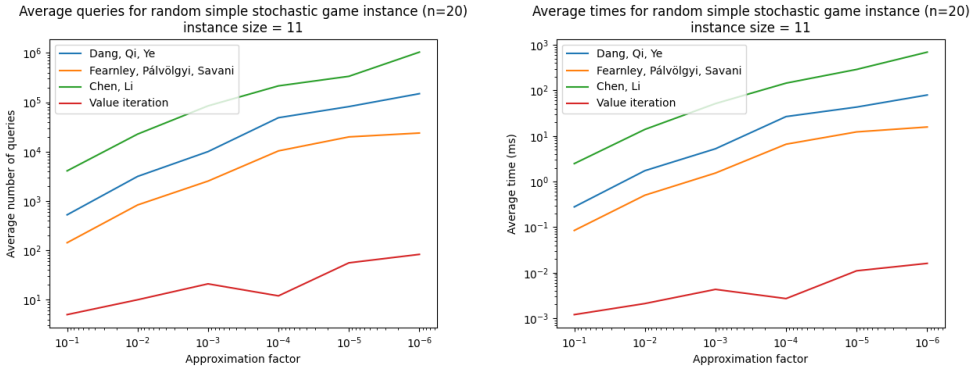

Figure 5.6: Value iteration easily outperforms the binary search style algorithms in solving simple stochastic games with a fixed number of nodes and varying approximation constant. Of the binary search algorithms, FPS is the most performant.

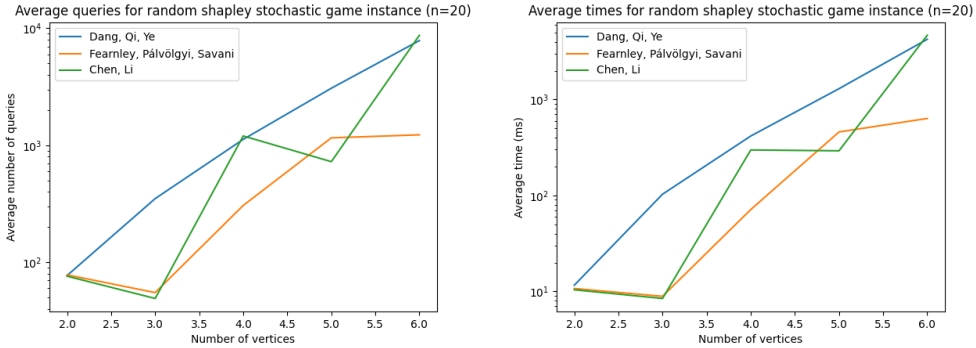

Figure 5.7: The binary search style algorithms take an approximately exponential time in solving random shapley's stochastic games. FPS and CL see similar performance for the sizes tested.

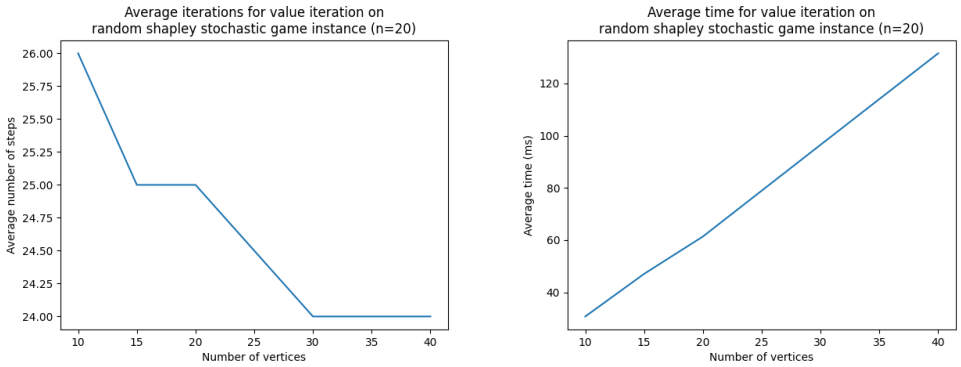

Figure 5.8: Value iteration takes approximately linear time and a close to constant number of queries to solve shapley's stochastic games on the sizes tested. It is much more performant than the binary search style algorithms. The constant factor involved in computing the monotone function is significant and results in longer solves than ARRIVAL and simple stochastic games.

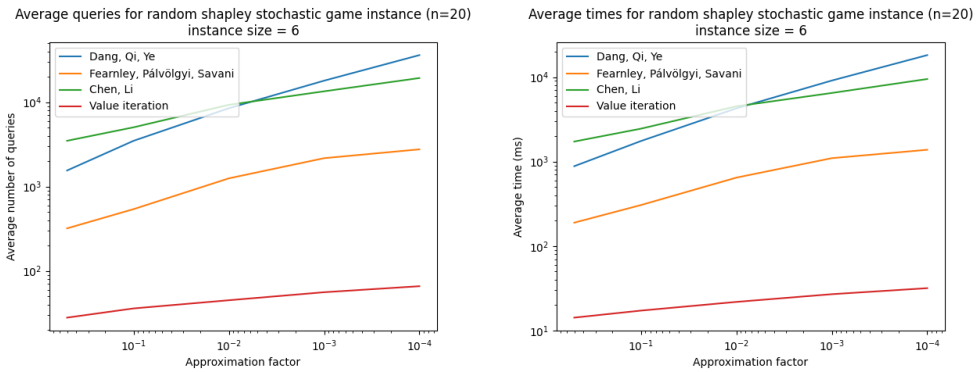

Figure 5.9: Value iteration outperforms the binary search style algorithms in solving shapley's stochastic games with fixed size and varying approximation constant.

All data used in these plots can be found here.

5.4 Discussion

On the whole, value iteration and simulating the ARRIVAL walk tend to be the most performant algorithms. Interestingly, FPS tends to be the most performant of the binary search style algorithms, beating out the asymptotically superior CL in almost all cases. More detailed discussion is given for individual cases below.

5.4.1 ARRIVAL

It can clearly be seen in fig. 5.1 and fig. 5.2 that simulating the arrival walk vastly outperforms the binary-search based algorithms on randomly generated instances of the arrival problem. The distinction is less clear when testing with the worst case instance for the walk as seen in fig. 5.3; FPS outperforms the walk in terms of query count for more than 15 vertices, and in terms of time for more than 20. This is somewhat unexpected as the number of steps in the walk on the worst case instance is $\Theta(2^n)$, while the upper bound we get from FPS when applied to the walk is $O(\log^{\lceil (n+2)/3 \rceil}(2^n)) = O(n^{\lceil (n+2)/3 \rceil})$, which is clearly a weaker bound than $O(2^n)$. I believe the cause of this is that the recursive binary search algorithms work particularly well on this specific long walk instance; the fixpoint for this specific instance will be something like $\vec{x} = (2^n, 2^{n-1}, \ldots, 1)$, and the binary search algorithms always start by querying midpoints which happen to be powers of two so coincide exactly with the actual fixpoint. This perhaps motivates further testing- are there other instances for which FPS outperforms the walk?
In comparing the binary search style algorithms, it can be seen that FPS performs best in all cases, and in particular performs better than the asymptotically superior CL algorithm. The difference between CL and DQY is less clear however; for random instances CL performs better as seen in fig. 5.1, but DQY performs better on the long walk instance as seen in fig. 5.3.

5.4.2 Simple Stochastic Games

Similarly to ARRIVAL, value iteration is the most performant algorithm for solving random simple stochastic games - it even seems to be the case that the number of queries goes *down* as the number of vertices goes up for the walk as seen in fig. 5.5. I believe that this is a limitation of the method of random instance generation which perhaps motivates further investigation into methods for generating 'hard' simple stochastic games. It could also be the case that testing with a fixed approximiation factor ε and stopping probability β causes this behaviour.
In comparing the binary search based algorithms, it can be seen in fig. 5.4 that FPS is again the most performant, and DQY is the least performant. The difference between FPS and CL is small in this case however.
In the test with varying approximation factor as seen in fig. 5.6, value iteration is again the most performant, with FPS the best of the binary search algorithms. DQY is again the slowest, with CL in the middle. Varying the approximation factor for the simple stochastic game problem has the effect of changing the height of the lattice that is searched; if β is the stopping probability, and ε is the approximation factor, the associated discretized function is defined on $\left[\left\lfloor \frac{1}{\beta\varepsilon} \right\rfloor\right]^d$. Since KT runs in worst case complexity $O(Nd)$ where N is the height of the lattice, and FPS in $O(\log^{\lceil \frac{2n+2}{3} \rceil} N)$, one might expect that for very small approximation factors that FPS is more performant. This is not shown in the results however, so perhaps more investigation should be done into finding instances of simple stochastic games which are 'hard' for value iteration.

5.4.3 Shapley's Stochastic Games

Testing on shapley's stochastic games was much more limited than the other two problems as the associated monotone function took a lot longer to compute. It could be the case the the LP solver that I used (`soplex`[11]) is not optimized for solving many thousands of small LPs, or that solving LPs in this fashion will necessarily take significantly more time than the functions for ARRIVAL and simple stochastic games.
In comparing the algorithms running on random instances, it can be seen in fig. 5.7 and fig. 5.8 that again value iteration is the most performant. DQY is the least performant on random instances, and the difference between CL and FPS is small. Interestingly, there seems to be some association between the parity of the dimension and the performance of CL and FPS. I believe this is because of the $\lceil \cdot \rceil$ in the exponent of their complexities caused by subproblems with dimension less than 3 during decomposition, and the only reason this is not seen in other plots is because measurements are taken with less granularity on size. Perhaps this motivates testing with more datapoints on the other problems as well.
The test with varying approximation factor as seen in fig. 5.9 shows again that value iteration is the most performant, and FPS is the most performant of the binary search algorithms. There is not much difference between CL and DQY in this case. Similarly to simple stochastic games, one would expect that for very small approximation factors that the binary search algorithms perform better than value iteration - but again this is not seen in the results for these tests. This is again perhaps motivation for more investigation in finding 'hard' stochastic games for the value iteration algorithm.

In shapley's stochastic games, many parameters were also unchanged through all tests. The maximum value in the payoff matrix for shapley's could be varied, the stopping probabilities for both, the number of actions at each state in shapley's, and the number of successors for all states in both problems could all be varied.

5.5 Further Work

The main shortcoming of the testing I have carried out is the random instance generation described in section 5.2.2. In all problems, the results such as fig. 5.2 give strong evidence that for the most basic algorithms like simulating the walk and value iteration, my scheme for random problem generation does not generate 'hard' problems so investigation into this could be carried out. Another extension could be to implement strategy improvement and quadratic programming to solve simple stochastic games and further compare.
In [17] both of these issues are somewhat addressed when testing value iteration, strategy improvement, and quadratic programming on simple stochatic games. The authors build a library of extremal problems that are difficult for particular algorithms, as well as a more sophisticated method of generating random problems than I have done here. Perhaps transitive comparisons can be made to results in [17] given I have both tested value iteration, although the difference in how I generate problems makes this tenuous. Quadratic programming and strategy improvement based solvers could also be implemented and tested for simple stochastic games against all algorithms here.
Time could be spent on on optimizing the monotone function for shapley's stochastic games, as tests have been limited to small sizes so an increase in performance could yield more interesting data.
The asymptotic advantage the binary-search style algorithms achieve over the iterative algorithms is dependent on the ratio between the height of the lattice and it's dimension. In particular the asymptotic bound for FPS is stronger than KT if $d \in o(\frac{\log N}{\log \log N})$. In simple and shapley's stochastic games, solving to higher precision ε causes an increase in the height of the searched lattice so testing with very high precision approximation constraints along with finding 'hard' instances of these problems could demonstrate a practical advantage to FPS over value iteration. This would require additional support in my implementation for high precision numerical types such as GMP[12], and I believe finding an LP solver for shapley's stochastic games with strong support for high precision types will also be difficult.

Chapter 6

Conclusions

6.1 Testing the Algorithms

The main conclusions drawn from the practical section of this project are the following,

- the simplest iterative algorithms of value iteration and simulating the ARRIVAL walk perform better in almost all cases on the model of random instances of ARRIVAL, simple stochastic games, and shapley's stochastic games described in section 5.2.2,
- the asymptotically superior monotone decomposition algorithm 5 tends to require more queries and time than the simpler fixpoint decomposition algorithm 3,
- the inner algorithm combined with the fixpoint decomposition algorithm 3 tends to be the most performant of all of the binary search algorithms.
- one case where the fixpoint decomposition 3 performs better than simulating the ARRIVAL walk was found in fig. 5.3, although I believe this to be anomalous due to the nature of the specific problem being particularly easy for the binary-search style algorithms.

The main limitation of testing was the methods of generating problems for testing; randomly generated instances of the ARRIVAL problem proved to be relatively easy in comparison to the worst case, with similar findings for simple stochastic games and shapley's stochastic games. The solver for shapley's stochastic games was particularly slow so testing was restricted to lower dimensional versions of the problem.

6.2 State of the Art Algorithms

The key takeaway is the many theoretical questions relating to the TARSKI problem remain open. These include,

- open problem 4.5.2. Is TARSKI fixed-parameter tractable? The inner algorithm from [10] described in section 4.2 makes this seem somewhat plausible. Is there notion of a 'witness' in higher than 3 dimensions?

- open problem 4.5.4. Is there an improved lower bound for the TARSKI problem in some dimension larger than 3? Perhaps the methods in proving the lower bound for the 2 dimensional variant in [9] have an analogue in some dimension larger than 3.
- open problem 4.5.5. Is TARSKI complete for some natural complexity class? Although in [9] inclusion of TARSKI in **PPAD** $\cap$ **PLS** is shown, there is no non-trivial complexity class for which TARSKI is known to be complete. Such a result could be useful on giving evidence on the hardness of TARSKI.

6.3 Future Work

With regard to the practical testing of the TARSKI algorithms, some ideas for future improvements are as follows,

- can problem generation for all three problems be improved?
  - is there a method of generating random ARRIVAL instances which have a comparable walk-length to the worst-case instance?
  - are there 'long' ARRIVAL instances other than the worst-case described in fig. 3.3 for which the fixpoint decomposition still outperforms simulating the walk as in fig. 5.3?
  - does randomizing other parameters like number of successors, stopping probability, and payoff matrix entry size, of simple stochastic games and shapley's stochastic games have any effect on the relative performance of all of the algorithms?
- can the performance of the implementation of solving shapley's stochastic games using TARSKI algorithms be improved? The slowness of solving many linear programs for the monotone function for shapley's stochastic games resulted in the experiment for this problem being limited in size.

For future work on the theoretical aspects of TARSKI and related problems, I propose the following problems as being worth some thought,

- open problem 4.5.3. Is there an algorithm for solving the four dimensional TARSKI$(N,4)$ problem in $O(\log^2 N)$ queries? Generalizing the inner algorithm to arbitrary dimensions is potentially a very difficult problem, and working on the TARSKI$(N,4)$ problem would seem to be a reasonable intermediate step. A result in the positive could perhaps also be illuminating to higher dimensions.
- is there anything to be gleaned from studying the TARSKI problem in the context of monotone functions from instances of the ARRIVAL problem? ARRIVAL problem instances have a very simple combinatorial structure and there are already interesting connections between the ARRIVAL and TARSKI through the reduction given in section 3.1. Is there an interpretation of any of the features in the state of the art algorithms in the context of ARRIVAL? And do they provide any insight on how to make progress with the general problem?

Bibliography

[1] Ching-Lueh Chang, Yuh-Dauh Lyuu, and Yen-Wu Ti. The complexity of tarski's fixed point theorem. *Theoretical Computer Science*, 401(1):228–235, 2008. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2008.05.005. URL https://www.sciencedirect.com/science/article/pii/S0304397508003812.

[2] Krishnendu Chatterjee and Thomas A. Henzinger. *Value Iteration*, pages 107–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-69850-0. doi: 10.1007/978-3-540-69850-0_7. URL https://doi.org/10.1007/978-3-540-69850-0_7.

[3] Xi Chen and Yuhao Li. Improved upper bounds for finding tarski fixed points. 2022.

[4] V. Chvátal. *Linear Programming*. A series of books in the mathematical sciences. W. H. Freeman, 1983. ISBN 9781429280518. URL https://books.google.co.uk/books?id=05QZvgAACAAJ.

[5] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992. ISSN 0890-5401. doi: https://doi.org/10.1016/0890-5401(92)90048-K. URL https://www.sciencedirect.com/science/article/pii/089054019290048K.

[6] Chuangyin Dang, Qi Qi, and Yinyu Ye. Computations and complexities of tarski's fixed points and supermodular games. 2020.

[7] Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. Arrival: A zero-player graph game in np ∩ conp. 2017.

[8] Kousha Etessami and Mihalis Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010. doi: 10.1137/080720826. URL https://doi.org/10.1137/080720826.

[9] Kousha Etessami, Christos Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. Tarski's theorem, supermodular games, and the complexity of equilibria. 2019.

[10] John Fearnley, Dömötör Pálvölgyi, and Rahul Savani. A faster algorithm for finding tarski fixed points. 2021.

[11] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald,

Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The scip optimization suite 7.0. Technical Report 20-10, ZIB, Takustr. 7, 14195 Berlin, 2020.

[12] Torbjrn Granlund and Gmp Development Team. *GNU MP 6.0 Multiple Precision Arithmetic Library*. Samurai Media Limited, London, GBR, 2015. ISBN 9789888381968.

[13] Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. Arrival: Next stop in cls, 2018.

[14] Bernd Gärtner, Sebastian Haslebacher, and Hung P. Hoang. A subexponential algorithm for arrival. 2021.

[15] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/2627820.

[16] Koos Vrieze Jerzy Filar. *Competitive Markov Decision Processes*. 1997. ISBN 978-0-387-94805-8.

[17] Jan Křetínský, Emanuel Ramneantu, Alexander Slivinskiy, and Maximilian Weininger. Comparison of algorithms for simple stochastic games. *Electronic Proceedings in Theoretical Computer Science*, 326:131–148, September 2020. ISSN 2075-2180. doi: 10.4204/eptcs.326.9. URL http://dx.doi.org/10.4204/EPTCS.326.9.

[18] Graham Manuell. A simple lower bound for arrival, 2021.

[19] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, jan 1998. ISSN 1049-3301. doi: 10.1145/272991.272995. URL https://doi.org/10.1145/272991.272995.

[20] Vittorio Romeo. passing functions to functions, 2017. URL https://vittorioromeo.info/index/blog/passing_functions_to_functions.html.

[21] L. S. Shapley. Stochastic games*. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi: 10.1073/pnas.39.10.1095. URL https://www.pnas.org/doi/abs/10.1073/pnas.39.10.1095.

[22] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285 – 309, 1955.

[23] James E. Ward and Richard E. Wendell. Approaches to sensitivity analysis in linear programming. *Annals of Operations Research*, 27(1):3–38, 1990. doi: 10.1007/BF02055188. URL https://doi.org/10.1007/BF02055188.