The complexity of computing a Tarski fixed point of a monotone function, with applications to games and equilibibria

Kousha Etessami

University of Edinburgh

Simons Institute
Games and Equilibria Workshop
February, 2021

(This talk is based on joint work with:
C. Papadimitriou, A. Rubinstein, and M. Yannakakis,
in a paper that appeared at ITCS'2020.)

Tarski's Fixed Point Theorem

Recall: A partially ordered set $(L, \leq)$ is a **complete lattice** if every non-empty subset $S \subseteq L$ has both a *least upper bound* (or *supremum* or *join*), and a *greatest lower bound* (or *infimum* or *meet*) in L.

A function $f : L \to L$ is **monotone** if

$$\forall x, y \in L, \quad x \leq y \;\Rightarrow f(x) \leq f(y).$$

Let $\mathrm{Fix}(f) := \{x \in L \mid x = f(x)\}$ denote the set of **fixed points** of f.

Tarski's Fixed Point Theorem

Recall: A partially ordered set $(L, \leq)$ is a **complete lattice** if every non-empty subset $S \subseteq L$ has both a *least upper bound* (or *supremum* or *join*), and a *greatest lower bound* (or *infimum* or *meet*) in L.

A function $f : L \to L$ is **monotone** if

$$\forall x, y \in L, \quad x \leq y \ \Rightarrow f(x) \leq f(y).$$

Let $\mathrm{Fix}(f) := \{x \in L \mid x = f(x)\}$ denote the set of **fixed points** of f.

Theorem [Tarski, 1955]

Every monotone function $f : L \to L$ from a complete lattice $(L, \leq)$ to itself, has a non-empty set $\mathrm{Fix}(f)$ of fixed points, which themselves form a complete lattice $(\mathrm{Fix}(f), \leq)$ under the same partial order $\leq$.

(In particular, f has a *Least Fixed Point*, and a *Greatest Fixed Point*.)

Question: how hard is it to compute a (any) fixed point of a given monotone function $f : L \to L$ when $(L, \leq)$ is finite?

Question: how hard is it to compute a (any) fixed point of a given monotone function $f : L \rightarrow L$ when $(L, \leq)$ is finite?

More specifically, how hard is it when $(L, \leq)$ is the d-dimensional *euclidean grid lattice*, $L = \{1, \ldots, N\}^d = [N]^d$, under the standard coordinate-wise partial order $\leq$ on vectors.

In other words, by definition, for $x, y \in [N]^d$:

$$x \leq y \quad \Leftrightarrow \quad x_i \leq y_i, \quad \text{for all } i \in \{1, \ldots, d\}.$$

Question: how hard is it to compute a (<u>any</u>) fixed point of a given monotone function $f : L \rightarrow L$ when $(L, \leq)$ is finite?

More specifically, how hard is it when $(L, \leq)$ is the d-dimensional *euclidean grid lattice*, $L = \{1, \ldots, N\}^d = [N]^d$, under the standard coordinate-wise partial order $\leq$ on vectors.

In other words, by definition, for $x, y \in [N]^d$:

$$x \leq y \quad \Leftrightarrow \quad x_i \leq y_i, \;\; \text{for all } i \in \{1, \ldots, d\}.$$

As we will see, this question has important applications, including for:

- equilibrium computation problems (for *supermodular games*),
- solving (i.e., computing the *value* of) *stochastic games*.

And we will see that this problem has an intriguing computational complexity status....

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \rightarrow [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}$, $f(\bar{1})$, $f(f(\bar{1}))$, $\ldots$, $f^i(\bar{1})$, $\ldots$

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \to [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}$, $f(\bar{1})$, $f(f(\bar{1}))$, $\ldots$, $f^i(\bar{1})$, $\ldots$

From monotonicity of f, it follows that for all $i \geq 0$, $f^i(\bar{1}) \leq f^{i+1}(\bar{1})$.

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \rightarrow [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}\ ,\ f(\bar{1})\ ,\ f(f(\bar{1}))\ ,\ \ldots\ ,\ f^i(\bar{1})\ ,\ \ldots$

From monotonicity of f, it follows that for all $i \geq 0$, $f^i(\bar{1}) \leq f^{i+1}(\bar{1})$.

Hence, unless we reach a fixed point $f^i(\bar{1}) = f^{i+1}(\bar{1})$, the sum of the coordinates must increase by at least 1 in each iteration.

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \rightarrow [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}\ ,\ f(\bar{1})\ ,\ f(f(\bar{1}))\ ,\ \ldots\ ,\ f^i(\bar{1})\ ,\ \ldots$

From monotonicity of f, it follows that for all $i \geq 0$, $f^i(\bar{1}) \leq f^{i+1}(\bar{1})$.

Hence, unless we reach a fixed point $f^i(\bar{1}) = f^{i+1}(\bar{1})$, the sum of the coordinates must increase by at least 1 in each iteration.
Hence, we will reach a fixed point in at most $d \cdot N$ iterations.

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \rightarrow [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}\ ,\ f(\bar{1})\ ,\ f(f(\bar{1}))\ ,\ \ldots\ ,\ f^i(\bar{1})\ ,\ \ldots$

From monotonicity of f, it follows that for all $i \geq 0$, $f^i(\bar{1}) \leq f^{i+1}(\bar{1})$.

Hence, unless we reach a fixed point $f^i(\bar{1}) = f^{i+1}(\bar{1})$, the sum of the coordinates must increase by at least 1 in each iteration.
Hence, we will reach a fixed point in at most $d \cdot N$ iterations.

The fixed point computed by this is the *least fixed point* (LFP).

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \to [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}\ ,\ f(\bar{1})\ ,\ f(f(\bar{1}))\ ,\ \ldots\ ,\ f^i(\bar{1})\ ,\ \ldots$

From monotonicity of f, it follows that for all $i \geq 0$, $f^i(\bar{1}) \leq f^{i+1}(\bar{1})$.

Hence, unless we reach a fixed point $f^i(\bar{1}) = f^{i+1}(\bar{1})$, the sum of the coordinates must increase by at least 1 in each iteration.
Hence, we will reach a fixed point in at most $d \cdot N$ iterations.

The fixed point computed by this is the *least fixed point* (LFP).

We can similarly compute the *greatest fixed point* (GFP) within $d \cdot N$ iterations, by starting instead at the top element $\overline{N} = (N, N, \ldots, N)$.

First, an easy classic algorithm: Kleene/Tarski iteration

To find a fixed point of $f : [N]^d \rightarrow [N]^d$:

start with $\bar{1} = (1, 1, \ldots, 1)$, the bottom element of $[N]^d$, and compute the sequence: $\bar{1}$, $f(\bar{1})$, $f(f(\bar{1}))$, ... , $f^i(\bar{1})$, ...

From monotonicity of f, it follows that for all $i \geq 0$, $f^i(\bar{1}) \leq f^{i+1}(\bar{1})$.

Hence, unless we reach a fixed point $f^i(\bar{1}) = f^{i+1}(\bar{1})$, the sum of the coordinates must increase by at least 1 in each iteration.
Hence, we will reach a fixed point in at most $d \cdot N$ iterations.

The fixed point computed by this is the *least fixed point* (LFP).

We can similarly compute the *greatest fixed point* (GFP) within $d \cdot N$ iterations, by starting instead at the top element $\overline{N} = (N, N, \ldots, N)$.

Question: Suppose we don't care which fixed point we compute. Suppose we just want to compute some fixed point.
Can we do better than $d \cdot N$ iterations?

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \rightarrow [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \rightarrow [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

For $a, b \in [N]^d$, where $a \leq b$, let $L(a, b) = \{x \in [N]^d \mid a \leq x \leq b\}$, denote the sublattice of grid points between a and b. So, $[N]^d = L(\overline{1}, \overline{N})$.

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \rightarrow [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

For $a, b \in [N]^d$, where $a \leq b$, let $L(a, b) = \{x \in [N]^d \mid a \leq x \leq b\}$, denote the sublattice of grid points between a and b. So, $[N]^d = L(\bar{1}, \overline{N})$.

<u>1-dimensional case</u>: for $d = 1$, suppose we are given a monotone function $f : L(a, b) \rightarrow L(a, b)$, with $1 \leq a \leq b \leq N$.
We can compute a fixed point of f by binary search, as follows:

Let $mid := \lfloor \frac{a+b}{2} \rfloor$. Evaluate $f(mid)$.

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \rightarrow [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

For $a, b \in [N]^d$, where $a \leq b$, let $L(a, b) = \{x \in [N]^d \mid a \leq x \leq b\}$, denote the sublattice of grid points between a and b. So, $[N]^d = L(\overline{1}, \overline{N})$.

1-dimensional case: for $d = 1$, suppose we are given a monotone function $f : L(a, b) \rightarrow L(a, b)$, with $1 \leq a \leq b \leq N$.
We can compute a fixed point of f by binary search, as follows:

Let $mid := \lfloor \frac{a+b}{2} \rfloor$. Evaluate $f(mid)$. If $f(mid) = mid$, we are done.

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \rightarrow [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

For $a, b \in [N]^d$, where $a \leq b$, let $L(a, b) = \{x \in [N]^d \mid a \leq x \leq b\}$, denote the sublattice of grid points between a and b. So, $[N]^d = L(\bar{1}, \overline{N})$.

<u>1-dimensional case</u>: for $d = 1$, suppose we are given a monotone function $f : L(a, b) \rightarrow L(a, b)$, with $1 \leq a \leq b \leq N$.
We can compute a fixed point of f by binary search, as follows:

Let $mid := \lfloor \frac{a+b}{2} \rfloor$. Evaluate $f(mid)$. If $f(mid) = mid$, we are done.

If $f(mid) > mid$, then $f : L(mid, b) \rightarrow L(mid, b)$ is montone and has a fixed point.

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \rightarrow [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

For $a, b \in [N]^d$, where $a \leq b$, let $L(a, b) = \{x \in [N]^d \mid a \leq x \leq b\}$, denote the sublattice of grid points between a and b. So, $[N]^d = L(\overline{1}, \overline{N})$.

<u>1-dimensional case</u>: for $d = 1$, suppose we are given a monotone function $f : L(a, b) \rightarrow L(a, b)$, with $1 \leq a \leq b \leq N$.
We can compute a fixed point of f by binary search, as follows:

Let $mid := \lfloor \frac{a+b}{2} \rfloor$. Evaluate $f(mid)$. If $f(mid) = mid$, we are done.

If $f(mid) > mid$, then $f : L(mid, b) \rightarrow L(mid, b)$ is montone and has a fixed point.

Likewise, if $f(mid) < mid$, then $f : L(a, mid) \rightarrow L(a, mid)$ is monotone and has a fixed point.

An $O(\log^d N)$ algorithm ([Dang-Qi-Ye,2012])

We can compute a fixed point of $f : [N]^d \to [N]^d$, using binary search combined with recursion, in $O(\log^d N)$ queries to the function f.

For $a, b \in [N]^d$, where $a \leq b$, let $L(a, b) = \{x \in [N]^d \mid a \leq x \leq b\}$, denote the sublattice of grid points between a and b. So, $[N]^d = L(\bar{1}, \overline{N})$.

<u>1-dimensional case</u>: for $d = 1$, suppose we are given a monotone function $f : L(a, b) \to L(a, b)$, with $1 \leq a \leq b \leq N$.
We can compute a fixed point of f by binary search, as follows:

Let $mid := \lfloor \frac{a+b}{2} \rfloor$. Evaluate $f(mid)$. If $f(mid) = mid$, we are done.

If $f(mid) > mid$, then $f : L(mid, b) \to L(mid, b)$ is montone and has a fixed point.

Likewise, if $f(mid) < mid$, then $f : L(a, mid) \to L(a, mid)$ is monotone and has a fixed point.

So, by repeating, we can find a fixed point of $f : [N] \to [N]$ in $\log N$ iterations (i.e., $\log N$ function evaluations).

For dimensions $d > 1$, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$, be defined as follows:
for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.

For dimensions $d > 1$, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$, be defined as follows:
for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.
Note that $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$ defines a monotone function.

For dimensions $d > 1$, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$, be defined as follows:
for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.
Note that $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$ defines a monotone function.
Let $c := \lfloor \frac{N+1}{2} \rfloor$.

<u>For dimensions $d > 1$</u>, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \rightarrow [N]^{d-1}$, be defined as follows:
for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.
Note that $f\langle c\rangle : [N]^{d-1} \rightarrow [N]^{d-1}$ defines a monotone function.
Let $c := \lfloor \frac{N+1}{2} \rfloor$. If we (recursively) compute $x^* \in \mathrm{Fix}(f\langle c\rangle)$, then either:

- $f_d(x^*, c) = c$, in which case $(x^*, c) \in \mathrm{Fix}(f)$ and we are done; or
- $f_d(x^*, c) > c$, in which case $f : L((x^*, c), \overline{N}) \rightarrow L((x^*, c), \overline{N})$ is monotone, and we have "halved" the range of values to consider in the last coordinate in our search for a fixed point of f; or,
- $f_d(x^*, c) < c$, in which case $f : L(\overline{1}, (x^*, c)) \rightarrow L(\overline{1}, (x^*, c))$, and we have again "halved" the range of values to consider in the last coordinate in our search for a fixed point of f.

For dimensions $d > 1$, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$, be defined as follows: for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.
Note that $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$ defines a monotone function.
Let $c := \lfloor \frac{N+1}{2} \rfloor$. If we (recursively) compute $x^* \in \mathrm{Fix}(f\langle c\rangle)$, then either:

- $f_d(x^*, c) = c$, in which case $(x^*, c) \in \mathrm{Fix}(f)$ and we are done; or
- $f_d(x^*, c) > c$, in which case $f : L((x^*, c), \overline{N}) \to L((x^*, c), \overline{N})$ is monotone, and we have "halved" the range of values to consider in the last coordinate in our search for a fixed point of f; or,
- $f_d(x^*, c) < c$, in which case $f : L(\overline{1}, (x^*, c)) \to L(\overline{1}, (x^*, c))$, and we have again "halved" the range of values to consider in the last coordinate in our search for a fixed point of f.

Applying this recursively yields, by induction, an algorithm that requires at most $\log^{d-1} N \cdot \log N = \log^d N$ function evaluations to compute a fixed point of a monotone function $f : [N]^d \to [N]^d$.

For dimensions $d > 1$, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \rightarrow [N]^{d-1}$, be defined as follows: for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.
Note that $f\langle c\rangle : [N]^{d-1} \rightarrow [N]^{d-1}$ defines a monotone function.
Let $c := \lfloor \frac{N+1}{2} \rfloor$. If we (recursively) compute $x^* \in \mathrm{Fix}(f\langle c\rangle)$, then either:

- $f_d(x^*, c) = c$, in which case $(x^*, c) \in \mathrm{Fix}(f)$ and we are done; or
- $f_d(x^*, c) > c$, in which case $f : L((x^*, c), \overline{N}) \rightarrow L((x^*, c), \overline{N})$ is monotone, and we have "halved" the range of values to consider in the last coordinate in our search for a fixed point of f; or,
- $f_d(x^*, c) < c$, in which case $f : L(\overline{1}, (x^*, c)) \rightarrow L(\overline{1}, (x^*, c))$, and we have again "halved" the range of values to consider in the last coordinate in our search for a fixed point of f.

Applying this recursively yields, by induction, an algorithm that requires at most $\log^{d-1} N \cdot \log N = \log^d N$ function evaluations to compute a fixed point of a monotone function $f : [N]^d \rightarrow [N]^d$.

Note: for $d \in o(\frac{\log N}{\log \log N})$, $\log^d N$ is better than $d \cdot N$.

For dimensions $d > 1$, we procede recursively:
For $c \in [N]$, let $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$, be defined as follows: for $x \in [N]^{d-1}$ and $i \in \{1, \ldots, d-1\}$, let $f\langle c\rangle_i(x) := f_i(x, c)$.
Note that $f\langle c\rangle : [N]^{d-1} \to [N]^{d-1}$ defines a monotone function.
Let $c := \lfloor \frac{N+1}{2} \rfloor$. If we (recursively) compute $x^* \in \mathrm{Fix}(f\langle c\rangle)$, then either:

- $f_d(x^*, c) = c$, in which case $(x^*, c) \in \mathrm{Fix}(f)$ and we are done; or
- $f_d(x^*, c) > c$, in which case $f : L((x^*, c), \overline{N}) \to L((x^*, c), \overline{N})$ is monotone, and we have "halved" the range of values to consider in the last coordinate in our search for a fixed point of f; or,
- $f_d(x^*, c) < c$, in which case $f : L(\overline{1}, (x^*, c)) \to L(\overline{1}, (x^*, c))$, and we have again "halved" the range of values to consider in the last coordinate in our search for a fixed point of f.

Applying this recursively yields, by induction, an algorithm that requires at most $\log^{d-1} N \cdot \log N = \log^d N$ function evaluations to compute a fixed point of a monotone function $f : [N]^d \to [N]^d$.

Note: for $d \in o(\frac{\log N}{\log \log N})$, $\log^d N$ is better than $d \cdot N$.
Can we do better?

The Tarski Problem

Let's define our task as an explicit computational problem:

Definition (the Tarski problem)

Input: A function $f : [N]^d \to [N]^d$ with $N = 2^n$ for some $n \geq 1$, given by a boolean circuit, C_f, with $(d \cdot n)$ input gates and $(d \cdot n)$ output gates.
Output: Either a (any) fixed point $x^* \in \mathrm{Fix}(f)$, or else a witness pair of vectors $x, y \in [N]^d$ such that $x \leq y$ and $f(x) \not\leq f(y)$.

- Note: Tarski is a *total* search problem: if f is monotone, it will contain a fixed point in $[N]^d$, and otherwise it will contain such a witness pair of vectors that exhibit non-monotonicity.
- (If f is non-monotone it may of course have both witnesses for non-monotonicity and fixed points; either output will do.)

The $\texttt{Tarski}$ Problem

Let's define our task as an explicit computational problem:

Definition (the $\texttt{Tarski}$ problem)

Input: A function $f : [N]^d \to [N]^d$ with $N = 2^n$ for some $n \geq 1$, given by a boolean circuit, C_f, with $(d \cdot n)$ input gates and $(d \cdot n)$ output gates.
Output: Either a (any) fixed point $x^* \in \mathrm{Fix}(f)$, or else a witness pair of vectors $x, y \in [N]^d$ such that $x \leq y$ and $f(x) \not\leq f(y)$.

- Note: $\texttt{Tarski}$ is a *total* search problem: if f is monotone, it will contain a fixed point in $[N]^d$, and otherwise it will contain such a witness pair of vectors that exhibit non-monotonicity.
- (If f is non-monotone it may of course have both witnesses for non-monotonicity and fixed points; either output will do.)

Question: What is the complexity of this total search problem?

A harder problem: computing the LFP (or the GFP)

Question: What if, instead of the `Tarski` problem, our task was to compute the LFP, rather than just <u>any</u> fixed point?

A harder problem: computing the LFP (or the GFP)

Question: What if, instead of the `Tarski` problem, our task was to compute the LFP, rather than just <u>any</u> fixed point?

Proposition

1. Given a monotone $f : [N] \rightarrow [N]$, $N = 2^n$, by a boolean circuit, (i.e., already for $d = 1$) it is **NP**-hard to compute the LFP of f. (Likewise, it is **NP**-hard to compute the GFP of f.)
2. Given a montone $f : [N] \rightarrow [N]$ by an oracle, computing the LFP requires $\Omega(N)$ queries to f. (Likewise for the GFP.)

The proofs are easy: (1.) is a simple reduction from SAT.
For (2.): let $f : [N] \rightarrow [N]$ be the family of monotone functions where $f(N) := N$, and for all $x \in \{1, \ldots, N-1\}$, $f(x) \in \{x, x+1\}$. The LFP of such an f is $\neq N$ iff $\exists x \in \{1, \ldots, N-1\}$ such that $f(x) = x$. In the oracle model, finding such an x requires trying all $x \in \{1, \ldots, N-1\}$. $\square$

Towards the complexity of `Tarski`: some standard total search complexity classes.

Two very well-studied discrete total search complexity classes are:

- **PLS** (*Polynomial Local Search*) [JPY'88]
- **PPAD** (*Polynomial Parity Argument – Directed*) [P'94]

PLS consists of discrete local search problems that can be phrased as follows: given a instance $I \in \{0,1\}^*$, and a start "solution" $x \in \{0,1\}^{p(|I|)}$, compute a "locally optimal" solution, $x^* \in \{0,1\}^{p(|I|)}$, with respect to a (P-time computable) objective function $g_I(x)$, and (P-time computable) neighborhood function, $\mathcal{N}_I(x)$.

Towards the complexity of $\texttt{Tarski}$: some standard total search complexity classes.

Two very well-studied discrete total search complexity classes are:

- **PLS** (*Polynomial Local Search*) [JPY'88]
- **PPAD** (*Polynomial Parity Argument – Directed*) [P'94]

PLS consists of discrete local search problems that can be phrased as follows: given a instance $I \in \{0,1\}^*$, and a start "solution" $x \in \{0,1\}^{p(|I|)}$, compute a "locally optimal" solution, $x^* \in \{0,1\}^{p(|I|)}$, with respect to a (P-time computable) objective function $g_I(x)$, and (P-time computable) neighborhood function, $\mathcal{N}_I(x)$.
There are many **PLS**-complete problems. One is: given a boolean circuit, C, with n input gates and m output gates, compute $x^* \in \{0,1\}^n$ such that $integer(C(x^*)) \geq integer(C(x'))$ for all $x' \in \{0,1\}^n$ whose hamming distance from x^* is 1.

Towards the complexity of Tarski: some standard total search complexity classes.

Two very well-studied discrete total search complexity classes are:

- **PLS** (*Polynomial Local Search*) [JPY'88]
- **PPAD** (*Polynomial Parity Argument – Directed*) [P'94]

PLS consists of discrete local search problems that can be phrased as follows: given a instance $I \in \{0,1\}^*$, and a start "solution" $x \in \{0,1\}^{p(|I|)}$, compute a "locally optimal" solution, $x^* \in \{0,1\}^{p(|I|)}$, with respect to a (P-time computable) objective function $g_I(x)$, and (P-time computable) neighborhood function, $\mathcal{N}_I(x)$.

There are many **PLS**-complete problems. One is: given a boolean circuit, C, with n input gates and m output gates, compute $x^* \in \{0,1\}^n$ such that $\mathit{integer}(C(x^*)) \geq \mathit{integer}(C(x'))$ for all $x' \in \{0,1\}^n$ whose hamming distance from x^* is 1.

PPAD can be defined in many ways. One **PPAD**-complete problem ([Chen-Deng'06]) is this: compute a mixed Nash Equilibrium for a given 2-player normal form (bimatrix) game.

$\texttt{Tarski} \in$ **PLS** $\cap$ **PPAD**

Theorem

$\texttt{Tarski} \in$ **PLS** $\cap$ **PPAD**

Proof sketch that $\texttt{Tarski}$ is in **PLS**: given an instance C_f of $\texttt{Tarski}$, consider the set of "solutions" to be $S_f = \{x \in [N]^d \mid x \leq f(x)\}$, and define the objective function to be $g_f(x) := \sum_{i=1}^{d} x_i$, and the neighborhood function to be $\mathcal{N}_f(x) := \{f(x)\}$. It is not hard to show that x^* is a local optimum iff $x^* \in \mathrm{Fix}(f)$.

`Tarski` $\in$ **PLS** $\cap$ **PPAD**

Theorem

`Tarski` $\in$ **PLS** $\cap$ **PPAD**

Proof sketch that `Tarski` is in **PLS**: given an instance C_f of `Tarski`, consider the set of "solutions" to be $S_f = \{x \in [N]^d \mid x \leq f(x)\}$, and define the objective function to be $g_f(x) := \sum_{i=1}^{d} x_i$, and the neighborhood function to be $\mathcal{N}_f(x) := \{f(x)\}$. It is not hard to show that x^* is a local optimum iff $x^* \in \mathrm{Fix}(f)$.

The **proof** that `Tarski` is in **PPAD** is more involved. It uses: (1.) a characterization of **PPAD** from [E.-Yannakakis'07], (2.) a special simplicial subdivision of the d-cube, $[0,1]^d$ ([Freudenthal,1942]), (3.) a divide-and-conquer algorithm, to show that `Tarski` $\in P^{\mathbf{PPAD}}$, and (4.) the fact ([Buss-Johnson,2012]) that **PPAD** is closed under P-time Turing reductions.

Is `Tarski` "hard"?

Note that no search problem in **PLS** or in **PPAD** can be **NP**-hard, unless **NP** $=$ **co-NP**.

Also, since `Tarski` $\in$ **PLS** $\cap$ **PPAD**, it cannot be **PLS**-complete (nor **PPAD**-complete), unless **PLS** $\subseteq$ **PPAD** (or **PPAD** $\subseteq$ **PLS**, respectively). Neither of these two inclusions is known, nor widely believed.

Is `Tarski` "hard"?

Note that no search problem in **PLS** or in **PPAD** can be **NP**-hard, unless **NP** $=$ **co-NP**.

Also, since `Tarski` $\in$ **PLS** $\cap$ **PPAD**, it cannot be **PLS**-complete (nor **PPAD**-complete), unless **PLS** $\subseteq$ **PPAD** (or **PPAD** $\subseteq$ **PLS**, respectively). Neither of these two inclusions is known, nor widely believed.

However, we can provide some tentative "evidence" that `Tarski` is "somewhat hard"....

Condon's and Shapley's stochastic games reduce to `Tarski`

Theorem

The following problems are P-time reducible to `Tarski`:

- Given an instance, G, of Condon's simple stochastic (reachability) game (SSG), compute the exact value $val(G)$ of G.
- Given an instance, G, of Shapley's original stochastic game, and given $\epsilon > 0$ (in binary), compute an approximate value, v', such that $|val(G) - v'| < \epsilon$.

Note: It is a long-standing open problem whether the value of SSGs can be computed in P-time. It is at least as hard as solving *parity games* and *mean payoff games*. Approximating the value of Shapley's games is, in turn, at least as hard as computing the value of a SSG.

Proof sketch that solving SSGs is reducible to `Tarski`:
Given a SSG, $G = (V, V_0, V_1, V_2, \delta)$, with vertices $V = \{v_1, \ldots, v_n\}$, with 0-sink v_{n-1} and 1-sink v_n, consider the following system of n equations in n unknowns:

$$x_i = \begin{cases} \sum_{\{v_j \in V \mid (v_i, p_{v_i,v_j}, v_j) \in \delta\}} p_{v_i,v_j} x_j & \text{if } v_i \in V_0 \\ \max\{x_j \mid (v_i, \bot, v_j) \in \delta\} & \text{if } v_i \in V_1 \\ \min\{x_j \mid (v_i, \bot, v_j) \in \delta\} & \text{if } v_i \in V_2 \\ 0 & \text{if } v_i = v_{n-1} \text{ is the } \mathbf{0}\text{-sink} \\ 1 & \text{if } v_i = v_n \text{ is the } \mathbf{1}\text{-sink} \end{cases}$$

- Denote these equations by $x = F(x)$. $F : [0,1]^n \to [0,1]^n$ defines a monotone (continuous) map.
- The n-vector of *values* of the SSG, starting at each vertex v_i, is given by the LFP solution, $q^* \in [0,1]^n$, of $x = F(x)$.
- For $\beta > 0$, the β-discounted equations $x = (1 - \beta)F(x)$ are also monotone, and also a *contraction* map. Hence (by Banach's fixed point theorem) they have a unique fixed point $q^\beta \in [0,1]^n$.

- It is possible to choose a small enough $\beta > 0$, such that one can recover the LFP, q^*, of $x = F(x)$, from the unique fixed point q^β of $x = (1 - \beta)F(x)$.
- Finally, we can define a discretized monotone function $H : [M]^n \to [M]^n$, a discretization of $(1 - \beta)F(x)$, such that a fixed point of H yields a point ϵ-close to the unique fixed point q^β, for a chosen $\epsilon > 0$, from which we can uniquely recover q^β, and in turn uniquely recover q^*.

`Tarski` is P-time equivalent to computing a pure NE for a supermodular game

Supermodular games ([Topkis,1979]), and *games with strategic complementarities* ([Milgrom-Roberts,1990]), are important classes of games with widespread applications in economics (for modeling oligopolies, markets with search costs, bank runs, arms races, ...).

`Tarski` is P-time equivalent to computing a pure NE for a supermodular game

Supermodular games ([Topkis,1979]), and *games with strategic complementarities* ([Milgrom-Roberts,1990]), are important classes of games with widespread applications in economics (for modeling oligopolies, markets with search costs, bank runs, arms races, ...).

These games always have a **pure** Nash Equilibrium ([Topkis'79]). The proof of existence of a pure NE uses the fact that their (infimum) "best response correspondence" defines a *monotone function*, and applies Tarski's fixed point theorem.

`Tarski` is P-time equivalent to computing a pure NE for a supermodular game

Supermodular games ([Topkis,1979]), and *games with strategic complementarities* ([Milgrom-Roberts,1990]), are important classes of games with widespread applications in economics (for modeling oligopolies, markets with search costs, bank runs, arms races, ...).

These games always have a **pure** Nash Equilibrium ([Topkis'79]). The proof of existence of a pure NE uses the fact that their (infimum) "best response correspondence" defines a *monotone function*, and applies Tarski's fixed point theorem. We show that:

Theorem

- Computing a pure NE for a given k-player discrete supermodular game with strategy space $[N]^{d_i}$ for player i, given its (infimum) best response correspondence, is P-time reducible to `Tarski`.
- `Tarski` is P-time reducible to computing a pure NE for a given 2-player discrete supermodular games with strategy space $[N]^d$ for each player.

Definition 1: A function $f : L \to \mathbb{R}$, where L is a lattice, is **supermodular** if $\forall x, y \in L$, $f(x) + f(y) \leq f(x \wedge y) + f(x \vee y)$.

Definition 2: A function $f : L_1 \times L_2 \to \mathbb{R}$ has **increasing differences** in its two arguments if for all $x' \geq x$ in L_1 and $y' \geq y$ in L_2, $f(x', y') - f(x', y) \geq f(x, y') - f(x, y)$.

Definition: In a **supermodular game** with k players, each player $i \in [k]$ has a complete lattice S_i of strategies. Let $S = \prod_{i=1}^{k} S_i$ be the product lattice of pure strategy profiles. Every player's utility function $u_i : S \to \mathbb{R}$ must satisfy the following conditions:

C1. $u_i(s_i; s_{-i})$ is upper semicontinuous in s_i for fixed s_{-i}, and continuous in s_{-i} for fixed s_i, and has a finite upper bound. (This condition holds *trivially* when S_i is a finite subset of $\mathbb{R}^{m_i}$.)

C2. $u_i(s_i; s_{-i})$ is *supermodular* in s_i for fixed s_{-i}.

C3. $u_i(s_i; s_{-i})$ has *increasing differences* in s_i and s_{-i}.

A lower bound for `Tarski` in the oracle model

Theorem

- *Any deterministic black-box (oracle) algorithm for computing a Tarski fixed point of a monotone function* $f : [N]^2 \rightarrow [N]^2$ *requires* $\Omega(\log^2 N)$ *queries.*
- *Any randomized black-box (oracle) algorithm for computing a Tarski fixed point of a monotone function* $f : [N]^2 \rightarrow [N]^2$*, requires* $\Omega(\log^2 N)$ *queries in expectation (and w. h. p.).*

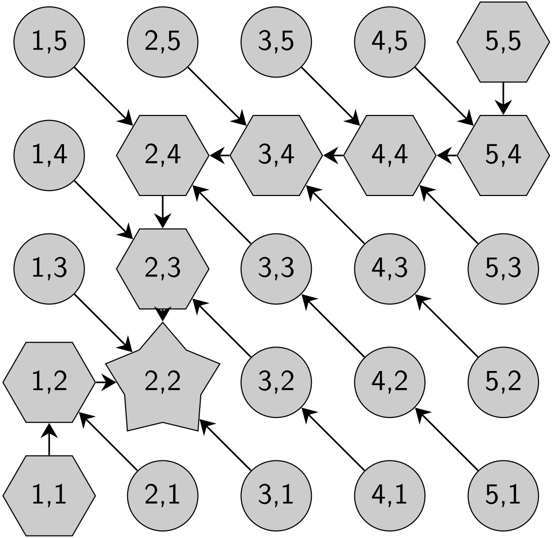
A lower bound for `Tarski` in the oracle model

Theorem

- *Any deterministic black-box (oracle) algorithm for computing a Tarski fixed point of a monotone function* $f : [N]^2 \to [N]^2$ *requires* $\Omega(\log^2 N)$ *queries.*
- *Any randomized black-box (oracle) algorithm for computing a Tarski fixed point of a monotone function* $f : [N]^2 \to [N]^2$*, requires* $\Omega(\log^2 N)$ *queries in expectation (and w. h. p.).*

The lower bound proof uses a family of functions we call "*herringbones*", whose "vector field" looks a bit like a fish bone with a unique fixed point....

An example of a “herringbone” function $f : [5]^2 \to [5]^2$:



Conclusions

- We have shown that $\texttt{Tarski} \in \textbf{PLS} \cap \textbf{PPAD}$.

Conclusions

- We have shown that `Tarski` $\in$ **PLS** $\cap$ **PPAD**.
- We have shown that `Tarski` is at least as hard as solving both simple stochastic games and Shapley's stochastic games; and just as hard as finding a pure NE for supermodular games.

Conclusions

- We have shown that $\texttt{Tarski} \in \textbf{PLS} \cap \textbf{PPAD}$.
- We have shown that $\texttt{Tarski}$ is at least as hard as solving both simple stochastic games and Shapley's stochastic games; and just as hard as finding a pure NE for supermodular games.
- We have shown, in the oracle model, for 2-dimensional monotone functions $f : [N]^2 \to [N]^2$, a lower bound of $\Omega(\log^2 N)$ for the (expected) number of (randomized) queries required to find a fixed point, which matches the $O(\log^2 N)$ upper bound for $d = 2$.

Conclusions

- We have shown that `Tarski` $\in$ **PLS** $\cap$ **PPAD**.
- We have shown that `Tarski` is at least as hard as solving both simple stochastic games and Shapley's stochastic games; and just as hard as finding a pure NE for supermodular games.
- We have shown, in the oracle model, for 2-dimensional monotone functions $f : [N]^2 \to [N]^2$, a lower bound of $\Omega(\log^2 N)$ for the (expected) number of (randomized) queries required to find a fixed point, which matches the $O(\log^2 N)$ upper bound for $d = 2$.
- Can we do much better than $O(\log^d N)$ for <u>small</u> values of $d > 2$?

  We know that for large $d = \omega(\frac{\log N}{\log \log N})$, the $d \cdot N$ upper bound is already better than $\log^d N$.

Conclusions

- We have shown that `Tarski` $\in$ **PLS** $\cap$ **PPAD**.
- We have shown that `Tarski` is at least as hard as solving both simple stochastic games and Shapley's stochastic games; and just as hard as finding a pure NE for supermodular games.
- We have shown, in the oracle model, for 2-dimensional monotone functions $f : [N]^2 \to [N]^2$, a lower bound of $\Omega(\log^2 N)$ for the (expected) number of (randomized) queries required to find a fixed point, which matches the $O(\log^2 N)$ upper bound for $d = 2$.
- Can we do much better than $O(\log^d N)$ for <u>small</u> values of $d > 2$?
  We know that for large $d = \omega(\frac{\log N}{\log \log N})$, the $d \cdot N$ upper bound is already better than $\log^d N$.
- Very recent result by [Fearnley-Savani, 2021]:
  For $d \geq 3$, they give a $O(\log^{d-1} N)$ query algorithm for finding a fixed point of a monotone function $f : [N]^d \to [N]^d$.

Conclusions

- We have shown that `Tarski` $\in$ **PLS** $\cap$ **PPAD**.
- We have shown that `Tarski` is at least as hard as solving both simple stochastic games and Shapley's stochastic games; and just as hard as finding a pure NE for supermodular games.
- We have shown, in the oracle model, for 2-dimensional monotone functions $f : [N]^2 \rightarrow [N]^2$, a lower bound of $\Omega(\log^2 N)$ for the (expected) number of (randomized) queries required to find a fixed point, which matches the $O(\log^2 N)$ upper bound for $d = 2$.
- Can we do much better than $O(\log^d N)$ for <u>small</u> values of $d > 2$?
  We know that for large $d = \omega(\frac{\log N}{\log \log N})$, the $d \cdot N$ upper bound is already better than $\log^d N$.
- Very recent result by [Fearnley-Savani, 2021]:
  For $d \geq 3$, they give a $O(\log^{d-1} N)$ query algorithm for finding a fixed point of a monotone function $f : [N]^d \rightarrow [N]^d$.
- Many, many, questions remain open.