Learning unary automata

Gregor Gramlich   Ralf Herrmann

Institut für Informatik
Johann Wolfgang Goethe–Universität
Frankfurt am Main

30-June-2005, Descriptional Complexity of Formal Systems

Outline

1. Introduction
   - Unary Regular Languages
   - Algorithmic Learning Theory
2. Consistency Problems and PAC Learning
   - Minimum Consistent DFA
   - Minimum Consistent NFA
   - PAC Learning and VC Dimension
3. Learning with Equivalence Queries

Outline

Basics on Unary DFAs.

- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:

- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

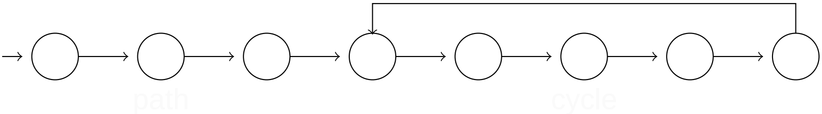
Basics on Unary DFAs.

- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:

- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basics on Unary DFAs.

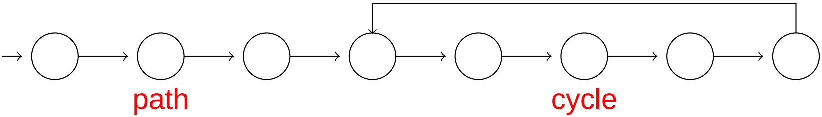
- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:



- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basics on Unary DFAs.

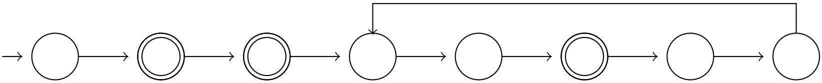
- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:



- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basics on Unary DFAs.

- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:

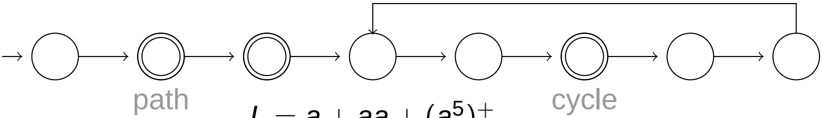


$$L = a + aa + (a^5)^+$$

- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basics on Unary DFAs.

- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:

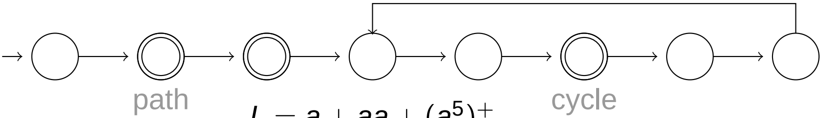


$$L = a + aa + (a^5)^+$$

- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basics on Unary DFAs.

- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:

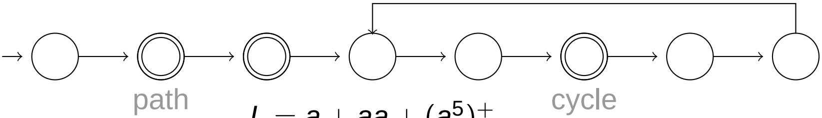


$$L = a + aa + (a^5)^+$$

- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basics on Unary DFAs.

- A unary language is defined over $\Sigma = \{a\}$.
- A unary regular language is represented by a DFA:



$$L = a + aa + (a^5)^+$$

- Ultimate period of a regular language = cycle length.
- Minimum ultimate period = minimum cycle length.
- A unary DFA that consists only of a cycle is called a cyclic DFA.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Basic Notions in Algorithmic Learning Theory

- Concept $c \subseteq X$.
- Concept class $\mathcal{C} \subseteq \mathcal{P}(X)$, concept $c \in \mathcal{C}$.
- Example $x \in X$ is a positive example (for c), if $x \in c$ and a negative example, if $x \notin c$.
- Common problem: Given a set of classified examples, give a good hypothesis for the concept.

In our context:

- Concept class: class of unary regular languages representable by automata of a certain size.
- Concept: language from the concept class.
- Universe X: set of words $\{a\}^*$.
- Example: word.

Problems Considered in Learning Theory

- Consistency problem (minimum size of a consistent hypothesis)
- PAC learning
- Learning with equivalence queries

Problems Considered in Learning Theory

- Consistency problem (minimum size of a consistent hypothesis)
- PAC learning (hypothesis that is correct on most examples with high probability)
- Learning with equivalence queries

Problems Considered in Learning Theory

- Consistency problem (minimum size of a consistent hypothesis)
- PAC learning (hypothesis that is correct on most examples with high probability)
- Learning with equivalence queries (every wrong hypothesis is answered with a counterexample)

Outline

1. Introduction
   - Unary Regular Languages
   - Algorithmic Learning Theory
2. Consistency Problems and PAC Learning
   - Minimum Consistent DFA
   - Minimum Consistent NFA
   - PAC Learning and VC Dimension
3. Learning with Equivalence Queries

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.
- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
- We show:

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.

- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)

- We show:

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.

- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
  Even polynomial approximations are NP-complete. (Pitt & Warmuth, 1993)
  Any “reasonable” efficient approximation is impossible under cryptographic assumptions. (Gramlich & Schnitger, 2005)

- We show:

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.
- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
  Even polynomial approximations are NP-complete. (Pitt & Warmuth, 1993)
  Any “reasonable” efficient approximation is impossible under cryptographic assumptions. (Gramlich & Schnitger, 2005)
- We show:

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.

- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
  Even polynomial approximations are NP-complete. (Pitt & Warmuth, 1993)
  Any "reasonable" efficient approximation is impossible under cryptographic assumptions. (Gramlich & Schnitger, 2005)
- We show:

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.

- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
  Even polynomial approximations are NP-complete. (Pitt & Warmuth, 1993)
  Any "reasonable" efficient approximation is impossible under cryptographic assumptions. (Gramlich & Schnitger, 2005)
- We show:

Consistency problem for unary example sets

The unary minimum consistent DFA problem is efficiently solvable.

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.

- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
  Even polynomial approximations are NP-complete. (Pitt & Warmuth, 1993)
  Any “reasonable” efficient approximation is impossible under cryptographic assumptions. (Gramlich & Schnitger, 2005)
- We show:

Consistency problem for unary example sets

The unary minimum consistent DFA problem is efficiently solvable.

Minimum Consistent DFA

- Input: example sets $P, N \subseteq \Sigma^*$.
- Output: size of a minimum DFA, consistent with P and N.

- Known:
  NP-complete for $|\Sigma| \geq 2$. (Gold, 1978)
  Even polynomial approximations are NP-complete. (Pitt & Warmuth, 1993)
  Any "reasonable" efficient approximation is impossible under cryptographic assumptions. (Gramlich & Schnitger, 2005)
- We show:

Consistency problem for unary example sets

The unary minimum consistent DFA problem is efficiently solvable.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = $1+\max\{\min(x, y) \mid x, y$ collide modulo $z\}$.
  - Optimum size = $z+$ optimum path length.
- Compute the size of a minimum consistent DFA.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = $1+\max\{\min(x, y) | \ x, y$ collide modulo $z\}$.
  - Optimum size = $z+$ optimum path length.
- Compute the size of a minimum consistent DFA.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = $1+\max\{\min(x, y) | \, x, y$ collide modulo $z\}$.
  - Optimum size = $z+$ optimum path length.
- Compute the size of a minimum consistent DFA.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = 1+max$\{\min(x, y) \mid x, y \text{ collide modulo } z\}$.
  - Optimum size = z+ optimum path length.
- Compute the size of a minimum consistent DFA.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = 1+max$\{\min(x, y) | \; x, y$ collide modulo $z\}$.
  - Optimum size = z+ optimum path length.
- Compute the size of a minimum consistent DFA.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = $1+\max\{\min(x, y) \mid x, y$ collide modulo $z\}$.
  - Optimum size = $z+$ optimum path length.
- Compute the size of a minimum consistent DFA.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = $1+\max\{\min(x, y) \mid x, y$ collide modulo $z\}$.
  - Optimum size = $z+$ optimum path length.
- Compute the size of a minimum consistent DFA.
  - Compute the optimum DFA size for cycle lengths $z = 1, 2, \ldots$.
  - Stop, if we cannot improve any more, because the cycle length is larger than the best size found so far.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = 1+max$\{\min(x, y) \mid x, y$ collide modulo $z\}$.
  - Optimum size = z+ optimum path length.
- Compute the size of a minimum consistent DFA.
  - Compute the optimum DFA size for cycle lengths $z = 1, 2, \ldots$.
  - Stop, if we cannot improve any more, because the cycle length is larger than the best size found so far.

Minimum Consistent DFA

- Easy, if we fix cycle length z first.
  - We say example lengths x, y collide modulo z, if $x \equiv y \pmod{z}$ and $a^x \in P$ and $a^y \in N$.
  - If x, y collide modulo z, at least one of them must not reach the cycle in a consistent DFA with cycle length z.
  - Optimum path length = 1+max$\{\min(x, y) | \, x, y$ collide modulo $z\}$.
  - Optimum size = z+ optimum path length.
- Compute the size of a minimum consistent DFA.
  - Compute the optimum DFA size for cycle lengths $z = 1, 2, \ldots$.
  - Stop, if we cannot improve any more, because the cycle length is larger than the best size found so far.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$ for ℓ large enough.
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$ for ℓ large enough.
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$ for ℓ large enough.
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Efficiency of the Algorithm

- Number of iterations = optimum size.
- Obvious: optimum size $\leq$ |longest example| + 1.
- Representing inputs: example lengths are coded in binary, input length $\approx \ell = \sum_{x \in P \cup N} \log x$.
- Optimum size $\leq \ell^3$ for ℓ large enough.
- Linear time for each iteration (fixed cycle length).
- Total time $O(\ell^4)$.

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any “reasonable” approximation is intractable.
- We show:

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any “reasonable” approximation is intractable.
- We show:

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any “reasonable” approximation is intractable.
- We show:

Intractability

Unary problem is not in P, unless $NP \subseteq DTIME(n^{O(\log n)})$.

Approximability

A unary NFA with $O(opt^2)$ states can be constructed efficiently.

Only in the article.

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any "reasonable" approximation is intractable.
- We show:

Intractability

Unary problem is not in *P*, unless $NP \subseteq \mathrm{DTIME}(n^{O(\log n)})$.

Approximability

A unary NFA with $O(opt^2)$ states can be constructed efficiently.

Only in the article.

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any "reasonable" approximation is intractable.
- We show:

Intractability

Unary problem is not in *P*, unless $NP \subseteq \mathrm{DTIME}(n^{O(\log n)})$.

Approximability

A unary NFA with $O(opt^2)$ states can be constructed efficiently.

Only in the article.

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any "reasonable" approximation is intractable.
- We show:

Intractability

Unary problem is not in *P*, unless $NP \subseteq \mathrm{DTIME}(n^{O(\log n)})$.

Approximability

A unary NFA with $O(\mathrm{opt}^2)$ states can be constructed efficiently.

Only in the article.

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any "reasonable" approximation is intractable.
- We show:

Intractability

Unary problem is not in *P*, unless $NP \subseteq \mathrm{DTIME}(n^{O(\log n)})$.

Approximability

A unary NFA with $O(\mathrm{opt}^2)$ states can be constructed efficiently.

Only in the article.

Minimum Consistent NFA

- Known:
  NP-complete for $|\Sigma| \geq 2$, even any "reasonable" approximation is intractable.
- We show:

Intractability

Unary problem is not in *P*, unless $NP \subseteq \mathrm{DTIME}(n^{O(\log n)})$.

Approximability

A unary NFA with $O(\mathrm{opt}^2)$ states can be constructed efficiently.

Only in the article.

PAC Learning

A is a PAC (probably approximately correct) algorithm, if

- A determines how many examples it needs in dependence on input parameters $0 < \delta < 1/2$ and $0 < \epsilon < 1/2$.
- Classified examples are chosen randomly under some distribution $\mathcal{D}$ and presented to A.
- With probability $\geq 1 - \delta$, A computes a hypothesis that classifies only an ϵ-portion (measured under $\mathcal{D}$) of examples incorrectly.

Relation to the consistency problem:

- An algorithm that solves the consistency problem for $\mathcal{C}$ on $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{\mathrm{VC}(\mathcal{C})}{\epsilon}\log\frac{1}{\epsilon})$ examples is a PAC algorithm.

PAC Learning

A is a PAC (probably approximately correct) algorithm, if

- *A* determines how many examples it needs in dependence on input parameters $0 < \delta < 1/2$ and $0 < \epsilon < 1/2$.
- Classified examples are chosen randomly under some distribution $\mathcal{D}$ and presented to *A*.
- With probability $\geq 1 - \delta$, *A* computes a hypothesis that classifies only an ϵ-portion (measured under $\mathcal{D}$) of examples incorrectly.

Relation to the consistency problem:

- An algorithm that solves the consistency problem for $\mathcal{C}$ on $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{\mathrm{VC}(\mathcal{C})}{\epsilon}\log\frac{1}{\epsilon})$ examples is a PAC algorithm.

PAC Learning

A is a PAC (probably approximately correct) algorithm, if

- *A* determines how many examples it needs in dependence on input parameters $0 < \delta < 1/2$ and $0 < \epsilon < 1/2$.
- Classified examples are chosen randomly under some distribution $\mathcal{D}$ and presented to *A*.
- With probability $\geq 1 - \delta$, *A* computes a hypothesis that classifies only an ϵ-portion (measured under $\mathcal{D}$) of examples incorrectly.

Relation to the consistency problem:

- An algorithm that solves the consistency problem for $\mathcal{C}$ on $\Theta(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{\mathrm{VC}(\mathcal{C})}{\epsilon} \log \frac{1}{\epsilon})$ examples is a PAC algorithm.

PAC Learning

A is a PAC (probably approximately correct) algorithm, if

- *A* determines how many examples it needs in dependence on input parameters $0 < \delta < 1/2$ and $0 < \epsilon < 1/2$.
- Classified examples are chosen randomly under some distribution $\mathcal{D}$ and presented to *A*.
- With probability $\geq 1 - \delta$, *A* computes a hypothesis that classifies only an ϵ-portion (measured under $\mathcal{D}$) of examples incorrectly.

Relation to the consistency problem:

- An algorithm that solves the consistency problem for $\mathcal{C}$ on $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{\mathrm{VC}(\mathcal{C})}{\epsilon}\log\frac{1}{\epsilon})$ examples is a PAC algorithm.

PAC Learning

A is a PAC (probably approximately correct) algorithm, if

- *A* determines how many examples it needs in dependence on input parameters $0 < \delta < 1/2$ and $0 < \epsilon < 1/2$.
- Classified examples are chosen randomly under some distribution $\mathcal{D}$ and presented to *A*.
- With probability $\geq 1 - \delta$, *A* computes a hypothesis that classifies only an ϵ-portion (measured under $\mathcal{D}$) of examples incorrectly.

Relation to the consistency problem:

- An algorithm that solves the consistency problem for $\mathcal{C}$ on $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{\mathrm{VC}(\mathcal{C})}{\epsilon}\log\frac{1}{\epsilon})$ examples is a PAC algorithm.

PAC Learning

A is a PAC (probably approximately correct) algorithm, if

- A determines how many examples it needs in dependence on input parameters $0 < \delta < 1/2$ and $0 < \epsilon < 1/2$.
- Classified examples are chosen randomly under some distribution $\mathcal{D}$ and presented to A.
- With probability $\geq 1 - \delta$, A computes a hypothesis that classifies only an ϵ-portion (measured under $\mathcal{D}$) of examples incorrectly.

Relation to the consistency problem:

- An algorithm that solves the consistency problem for $\mathcal{C}$ on $\Theta(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{\mathrm{VC}(\mathcal{C})}{\epsilon} \log \frac{1}{\epsilon})$ examples is a PAC algorithm.

VC Dimension of a Class of Unary Languages

VC Dimension of $\mathcal{L}_n$

Let $\mathcal{L}_n$ be the class of unary languages, accepted by DFAs with at most n states and $\pi(n)$ be the number of primes $\leq n$. Then

$$n - 1 + \lfloor \log(\pi(n) + 1) \rfloor \;\leq\; \mathrm{VC}(\mathcal{L}_n) \;\leq\; n + \log(n).$$

- Bounds are almost tight: $\log(\pi(n) + 1) \approx \log n - \log \ln n$.

VC Dimension of a Class of Unary Languages

VC Dimension of $\mathcal{L}_n$

Let $\mathcal{L}_n$ be the class of unary languages, accepted by DFAs with at most n states and $\pi(n)$ be the number of primes $\leq n$. Then

$$n - 1 + \lfloor \log(\pi(n) + 1) \rfloor \ \leq \ \mathrm{VC}(\mathcal{L}_n) \ \leq \ n + \log(n).$$

- Bounds are almost tight: $\log(\pi(n) + 1) \approx \log n - \log \ln n$.

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ be the class of languages, that are accepted by unary DFAs with at most n states.

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ be the class of languages, that are accepted by unary DFAs with at most n states.

- The VC dimension of $\mathcal{L}_n$ is $n + \log n \pm \Theta(\log \log n)$.
- $(\mathcal{L}_n | n \in \mathbb{N})$ is efficiently PAC learnable, because the consistency problem is efficiently solvable and the number of examples needed is bounded by $\Theta(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{n + \log n}{\epsilon} \log \frac{1}{\epsilon})$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is not efficiently PAC learnable, if hypotheses from $\mathcal{N}_n$ are used to learn concepts from $\mathcal{N}_n$ and NP-complete problems are not solvable by Monte-Carlo Turing machines in time $n^{O(\log n)}$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is efficiently PAC learnable, if hypotheses from $\mathcal{N}_{O(n^2)}$ are used to learn concepts from $\mathcal{N}_n$.

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ be the class of languages, that are accepted by unary DFAs with at most n states.

- The VC dimension of $\mathcal{L}_n$ is $n + \log n \pm \Theta(\log \log n)$.
- $(\mathcal{L}_n | n \in \mathbb{N})$ is efficiently PAC learnable, because the consistency problem is efficiently solvable and the number of examples needed is bounded by $\Theta(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{n + \log n}{\epsilon} \log \frac{1}{\epsilon})$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is not efficiently PAC learnable, if hypotheses from $\mathcal{N}_n$ are used to learn concepts from $\mathcal{N}_n$ and NP-complete problems are not solvable by Monte-Carlo Turing machines in time $n^{O(\log n)}$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is efficiently PAC learnable, if hypotheses from $\mathcal{N}_{O(n^2)}$ are used to learn concepts from $\mathcal{N}_n$.

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ ($\mathcal{N}_n$) be the class of languages, that are accepted by unary DFAs (NFAs) with at most n states.

- The VC dimension of $\mathcal{L}_n$ is $n + \log n \pm \Theta(\log\log n)$.
- $(\mathcal{L}_n | n \in \mathbb{N})$ is efficiently PAC learnable, because the consistency problem is efficiently solvable and the number of examples needed is bounded by $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n+\log n}{\epsilon}\log\frac{1}{\epsilon})$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is not efficiently PAC learnable, if hypotheses from $\mathcal{N}_n$ are used to learn concepts from $\mathcal{N}_n$ and NP-complete problems are not solvable by Monte-Carlo Turing machines in time $n^{O(\log n)}$.

  (Consequence of the intractability of the NFA consistency problem.)
- $(\mathcal{N}_n | n \in \mathbb{N})$ is efficiently PAC learnable, if hypotheses from $\mathcal{N}_{O(n^2)}$ are used to learn concepts from $\mathcal{N}_n$.

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ ($\mathcal{N}_n$) be the class of languages, that are accepted by unary DFAs (NFAs) with at most n states.

- The VC dimension of $\mathcal{L}_n$ is $n + \log n \pm \Theta(\log\log n)$.
- $(\mathcal{L}_n | n \in \mathbb{N})$ is efficiently PAC learnable, because the consistency problem is efficiently solvable and the number of examples needed is bounded by $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n+\log n}{\epsilon}\log\frac{1}{\epsilon})$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is not efficiently PAC learnable, if hypotheses from $\mathcal{N}_n$ are used to learn concepts from $\mathcal{N}_n$ and NP-complete problems are not solvable by Monte-Carlo Turing machines in time $n^{O(\log n)}$. (Consequence of the intractability of the NFA consistency problem.)
- $(\mathcal{N}_n | n \in \mathbb{N})$ is efficiently PAC learnable, if hypotheses from $\mathcal{N}_{O(n^2)}$ are used to learn concepts from $\mathcal{N}_n$.

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ ($\mathcal{N}_n$) be the class of languages, that are accepted by unary DFAs (NFAs) with at most n states.

- The VC dimension of $\mathcal{L}_n$ is $n + \log n \pm \Theta(\log\log n)$.
- $(\mathcal{L}_n | n \in \mathbb{N})$ is efficiently PAC learnable, because the consistency problem is efficiently solvable and the number of examples needed is bounded by $\Theta(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n+\log n}{\epsilon}\log\frac{1}{\epsilon})$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is not efficiently PAC learnable, if hypotheses from $\mathcal{N}_n$ are used to learn concepts from $\mathcal{N}_n$ and NP-complete problems are not solvable by Monte-Carlo Turing machines in time $n^{O(\log n)}$. (Consequence of the intractability of the NFA consistency problem.)
- $(\mathcal{N}_n | n \in \mathbb{N})$ is efficiently PAC learnable, if hypotheses from $\mathcal{N}_{O(n^2)}$ are used to learn concepts from $\mathcal{N}_n$.

  (Consequence of the quadratic approximability of the NFA consistency problem and $VC(\mathcal{N}_n) \le n \log n$.)

Results: PAC Learning and VC Dimension

Let $\mathcal{L}_n$ ($\mathcal{N}_n$) be the class of languages, that are accepted by unary DFAs (NFAs) with at most n states.

- The VC dimension of $\mathcal{L}_n$ is $n + \log n \pm \Theta(\log \log n)$.
- $(\mathcal{L}_n | n \in \mathbb{N})$ is efficiently PAC learnable, because the consistency problem is efficiently solvable and the number of examples needed is bounded by $\Theta(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{n+\log n}{\epsilon} \log \frac{1}{\epsilon})$.
- $(\mathcal{N}_n | n \in \mathbb{N})$ is not efficiently PAC learnable, if hypotheses from $\mathcal{N}_n$ are used to learn concepts from $\mathcal{N}_n$ and *NP*-complete problems are not solvable by Monte-Carlo Turing machines in time $n^{O(\log n)}$. (Consequence of the intractability of the NFA consistency problem.)
- $(\mathcal{N}_n | n \in \mathbb{N})$ is efficiently PAC learnable, if hypotheses from $\mathcal{N}_{O(n^2)}$ are used to learn concepts from $\mathcal{N}_n$.
  (Consequence of the quadratic approximability of the NFA consistency problem and $\mathrm{VC}(\mathcal{N}_n) \leq n \log n$.)

Outline

Learning with Equivalence Queries

- Learning algorithm submits a hypothesis from $\mathcal{H}$ as a query to the oracle.
- Oracle compares the hypothesis with the concept c from $\mathcal{C}$ and
- How many counterexamples are sufficient and necessary to learn the concept?

Learning with Equivalence Queries

- Learning algorithm submits a hypothesis from $\mathcal{H}$ as a query to the oracle.
- Oracle compares the hypothesis with the concept c from $\mathcal{C}$ and
  - confirms equivalence or
  - returns counterexample.
- How many counterexamples are sufficient and necessary to learn the concept?

Learning with Equivalence Queries

- Learning algorithm submits a hypothesis from $\mathcal{H}$ as a query to the oracle.
- Oracle compares the hypothesis with the concept c from $\mathcal{C}$ and
  - confirms equivalence or
  - returns counterexample.
- How many counterexamples are sufficient and necessary to learn the concept?

Learning with Equivalence Queries

- Learning algorithm submits a hypothesis from $\mathcal{H}$ as a query to the oracle.
- Oracle compares the hypothesis with the concept c from $\mathcal{C}$ and
  - confirms equivalence or
  - returns counterexample.
- How many counterexamples are sufficient and necessary to learn the concept?

Learning with Equivalence Queries

- Learning algorithm submits a hypothesis from $\mathcal{H}$ as a query to the oracle.
- Oracle compares the hypothesis with the concept c from $\mathcal{C}$ and
  - confirms equivalence or
  - returns counterexample.
- How many counterexamples are sufficient and necessary to learn the concept?

Learning with Equivalence Queries

- Learning algorithm submits a hypothesis from $\mathcal{H}$ as a query to the oracle.
- Oracle compares the hypothesis with the concept c from $\mathcal{C}$ and
  - confirms equivalence or
  - returns counterexample.
- How many counterexamples are sufficient and necessary to learn the concept?

Angluin, 1990

Non-unary DFAs and NFAs are not learnable from equivalence queries with polynomially many counterexamples and hypotheses of polynomial size.

Learning Cyclic DFAs with Equivalence Queries

Let $\mathcal{C}_n = \mathcal{H}_n$ be the class of languages, that are accepted by a cyclic unary DFA with prime cycle length $\leq n$.

Learning prime cycles with equivalence queries

- There is a learning algorithm that needs at most $O(\frac{n^2}{\ln n})$ counterexamples to learn an arbitrary concept from $\mathcal{C}_n$.
- There is an oracle, such that any learning algorithm needs at least $\Omega(\frac{n^2}{\ln n})$ counterexamples.

Using larger hypotheses

If we allow unary cyclic DFAs with at most n^d $(d \leq n)$ states as hypotheses to learn concepts from $\mathcal{C}_n$, then the upper bound is $O(\frac{n^2}{d})$, whereas the lower bound is $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$.

Learning Cyclic DFAs with Equivalence Queries

Let $\mathcal{C}_n = \mathcal{H}_n$ be the class of languages, that are accepted by a cyclic unary DFA with prime cycle length $\leq n$.

Learning prime cycles with equivalence queries

- There is a learning algorithm that needs at most $O(\frac{n^2}{\ln n})$ counterexamples to learn an arbitrary concept from $\mathcal{C}_n$.
- There is an oracle, such that any learning algorithm needs at least $\Omega(\frac{n^2}{\ln n})$ counterexamples.

Using larger hypotheses

If we allow unary cyclic DFAs with at most n^d $(d \leq n)$ states as hypotheses to learn concepts from $\mathcal{C}_n$, then the upper bound is $O(\frac{n^2}{d})$ whereas the lower bound is $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$.

Learning Cyclic DFAs with Equivalence Queries

Let $\mathcal{C}_n = \mathcal{H}_n$ be the class of languages, that are accepted by a cyclic unary DFA with prime cycle length $\leq n$.

Learning prime cycles with equivalence queries

- There is a learning algorithm that needs at most $O(\frac{n^2}{\ln n})$ counterexamples to learn an arbitrary concept from $\mathcal{C}_n$.
- There is an oracle, such that any learning algorithm needs at least $\Omega(\frac{n^2}{\ln n})$ counterexamples.

Using larger hypotheses

If we allow unary cyclic DFAs with at most n^d ($d \leq n$) states as hypotheses to learn concepts from $\mathcal{C}_n$, then the upper bound is $O(\frac{n^2}{d})$, whereas the lower bound is $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$.

Learning Cyclic DFAs with Equivalence Queries

Let $\mathcal{C}_n = \mathcal{H}_n$ be the class of languages, that are accepted by a cyclic unary DFA with prime cycle length $\leq n$.

Learning prime cycles with equivalence queries

- There is a learning algorithm that needs at most $O(\frac{n^2}{\ln n})$ counterexamples to learn an arbitrary concept from $\mathcal{C}_n$.
- There is an oracle, such that any learning algorithm needs at least $\Omega(\frac{n^2}{\ln n})$ counterexamples.

Using larger hypotheses

If we allow unary cyclic DFAs with at most n^d $(d \leq n)$ states as hypotheses to learn concepts from $\mathcal{C}_n$, then the upper bound is $O(\frac{n^2}{d})$, whereas the lower bound is $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$.

Learning Cyclic DFAs with Equivalence Queries

Let $\mathcal{C}_n = \mathcal{H}_n$ be the class of languages, that are accepted by a cyclic unary DFA with prime cycle length $\leq n$.

Learning prime cycles with equivalence queries

- There is a learning algorithm that needs at most $O(\frac{n^2}{\ln n})$ counterexamples to learn an arbitrary concept from $\mathcal{C}_n$.
- There is an oracle, such that any learning algorithm needs at least $\Omega(\frac{n^2}{\ln n})$ counterexamples.

Using larger hypotheses

If we allow unary cyclic DFAs with at most n^d ($d \leq n$) states as hypotheses to learn concepts from $\mathcal{C}_n$, then the upper bound is $O(\frac{n^2}{d})$, whereas the lower bound is $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$.

Learning Cyclic DFAs with Equivalence Queries

Let $\mathcal{C}_n = \mathcal{H}_n$ be the class of languages, that are accepted by a cyclic unary DFA with prime cycle length $\leq n$.

Learning prime cycles with equivalence queries

- There is a learning algorithm that needs at most $O(\frac{n^2}{\ln n})$ counterexamples to learn an arbitrary concept from $\mathcal{C}_n$.
- There is an oracle, such that any learning algorithm needs at least $\Omega(\frac{n^2}{\ln n})$ counterexamples.

Using larger hypotheses

If we allow unary cyclic DFAs with at most n^d $(d \leq n)$ states as hypotheses to learn concepts from $\mathcal{C}_n$, then the upper bound is $O(\frac{n^2}{d})$, whereas the lower bound is $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$.

A Learning Algorithm

- Produce hypotheses with prime cycle length p consistent with previous counterexamples.
- Colliding examples modulo p indicate that cycle length p is impossible for the concept.
- At most p counterexamples for each prime p until collision.
- The algorithm needs at most

$$\sum_{p \leq n, p \text{ prime}} p = \Theta\left(\frac{n^2}{\ln n}\right).$$

counterexamples to learn a cyclic DFA with unknown prime cycle length $\leq n$.

A Learning Algorithm

- Produce hypotheses with prime cycle length p consistent with previous counterexamples.
- Colliding examples modulo p indicate that cycle length p is impossible for the concept.
- At most p counterexamples for each prime p until collision.
- The algorithm needs at most

$$\sum_{p \leq n, p \text{ prime}} p = \Theta\left(\frac{n^2}{\ln n}\right)$$

counterexamples to learn a cyclic DFA with unknown prime cycle length $\leq n$.

A Learning Algorithm

- Produce hypotheses with prime cycle length p consistent with previous counterexamples.
- Colliding examples modulo p indicate that cycle length p is impossible for the concept.
- At most p counterexamples for each prime p until collision.
- The algorithm needs at most

$$\sum_{p \leq n, p \text{ prime}} p = \Theta\left(\frac{n^2}{\ln n}\right)$$

counterexamples to learn a cyclic DFA with unknown prime cycle length $\leq n$.

A Learning Algorithm

- Produce hypotheses with prime cycle length p consistent with previous counterexamples.
- Colliding examples modulo p indicate that cycle length p is impossible for the concept.
- At most p counterexamples for each prime p until collision.
- The algorithm needs at most

$$\sum_{p \leq n, p \text{ prime}} p = \Theta\left(\frac{n^2}{\ln n}\right).$$

counterexamples to learn a cyclic DFA with unknown prime cycle length $\leq n$.

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.
- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \le n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

▸ Skip Example

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.

- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.

- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.

- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.

- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.

- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

A Malicious Oracle

- Oracle acts like someone who is cheating in the game of battle ships.

- The concept is not fixed, but constructed depending on the queries.
- Just make sure, the concept is consistent with the counterexamples.
- Every counterexample reveals as little information as possible.
- Number of counterexamples $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$.

Skip Example

Example for the Malicious Oracle

Hypothesis

Examples

	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis

Examples

	2	3	5	7
0	0	0	0	0

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis

Examples

	2	3	5	7
0	0	0	0	0

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis

Examples

	2	3	5	7
1	1	1	1	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis

Examples

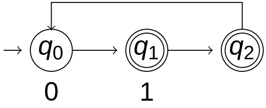
	2	3	5	7
1	1	1	1	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

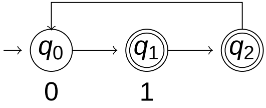
	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

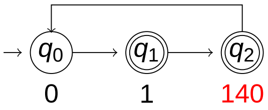
	2	3	5	7
140	0	2	0	0

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

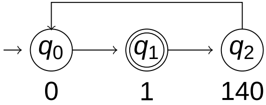
	2	3	5	7
140	0	2	0	0

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

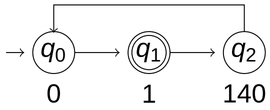
	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

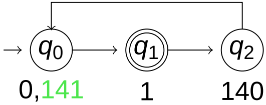
	2	3	5	7
141	1	0	1	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

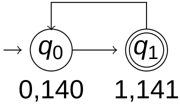
	2	3	5	7
141	1	0	1	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

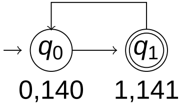
	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

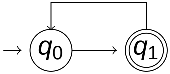
	2	3	5	7
106	0	1	1	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



0,140, 1,141

106

Examples

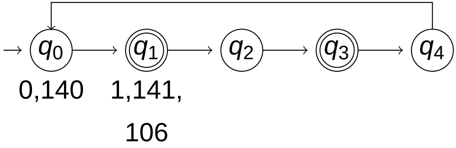
	2	3	5	7
106	0	1	1	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

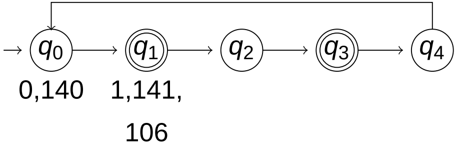
	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

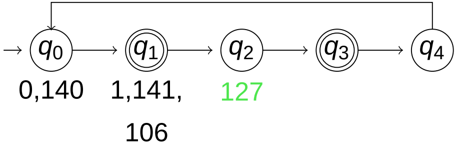
	2	3	5	7
127	1	1	2	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

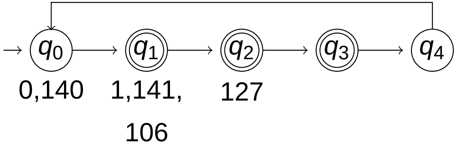
	2	3	5	7
127	1	1	2	1

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

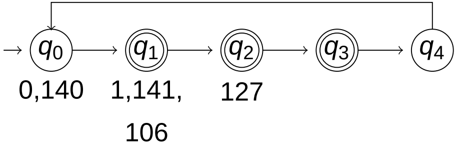
	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

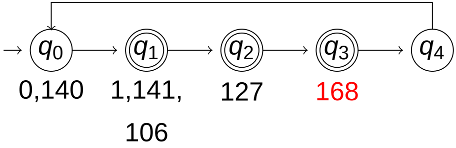
	2	3	5	7
168	0	0	3	0

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



Examples

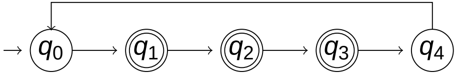
	2	3	5	7
168	0	0	3	0

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Example for the Malicious Oracle

Hypothesis



0,140 1,141, 106 127

...
At least $\sum_{p \leq n}(p-2) = \Omega\left(\frac{n^2}{\ln n}\right)$ counterexamples.

Examples

	2	3	5	7

List of residues

2	3	5	7
0	0	0	0
1	1	1	1
	2	2	2
		3	3
		4	4
			5
			6

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log\log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log \log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log \log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log \log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log \log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log \log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log \log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (I)

Consistency and PAC learning for DFAs

- The consistency problem for unary DFAs becomes simple.
- The VC dimension of unary DFAs with $\leq n$ states is $n + \log n \pm \Theta(\log\log n)$.
- Unary DFAs are efficiently PAC learnable.

Consistency and PAC learning for NFAs

- The consistency problem remains hard for unary NFAs, but can efficiently be approximated quadratically.
- It is hard to PAC learn unary NFAs with small hypotheses.
- Unary NFAs are efficiently PAC learnable with quadratically larger hypotheses.

Summary of Our Contributions (II)

Learning cyclic DFAs by equivalence queries

- Learn DFAs with $p \leq n$ states by hypotheses with $p \leq n$ states: $\Theta\left(\frac{n^2}{\ln n}\right)$ counterexamples are sufficient and necessary.
- Learn DFAs with at most n states by hypotheses with at most n^d states ($d \leq n$): $O(\frac{n^2}{d})$ counterexamples are sufficient and $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$ are necessary.

Summary of Our Contributions (II)

Learning cyclic DFAs by equivalence queries

- Learn DFAs with $p \leq n$ states by hypotheses with $p \leq n$ states: $\Theta\left(\frac{n^2}{\ln n}\right)$ counterexamples are sufficient and necessary.
- Learn DFAs with at most n states by hypotheses with at most n^d states ($d \leq n$): $O(\frac{n^2}{d})$ counterexamples are sufficient and $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$ are necessary.

Summary of Our Contributions (II)

Learning cyclic DFAs by equivalence queries

- Learn DFAs with $p \leq n$ states by hypotheses with $p \leq n$ states: $\Theta\left(\frac{n^2}{\ln n}\right)$ counterexamples are sufficient and necessary.
- Learn DFAs with at most n states by hypotheses with at most n^d states ($d \leq n$): $O(\frac{n^2}{d})$ counterexamples are sufficient and $\Omega(\frac{n^2}{d} \cdot \frac{\ln d}{(\ln n)^2})$ are necessary.

Further Interests, Open Questions

- Learning non-cyclic unary DFAs with equivalence queries.
- Learning unary NFAs with equivalence queries.
- Learning unary PFAs for fixed isolation.

Further Interests, Open Questions

- Learning non-cyclic unary DFAs with equivalence queries.
- Learning unary NFAs with equivalence queries.
- Learning unary PFAs for fixed isolation.

Further Interests, Open Questions

- Learning non-cyclic unary DFAs with equivalence queries.
- Learning unary NFAs with equivalence queries.
- Learning unary PFAs for fixed isolation. (Consistency, PAC, equivalence queries.)

Further Interests, Open Questions

- Learning non-cyclic unary DFAs with equivalence queries.
- Learning unary NFAs with equivalence queries.
- Learning unary PFAs for fixed isolation. (Consistency, PAC, equivalence queries.)

Further Interests, Open Questions

- Learning non-cyclic unary DFAs with equivalence queries.
- Learning unary NFAs with equivalence queries.
- Learning unary PFAs for fixed isolation. (Consistency, PAC, equivalence queries.)

Thanks for the attention.