CMPSC 465
Data Structures and Algorithms
Spring 2022

Instructor: Chunhao Wang

Greedy algorithms

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
    1.1 Find a set A of tasks that are early

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
    1.1 Find a set A of tasks that are early
    1.2 Sort the tasks of A in increasing deadlines

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
    1.1 Find a set A of tasks that are early
    1.2 Sort the tasks of A in increasing deadlines
    1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

$\mathcal{I} = \{A \subseteq S : \exists \text{ a way to schedule the tasks in } A \text{ s.t. no task is late}\}$

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

$\mathcal{I} = \{A \subseteq S : \exists$ a way to schedule the tasks in A s.t. no task is late$\}$

w : penalty

Finding optimal schedule using matroid

How to find an optimal schedule?

1. Optimizing over tasks in the canonical form:
   1.1 Find a set A of tasks that are early
   1.2 Sort the tasks of A in increasing deadlines
   1.3 Add late tasks in any order
2. Minimize penalties of late tasks $\equiv$ maximize penalties of early tasks

Modeled by a matroid $M = (S, \mathcal{I})$, where

$S = \{a_1, \ldots, a_n\}$

$\mathcal{I} = \{A \subseteq S : \exists \text{ a way to schedule the tasks in } A \text{ s.t. no task is late}\}$

w : penalty

Finding an optimal schedule $\equiv$ finding max-weighted indep. subset of M

Such M for task scheduling is a matroid

$M = (S, \mathcal{I})$ is a matroid

Such M for task scheduling is a matroid

$M = (S, \mathcal{I})$ is a matroid

- $\mathcal{I}$ has the hereditary property:
  if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$

Such M for task scheduling is a matroid

$M = (S, \mathcal{I})$ is a matroid

- $\mathcal{I}$ has the hereditary property:
  if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$
- Exchange property:
  Say $A, B \in \mathcal{I}$ and $|B| > |A|$.

Such M for task scheduling is a matroid

$M = (S, \mathcal{I})$ is a matroid

- $\mathcal{I}$ has the hereditary property:
  if $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$
- Exchange property:
  Say $A, B \in \mathcal{I}$ and $|B| > |A|$.
  Assume A and B are sorted in increasing order of deadlines
  We need to show there exists an $x \in B - A$ s.t. $A \cup \{x\} \in \mathcal{I}$
  See Cormen et al. proof of Theorem 16.13

Greedy algorithm for finding optimal scheduling

```
def GREEDY(M = (S, 𝓘), weights w):
```

Greedy algorithm for finding optimal scheduling

def $\text{GREEDY}(M = (S, \mathcal{I}), \textit{weights } w)$**:**

  Set $A := \{\,\}$;

Greedy algorithm for finding optimal scheduling

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;                // O(n log n)
```

Greedy algorithm for finding optimal scheduling

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;        // O(n log n)
    for x ∈ S:
```

Greedy algorithm for finding optimal scheduling

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;        // O(n log n)
    for x ∈ S:
        if A ∪ {x} ∈ I:
```

Greedy algorithm for finding optimal scheduling

def $\textsc{Greedy}$$(M = (S, \mathcal{I}),$ *weights* $w)$**:**

  Set $A := \{\,\}$;

  Sort S in decreasing order of w ; // $O(n \log n)$

  for $x \in S$**:**

    if $A \cup \{x\} \in \mathcal{I}$**:**

      $A := A \cup \{x\}$;

Greedy algorithm for finding optimal scheduling

def GREEDY($M = (S, \mathcal{I})$, *weights* w)**:**

- Set $A := \{\,\}$;
- Sort S in decreasing order of w ; // $O(n \log n)$
- **for** $x \in S$**:**
  - **if** $A \cup \{x\} \in \mathcal{I}$**:**
    - $A := A \cup \{x\}$;
- **return** A;

Greedy algorithm for finding optimal scheduling

def $\textsc{Greedy}(M = (S, \mathcal{I}), \textit{weights } w)$**:**

  Set $A := \{\,\}$;

  Sort S in decreasing order of w ; // $O(n \log n)$

  for $x \in S$**:**

    if $A \cup \{x\} \in \mathcal{I}$**:**

      $A := A \cup \{x\}$;

  return A;

Running time: let $n = |S|$

Greedy algorithm for finding optimal scheduling

```
def GREEDY(M = (S, I), weights w):
    Set A := { };
    Sort S in decreasing order of w ;          // O(n log n)
    for x ∈ S:
        if A ∪ {x} ∈ I:
            A := A ∪ {x};
    return A;
```

Running time: let $n = |S|$

Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$.

Greedy algorithm for finding optimal scheduling

```
1 def Greedy(M = (S, I), weights w):
2     Set A := { };
3     Sort S in decreasing order of w ;          // O(n log n)
4     for x ∈ S:
5         if A ∪ {x} ∈ I:
6             A := A ∪ {x};
7     return A;
```

Running time: let $n = |S|$

Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$. Lines 5-6 takes $O(n \cdot f(n))$

1 **def** $\textsc{Greedy}(M = (S, \mathcal{I}), \textit{weights } w)$**:**
2   Set $A := \{\,\}$;
3   Sort S in decreasing order of w ; // $O(n \log n)$
4   **for** $x \in S$**:**
5   **if** $A \cup \{x\} \in \mathcal{I}$**:**
6             $A := A \cup \{x\}$;
7   **return** A;

Running time: let $n = |S|$

Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$. Lines 5-6 takes $O(n \cdot f(n))$

Claim: $f(n) = O(n)$ for task scheduling problem (Homework)

Greedy algorithm for finding optimal scheduling

```
1 def GREEDY(M = (S, I), weights w):
2     Set A := { };
3     Sort S in decreasing order of w ;              // O(n log n)
4     for x ∈ S:
5         if A ∪ {x} ∈ I:
6             A := A ∪ {x};
7     return A;
```

Running time: let $n = |S|$

Assume checking if $A \cup \{x\} \in \mathcal{I}$ takes $O(f(n))$. Lines 5-6 takes $O(n \cdot f(n))$

Claim: $f(n) = O(n)$ for task scheduling problem (Homework)

Total running time: $O(n^2)$

Greedy algorithms

Horn formulas (Textbook Section 5.3)

Horn formulas — Prelude

Consider the following puzzle

Horn formulas — Prelude

Consider the following puzzle

- If Alice has a dog, then Bob has a cat

Horn formulas — Prelude

Consider the following puzzle

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat

Horn formulas — Prelude

Consider the following puzzle

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Horn formulas — Prelude

Consider the following puzzle

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Question: what pets do they have?

Boolean formulas

Basics of boolean formulas

Boolean formulas

Basics of boolean formulas

- **Variables:** possibilities
  Knowledge about variables is represented by a special type of boolean formulas

Boolean formulas

Basics of boolean formulas

- **Variables:** possibilities
  Knowledge about variables is represented by a special type of boolean formulas
  Goal; find a *consistent* explanation of the knowledge

Boolean formulas

Basics of boolean formulas

- **Variables:** possibilities
  Knowledge about variables is represented by a special type of boolean formulas
  Goal; find a *consistent* explanation of the knowledge
- **Boolean variable:** $x = 1$ (true) or $x = 0$ (false)

Boolean formulas

Basics of boolean formulas

- **Variables:** possibilities
  Knowledge about variables is represented by a special type of boolean formulas
  Goal; find a *consistent* explanation of the knowledge
- **Boolean variable:** $x = 1$ (true) or $x = 0$ (false)
- **Literal:** x (positive literal), $\bar{x}$ (negative literal)

Boolean formulas

Basics of boolean formulas

- **Variables:** possibilities
  Knowledge about variables is represented by a special type of boolean formulas
  Goal; find a *consistent* explanation of the knowledge
- **Boolean variable:** $x = 1$ (true) or $x = 0$ (false)
- **Literal:** x (positive literal), $\bar{x}$ (negative literal)
- **Clause:** a clause consists of literals connected by
  $\wedge$ (AND), $\vee$ (OR), $\implies$ (implies)

Boolean formulas

Basics of boolean formulas

- **Variables:** possibilities
  Knowledge about variables is represented by a special type of boolean formulas
  Goal; find a *consistent* explanation of the knowledge
- **Boolean variable:** $x = 1$ (true) or $x = 0$ (false)
- **Literal:** x (positive literal), $\bar{x}$ (negative literal)
- **Clause:** a clause consists of literals connected by
  $\wedge$ (AND), $\vee$ (OR), $\implies$ (implies)
  Examples: $x \wedge \bar{y}, \quad (x \wedge y) \implies z$

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \land x_2 \land \cdots \land x_n) \implies y$

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗
  - $(x \vee y) \implies z$

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗
  - $(x \vee y) \implies z$ ✗

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗
  - $(x \vee y) \implies z$ ✗
  - $\implies z$

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗
  - $(x \vee y) \implies z$ ✗
  - $\implies z$ ✓

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗
  - $(x \vee y) \implies z$ ✗
  - $\implies z$ ✓
- **Pure negative clauses** $\bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_n$

Horn formulas

In a Horn formula, there are only two types of clauses (**Horn clauses**):

- **Implication:** $(x_1 \wedge x_2 \wedge \cdots \wedge x_n) \implies y$
  LHS: AND of any number of positive literals
  RHS: single positive literal
  - $(x \wedge \bar{y}) \implies z$ ✗
  - $(x \vee y) \implies z$ ✗
  - $\implies z$ ✓
- **Pure negative clauses** $\bar{x}_1 \vee \bar{x}_2 \vee \cdots \vee \bar{x}_n$
  OR of any number of negative literals

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

$a \implies y$

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

$a \implies y$

$(b \wedge c) \implies x$

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

$$a \implies y$$
$$(b \wedge c) \implies x$$
$$(y \wedge z) \implies x$$

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

$a \implies y$

$(b \wedge c) \implies x$

$(y \wedge z) \implies x$

$\bar{a} \vee \bar{c}$

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

$$a \implies y$$

$$(b \wedge c) \implies x$$

$$(y \wedge z) \implies x$$

$$\bar{a} \vee \bar{c}$$

$$\bar{x} \vee \bar{z}$$

Horn formula example

Consider the puzzle:

- If Alice has a dog, then Bob has a cat
- If Charlie and Bob both have pets of the same species, then Alice has a cat
- Charlie and Alice don't share a pet of the same species

Define variables:

- a: Alice has a dog
- b: Bob has a dog
- c: Charlie has a dog
- x: Alice has a cat
- y: Bob has a cat
- z: Charlie has a cat

Modelled by a set of Horn clauses:

$a \implies y$

$(b \wedge c) \implies x$

$(y \wedge z) \implies x$

$\bar{a} \vee \bar{c}$

$\bar{x} \vee \bar{z}$

Question: satisfying assignment?

Greedy approach for Horn formulas

Problem (Horn Satisfiability)

Given a set of Horn clauses, determine whether or not there is a consistent explanation,

Greedy approach for Horn formulas

Problem (Horn Satisfiability)

Given a set of Horn clauses, determine whether or not there is a consistent explanation, i.e., an assignment of 0/1 to variables that satisfy all clauses

Greedy approach for Horn formulas

Problem (Horn Satisfiability)

Given a set of Horn clauses, determine whether or not there is a consistent explanation, i.e., an assignment of 0/1 to variables that satisfy all clauses

Example: $(x \wedge y) \implies z$, $\bar{x} \vee \bar{w}$

Greedy approach for Horn formulas

Problem (Horn Satisfiability)

Given a set of Horn clauses, determine whether or not there is a consistent explanation, i.e., an assignment of 0/1 to variables that satisfy all clauses

Example: $(x \wedge y) \implies z$, $\bar{x} \vee \bar{w}$ can be satisfied by $x = 0, y = 0, z = 0, w = 0$

Greedy approach for Horn formulas

Problem (Horn Satisfiability)

Given a set of Horn clauses, determine whether or not there is a consistent explanation, i.e., an assignment of 0/1 to variables that satisfy all clauses

Example: $(x \wedge y) \implies z$, $\bar{x} \vee \bar{w}$ can be satisfied by $x = 0, y = 0, z = 0, w = 0$

Greedy heuristic: start with all 0. Only set a variable to 1 if you need to, i.e., when an implication says you need to

Greedy approach for Horn formulas

Problem (Horn Satisfiability)

Given a set of Horn clauses, determine whether or not there is a consistent explanation, i.e., an assignment of 0/1 to variables that satisfy all clauses

Example: $(x \wedge y) \implies z$, $\bar{x} \vee \bar{w}$ can be satisfied by $x = 0, y = 0, z = 0, w = 0$

Greedy heuristic: start with all 0. Only set a variable to 1 if you need to, i.e., when an implication says you need to

Recall: $p \implies q \iff \bar{p} \vee q$

Pseudocode

def GREEDY_HORN(*set of Horn clauses*)**:**

Pseudocode

def GREEDY_HORN(*set of Horn clauses*)**:**

  Set all variables to 0;

Pseudocode

def GREEDY_HORN(*set of Horn clauses*)**:**

    Set all variables to 0;

    while *there exists an "* $\Longrightarrow$ *" that is not satisfied***:**

Pseudocode

def GREEDY_HORN(*set of Horn clauses*)**:**

    Set all variables to 0;
    while *there exists an "$\Longrightarrow$" that is not satisfied***:**
        Set its RHS to 1;

Pseudocode

```
def GREEDY_HORN(set of Horn clauses):
    Set all variables to 0;
    while there exists an "⟹" that is not satisfied:
        Set its RHS to 1;
    if all pure negative clauses are 1:
```

Pseudocode

```
def GREEDY_HORN(set of Horn clauses):
    Set all variables to 0;
    while there exists an "⟹" that is not satisfied:
        Set its RHS to 1;
    if all pure negative clauses are 1:
        return the assignment;
```

Pseudocode

```
def GREEDY_HORN(set of Horn clauses):
    Set all variables to 0;
    while there exists an "⟹" that is not satisfied:
        Set its RHS to 1;
    if all pure negative clauses are 1:
        return the assignment;
    else:
```

Pseudocode

```
def GREEDY_HORN(set of Horn clauses):
    Set all variables to 0;
    while there exists an "⟹" that is not satisfied:
        Set its RHS to 1;
    if all pure negative clauses are 1:
        return the assignment;
    else:
        return "unsatisfiable";
```

Pseudocode

```
def GREEDY_HORN(set of Horn clauses):
    Set all variables to 0;
    while there exists an "⟹" that is not satisfied:
        Set its RHS to 1;
    if all pure negative clauses are 1:
        return the assignment;
    else:
        return "unsatisfiable";
```

Example: $\implies x$, $x \implies y$, $(\bar{x} \vee \bar{y})$

Pseudocode

def $\textsc{Greedy\_Horn}$(*set of Horn clauses*)**:**

```
Set all variables to 0;
while there exists an "⟹" that is not satisfied:
    Set its RHS to 1;
if all pure negative clauses are 1:
    return the assignment;
else:
    return "unsatisfiable";
```

Example: $\implies x$, $x \implies y$, $(\bar{x} \vee \bar{y})$

$x \quad y$

Pseudocode

def GREEDY_HORN(*set of Horn clauses*)**:**

```
Set all variables to 0;
while there exists an "⟹" that is not satisfied:
    Set its RHS to 1;
if all pure negative clauses are 1:
    return the assignment;
else:
    return "unsatisfiable";
```

Example: $\implies x$, $x \implies y$, $(\bar{x} \vee \bar{y})$

x	y	
0	0	$\implies x$ ✗

Pseudocode

def $\textsc{Greedy\_Horn}$(*set of Horn clauses*)**:**

```
Set all variables to 0;
while there exists an "⟹" that is not satisfied:
    Set its RHS to 1;
if all pure negative clauses are 1:
    return the assignment;
else:
    return "unsatisfiable";
```

Example: $\implies x,\ x \implies y,\ (\bar{x} \vee \bar{y})$

x	y	
0	0	$\implies x$ ✗
1	0	$x \implies y$ ✗

Pseudocode

def $\textsc{Greedy\_Horn}$(*set of Horn clauses*)**:**

    Set all variables to 0;
    while *there exists an "$\implies$" that is not satisfied***:**
        Set its RHS to 1;
    if *all pure negative clauses are 1***:**
        return the assignment;
    else:
        return "unsatisfiable";

Example: $\implies x$, $x \implies y$, $(\bar{x} \vee \bar{y})$

x	y	
0	0	$\implies x$ ✗
1	0	$x \implies y$ ✗
1	1	$\implies x$ ✓, $x \implies y$ ✓, $(\bar{x} \vee \bar{y})$ ✗

Pseudocode

def GREEDY_HORN(*set of Horn clauses*)**:**

```
Set all variables to 0;
while there exists an "⟹" that is not satisfied:
    Set its RHS to 1;
if all pure negative clauses are 1:
    return the assignment;
else:
    return "unsatisfiable";
```

Example: $\implies x$, $x \implies y$, $(\bar{x} \vee \bar{y})$

x	y	
0	0	$\implies x$ ✗
1	0	$x \implies y$ ✗
1	1	$\implies x$ ✓, $x \implies y$ ✓, $(\bar{x} \vee \bar{y})$ ✗

Unsatisfiable

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Theorem

The variables set to 1 by GREEDY_HORN *must be 1 in **any** satisfying assignment*

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Theorem

The variables set to 1 by GREEDY_HORN *must be 1 in* ***any*** *satisfying assignment*

Exercise: Prove this by induction

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Theorem

The variables set to 1 by GREEDY_HORN *must be 1 in* ***any*** *satisfying assignment*

Exercise: Prove this by induction

How does this theorem help?

If all the pure negative clauses cannot be satisfied after the while loop, then there's no such assignment satisfying them

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Theorem

The variables set to 1 by GREEDY_HORN *must be 1 in* ***any*** *satisfying assignment*

Exercise: Prove this by induction

How does this theorem help?

If all the pure negative clauses cannot be satisfied after the while loop, then there's no such assignment satisfying them

Running time: Let n be the size of the Horn formula, i.e., the number occurrences of literals.

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Theorem

The variables set to 1 by GREEDY_HORN *must be 1 in* ***any*** *satisfying assignment*

Exercise: Prove this by induction

How does this theorem help?

If all the pure negative clauses cannot be satisfied after the while loop, then there's no such assignment satisfying them

Running time: Let n be the size of the Horn formula, i.e., the number occurrences of literals.

Total running time: $O(n^2)$.

Correctness and running time

Correctness: If GREEDY_HORN finds an assignment, then the problem has a satisfying assignment

If it returns "unsatisfiable", is it really unsatisfiable?

Theorem

The variables set to 1 by GREEDY_HORN *must be 1 in* ***any*** *satisfying assignment*

Exercise: Prove this by induction

How does this theorem help?

If all the pure negative clauses cannot be satisfied after the while loop, then there's no such assignment satisfying them

Running time: Let n be the size of the Horn formula, i.e., the number occurrences of literals.

Total running time: $O(n^2)$. Can be improved to $O(n)$ (**exercise**)