Greedy algorithms

David Kauchak
cs140
Spring 2024

Greedy algorithms

Algorithm that makes a local decision with the goal of creating a globally optimal solution

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to sub-problems

Administrative

Assignment 6
Grades
Dr. Dave's grades

Greedy

To solve the general problem:

Pick a locally optimal solution and repeat
Horn formula

A horn formula is a set of implications and negative clauses:

\[
\begin{align*}
\Rightarrow x & : x \land u \Rightarrow z \\
\Rightarrow y & : \overline{x} \lor \overline{y} \lor z
\end{align*}
\]

Negated literals ored

Goal

Given a horn formula, determine if the formula is satisfiable, i.e., an assignment of true/false to the variables that is consistent with all of the implications/causes:

\[
\begin{align*}
\Rightarrow x & : x \land u \Rightarrow z \\
\Rightarrow y & : \overline{x} \lor \overline{y} \lor z
\end{align*}
\]

\[
\begin{array}{cccc}
u & x & y & z \\
0 & 1 & 1 & 0
\end{array}
\]
A greedy solution?

\[ x \Rightarrow x \wedge z \Rightarrow w \wedge y \wedge z \Rightarrow x \]
\[ x \Rightarrow y \quad x \wedge y \Rightarrow w \quad \overline{w} \vee x \vee \overline{y} \]

w 0
x 0
y 0
z 0

A greedy solution?

\[ x \Rightarrow x \wedge z \Rightarrow w \wedge y \wedge z \Rightarrow x \]
\[ x \Rightarrow y \quad x \wedge y \Rightarrow w \quad \overline{w} \vee x \vee \overline{y} \]

w 0
x 1
y 0
z 0

A greedy solution?

\[ x \Rightarrow x \wedge z \Rightarrow w \wedge y \wedge z \Rightarrow x \]
\[ x \Rightarrow y \quad x \wedge y \Rightarrow w \quad \overline{w} \vee x \vee \overline{y} \]

w 0
x 1
y 1
z 0

A greedy solution?

\[ x \Rightarrow x \wedge z \Rightarrow w \wedge y \wedge z \Rightarrow x \]
\[ x \Rightarrow y \quad x \wedge y \Rightarrow w \quad \overline{w} \vee x \vee \overline{y} \]

w 1
x 1
y 1
z 0

A greedy solution?

\[ x \Rightarrow x \wedge z \Rightarrow w \wedge y \wedge z \Rightarrow x \]
\[ x \Rightarrow y \quad x \wedge y \Rightarrow w \quad \overline{w} \vee x \vee \overline{y} \]

w 0
x 1
y 0
z 0
A greedy solution?

\[ x \implies w \wedge x \wedge z \implies x \]

\[ x \implies y \implies w \vee \neg x \vee \neg y \]

\[ w \ 1 \]
\[ x \ 1 \quad \text{not satisfiable} \]
\[ y \ 1 \]
\[ z \ 0 \]

A greedy solution

```plaintext
A greedy solution

1: set all variables to false
2: for all implications i
3: if Empty(LHS(i))
4: RHS(i) = true
5: changed = true
6: while changed
7: changed = false
8: for all implications i
9: if LHS(i) = true and RHS(i) = true
10: RHS(i) = true
11: changed = true
12: for all negative clauses c
13: if c = false
14: return false
15: return true
```

A greedy solution

```plaintext
A greedy solution

1: set all variables to false
2: for all implications i
3: if Empty(LHS(i))
4: RHS(i) = true
5: changed = true
6: while changed
7: changed = false
8: for all implications i
9: if LHS(i) = true and RHS(i) = true
10: RHS(i) = true
11: changed = true
12: for all negative clauses c
13: if c = false
14: return false
15: return true
```

not satisfiable
A greedy solution

Base case:
1. set all variables to false
2. for all implications $i$
3. if DiF$\neg\varphi_i$($\neg V$)
4. $\text{changed} \leftarrow \text{true}$
5. while $\text{changed}$
6. $\text{changed} \leftarrow \text{false}$
7. for all implications $i$
8. if L$\varphi_i$($\neg V$) and R$\varphi_i$($\neg V$) = true
9. $\text{RHS}() \leftarrow \text{true}$
10. $\text{changed} \leftarrow \text{true}$
11. for all negative clauses $c$
12. if $c \leftarrow \text{false}$
13. return false
14. return true

How is this a greedy algorithm?

Correctness of greedy solution

Two parts:
- If our algorithm returns an assignment, is it a valid assignment?
- If our algorithm does not return an assignment, does an assignment exist?
Correctness of greedy solution

If our algorithm returns an assignment, is it a valid assignment?

```plaintext
Correctness of greedy solution
If our algorithm returns an assignment, is it a valid assignment?

```
Correctness of greedy solution

If our algorithm does not return an assignment, does an assignment exist?

Running time?

```plaintext
Horn(f)
1: set all variables to false
2: for all implications i
3: if Entry(LHS(i))
4: changed ← true
5: for all implications i
6: if LHS(i) ∧ true and RHS(i) ∧ true
7: RHS(i) ← true
8: if LHS(i) ∧ true and RHS(i) ∧ true
9: RHS(i) ← true
10: if LHS(i) ∧ true and RHS(i) ∧ true
11: RHS(i) ← true
12: if LHS(i) ∧ true and RHS(i) ∧ true
13: RHS(i) ← true
14: return false
15: return true
```

Running time?

```
O(nm)
```

Data compression

Given a file containing some data of a fixed alphabet Σ (e.g., A, B, C, D), we would like to pick a binary character code that minimizes the number of bits required to represent the data.

minimize the size of the encoded file

```
ACADAADB...
```

```
01010010100 ...
```
Compression algorithms

http://en.wikipedia.org/wiki/Lossless_data_compression

Simplifying assumption: frequency only

Assume that we only have character frequency information for a file

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Fixed length code

Use \([\lfloor \log_2 \Sigma \rfloor] \) bits for each character

A = 00
B = 01
C = 10
D = 11

How many bits to encode the file?

\[2 \times 70 + 2 \times 3 + 2 \times 20 + 2 \times 37 = 260 \text{ bits}\]
### Fixed length code

Use \([\log_2 |C|]\) bits for each character

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

\[2 \times 70 + 2 \times 3 + 2 \times 20 + 2 \times 37 = 260 \text{ bits}\]

Can we do better?

### Variable length code

What about:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

\[1 \times 70 + 2 \times 3 + 2 \times 20 + 1 \times 37 = 153 \text{ bits}\]

How many bits to encode the file?

### Decoding a file

A = 0
B = 01
C = 10
D = 1

What characters does this sequence represent?

A D or B?
Variable length code

What about:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

Is it decodeable?

\[1 \times 70 + 3 \times 3 + 3 \times 20 + 2 \times 37 = 213 \text{ bits}(18\% \text{ reduction})\]

How many bits to encode the file?

Prefix codes

A prefix code is a set of codes where no codeword is a prefix of any other codeword

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
</tbody>
</table>

Prefix tree

We can encode a prefix code using a full binary tree where each leaf represents an encoding of a symbol

A = 0
B = 01
C = 10
D = 11
Decoding using a prefix tree

To decode, we traverse the graph until a leaf node is reached and output the symbol

A = 0
B = 100
C = 101
D = 11

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

100111010100

B
Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

100111010100
B A D

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1001011010100
B A D C

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

100111010100
B A D C A

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

100111010100
B A D C A B
Determining the cost of a file

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

If we label the internal nodes with the sum of the children...

\[
\text{cost}(T) = \sum_{i=1}^{n} f_i \times \text{depth}(i)
\]

Cost is equal to the sum of the internal nodes (excluding the root) and the leaf nodes.
Determining the cost of a file

As we move down the tree, one bit gets read for every non-root node:

- 70 times we see a 0 by itself.
- 60 times we see a prefix that starts with a 1.
- Of those, 37 times we see an additional 1.
- The remaining 23 times we see an additional 0.
- Of those, 20 times we see a last 1, and 3 times a last 0.

A greedy algorithm?

Given file frequencies, can we come up with a prefix-free encoding (i.e., build a prefix tree) that minimizes the number of bits?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

A greedy algorithm?

Given file frequencies, can we come up with a prefix-free encoding (i.e., build a prefix tree) that minimizes the number of bits?

```
Huffman(F)
1. Q ← MakeHeap(F)
2. for i = 1 to |Q| - 1
3. allocate a new node : 
4. let r(f[x]) ← x ← ExtractMin(Q)
5. let r(f[x]) ← y ← ExtractMin(Q)
6. let f[x] ← f[x] + f[y]
7. let Insert(Q, x)
8. return ExtractMin(Q)
```
Symbol | Frequency
--- | ---
A | 70
B | 3
C | 20
D | 37

Heap

B 3
C 20
D 37
A 70

merging with this node will incur an additional cost of 23

B 23
C 20
D 37
A 70

Symbol | Frequency
--- | ---
A | 70
B | 3
C | 20
D | 37

Heap

BCD 60
A 70

Symbol | Frequency
--- | ---
A | 70
B | 3
C | 20
D | 37

Heap

ABCD 130
What is the code (assume left = 0)?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

What is the code (assume left = 0)?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>D</td>
<td>37</td>
</tr>
</tbody>
</table>

What is the code (assume left = 0)?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>000</td>
</tr>
<tr>
<td>C</td>
<td>001</td>
</tr>
<tr>
<td>D</td>
<td>01</td>
</tr>
</tbody>
</table>

Proving correctness

The algorithm selects the symbols with the two smallest frequencies first (call them \( f_1 \) and \( f_2 \))

Proving correctness: proof by contradiction

The algorithm selects the symbols with the two smallest frequencies first (call them \( f_1 \) and \( f_2 \))

Consider a tree that did not do this:

Is it optimal?
Proving correctness

The algorithm selects the symbols with the two smallest frequencies first (call them \( f_1 \) and \( f_2 \)).

Consider a tree that did not do this:

\[
\text{cost}(T) = \sum_{i} f_i \text{depth}(i)
\]

- frequencies don’t change
- cost will decrease since \( f_1 < f_i \)

contradiction

Runtime?

1 call to MakeHeap
2(n-1) calls ExtractMin
n-1 calls Insert

\( O(n \log n) \)

Non-optimal greedy algorithms

All the greedy algorithms we’ve looked at so far give the optimal answer.

Some of the most common greedy algorithms generate good, but non-optimal solutions:

- set cover
- clustering
- hill-climbing
- relaxation

Handout
Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100