Chapter 27. Maximum Marking Problems

Zhenjiang Hu, Wei Zhang

School of Computer Scieence
Peking University

December 24, 2025

Maximum Independent Sublist Sum Problem

Compute a way of marking of the elements in a list, such that no two marked elements are adjacent and the sum of the marked elements are maximum. For instance,

$$mis\ [1, 2, 3, 4, 5] = [1^*, 2, 3^*, 4, 5^*],$$

which gives the maximum sum of 9 among all the feasible marking.

Maximum Even-Segment Sum Problem

Compute a way of marking of the elements in a list, such that all marked elements are adjacent, the number of marked elements is even, and the sum of the marked elements are maximum. For instance,

$$mess\ [1, 2, 3, -4, 4] = [1, 2^*, 3^*, -4, 4],$$

An Optimal Coloring Problem

Suppose there are three markers: red, blue, and yellow. The problem is to find a way of marking all the elements such that each sort of mark does not appear continuously, and that the sum of the elements marked in red minus the sum of the elements marked in blue is maximum.

Maximum Marking Problem

Problem Definition

Given a list *xs* (of type $[\alpha]$), a *maximum marking problem* is to find a marking of *xs*'s elements such that

- the marked list xs^* (of type $[\alpha^*]$) satisfies a certain property *p*, and
- the sum of the marked elements in xs^* is maximum.

Specification

$$
\begin{aligned}
mmp &: ([\alpha^*] \to Bool) \to [\alpha] \to [\alpha^*] \\
mmp\ p &= \uparrow_{sum} / \circ p\triangleleft \circ gen
\end{aligned}
$$

Specification

$$
\begin{aligned}
mmp &: ([\alpha^*] \to Bool) \to [\alpha] \to [\alpha^*] \\
mmp\ p &= \uparrow_{sum} / \ \circ\ p \triangleleft\ \circ\ gen
\end{aligned}
$$

The function *gen* generates all the possible markings of input data.

$$
\begin{aligned}
gen &: [\alpha] \to [[\alpha^*]] \\
gen\ [] &= [] \\
gen\ [a] &= [[(a, True)], [(a, False)]] \\
gen\ (x +\!\!+\ y) &= gen\ x\ X_{+\!+}\ gen\ y
\end{aligned}
$$

Specification

$$
\begin{aligned}
mmp &: ([\alpha^*] \to Bool) \to [\alpha] \to [\alpha^*] \\
mmp\ p &= \uparrow_{sum} / \circ\ p\triangleleft \circ gen
\end{aligned}
$$

The function *gen* generates all the possible markings of input data.

$$
\begin{aligned}
gen &: [\alpha] \to [[\alpha^*]] \\
gen\ [] &= [] \\
gen\ [a] &= [[(a, True)], [(a, False)]] \\
gen\ (x \mathbin{+\!\!+} y) &= gen\ x\ X_{+\!\!+}\ gen\ y
\end{aligned}
$$

That is,

$$
gen = X_{+\!\!+} / \cdot (\lambda a.[[(a, True)], [(a, False)]]) *
$$

Theorem (Sasano et al.: ICFP'00)

For the specification

$$mmp\ p \ = \uparrow_{sum} / \ \circ \ p\triangleleft \ \circ gen$$

if $p = accept \circ h$ *where h is a right-to-left reduction with finite range, then we can have a linear time algorithm to solve the problem:*

$$mmp\ p \ = \uparrow_{fst} / \circ h'$$

where h' *is a right-to-left reduction.* [Note: $O(|range(h)| * n)$]

Isao Sasano, Zhenjiang Hu, Masato Takeichi, Mizuhito Ogawa, Make it Practical: A Generic Linear Time Algorithm for Solving Maximum Weightsum Problems , The 2000 ACM SIGPLAN International Conference on Functional Programming, (ICFP 2000), Montreal, Canada, 18-20 September 2000. ACM Press. pp. 137-149.

Right-to-Left Reduction: Review

$$\oplus \not\leftarrow [a_1, \ldots, a_{n-1}, a_n] \quad = \quad a_1 \oplus (\cdots \oplus (a_{n-1} \oplus a_n))$$

Right-to-Left Reduction: Review

$$\oplus \not\leftarrow [a_1, \ldots, a_{n-1}, a_n] \quad = \quad a_1 \oplus (\cdots \oplus (a_{n-1} \oplus a_n))$$

Derivation of Right-to-Left Reduction

If a function f is defined in the following form

$$\begin{array}{lcl} f\,[x] & = & k\ x \\ f\,(x : xs) & = & x \oplus f\ xs \end{array}$$

then f is a right-to-left reduction.

Right-to-Left Reduction: Review

$$\oplus \not\leftarrow [a_1, \ldots, a_{n-1}, a_n] \quad = \quad a_1 \oplus (\cdots \oplus (a_{n-1} \oplus a_n))$$

Derivation of Right-to-Left Reduction: Tupling

Let h be defined by

$$h\ xs = (f_1\ xs, \ldots, f_n\ xs).$$

If each f_i is defined in the following form

$$\begin{array}{lcl} f_i\ [x] & = & k_i\ x \\ f_i\ (x : xs) & = & x \oplus_i (f_1\ xs, \ldots, f_n\ xs) \end{array}$$

then h is a right-to-left reduction.

Example: Maximum Independent Sublist Sum Problem

The condition is that no two marked elements are adjacent.

$$p \quad : \quad [\alpha^*] \to \mathit{Bool}$$

Example: Maximum Independent Sublist Sum Problem

The condition is that no two marked elements are adjacent.

$$
\begin{array}{lcl}
p & : & [\alpha^*] \to \mathit{Bool} \\
p\ [x] & = & \mathit{True}
\end{array}
$$

Example: Maximum Independent Sublist Sum Problem

The condition is that no two marked elements are adjacent.

$$
\begin{array}{lcl}
p & : & [\alpha^*] \rightarrow Bool \\
p\ [x] & = & True \\
p\ (x : xs) & = & \textbf{if}\ marked\ x \\
 & & \textbf{then}\ not\ (marked\ (hd\ xs))\ \wedge\ p\ xs \\
 & & \textbf{else}\ p\ xs
\end{array}
$$

Example: Maximum Independent Sublist Sum Problem

The condition is that no two marked elements are adjacent.

$$
\begin{array}{lcl}
p & : & [\alpha^*] \rightarrow \mathit{Bool} \\
p\ [x] & = & \mathit{True} \\
p\ (x : xs) & = & \textbf{if } \mathit{marked}\ x \\
 & & \textbf{then } \mathit{not}\ (\mathit{marked}\ (\mathit{hd}\ xs)) \ \wedge \ p\ xs \\
 & & \textbf{else } p\ xs
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{hd} & : & [\alpha] \rightarrow \alpha \\
\mathit{hd}\ [x] & = & x \\
\mathit{hd}\ (x : xs) & = & x
\end{array}
$$

Example: Maximum Independent Sublist Sum Problem

The condition is that no two marked elements are adjacent.

$$
\begin{array}{lcl}
p & : & [\alpha^*] \to \mathit{Bool} \\
p\ [x] & = & \mathit{True} \\
p\ (x : xs) & = & \textbf{if}\ \mathit{marked}\ x \\
 & & \textbf{then}\ \mathit{not}\ (\mathit{marked}\ (\mathit{hd}\ xs))\ \wedge\ p\ xs \\
 & & \textbf{else}\ p\ xs
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{hd} & : & [\alpha] \to \alpha \\
\mathit{hd}\ [x] & = & x \\
\mathit{hd}\ (x : xs) & = & x
\end{array}
$$

How to express p in terms of a right-to-left reduction with finite range?

We calculate p to our required form.

- Fusing $mhd = marked \cdot hd$.

$$
\begin{array}{lcl}
mhd\ [x] & = & marked\ x \\
mhd\ (x : xs) & = & marked\ x
\end{array}
$$

(Note: $marked\ (x, b) = b$.)

We calculate p to our required form.

- Fusing $mhd = marked \cdot hd$.

$$\begin{array}{lcl} mhd\ [x] & = & marked\ x \\ mhd\ (x : xs) & = & marked\ x \end{array}$$

(Note: $marked\ (x, b) = b$.)

- Tupling p with mhd.

$$h\ xs \quad = \quad (p\ xs, mhd\ xs)$$

We calculate p to our required form.

- Fusing $mhd = marked \cdot hd$.

$$\begin{array}{lcl} mhd\ [x] & = & marked\ x \\ mhd\ (x : xs) & = & marked\ x \end{array}$$

(Note: $marked\ (x, b) = b$.)

- Tupling p with mhd.

$$h\ xs \quad = \quad (p\ xs, mhd\ xs)$$

- Thus, $p = fst \cdot h$.

Applying the theorem gives the following linear time program.

```
mis :: [Elem] -> (Class,Weight,[MElem])
mis xs = let opts = mis' xs
         in getmax [ (c,w,cand)
                   | (c,w,cand) <- opts,
                     c==2 || c==3]

mis' :: [Elem] -> [(Class,Weight,[MElem])]
mis' [x] = [(2,x,[(x,True)]), (3,0,[(x,False)])]
mis' (x:xs) =
  let opts = mis' xs
  in eachmax [(table (marked mx) c,
               (if marked mx then weight mx else 0)
               + w,
               mx:cand)
             | mx <- [mark x, unmark x],
               (c,w,cand) <- opts]

getmax :: (Eq c, Ord w) => [(c,w,a)] -> (c,w,a)
getmax [] = error "No solution."
getmax xs = foldr1 f xs
  where f (c1,w1,cand1) (c2,w2,cand2)
          = if w1>w2 then (c1,w1,cand1) else (c2,w2,cand2)

eachmax :: (Eq c, Ord w) => [(c,w,a)] -> [(c,w,a)]
eachmax xs = foldr f [] xs
  where f (c,w,cand) [] = [(c,w,cand)]
        f (c,w,cand) ((c',w',cand') : opts) =
            if c==c' then
               if w>w' then (c,w,cand) : opts
               else (c',w',cand') : opts
            else (c',w',cand') : f (c,w,cand) opts
```

```
type Weight = Int
type Elem = Weight
type MElem = (Elem,Bool)
type Class = Int

weight :: MElem -> Weight
weight (w,_) = w

marked :: MElem -> Bool
marked (_,m) = m

mark :: Elem -> MElem
mark x = (x,True)

unmark :: Elem -> MElem
unmark x = (x,False)

table :: Bool -> Class -> Class
table True 0 = 0
table True 1 = 0
table True 2 = 0
table True 3 = 2
table False 0 = 1
table False 1 = 1
table False 2 = 3
table False 3 = 3
```

Example: Maximum Segment Sum Problem

The property is that all marked elements in a list should be adjacent (connected)

$$
\begin{array}{lcl}
\mathit{conn}\ [x] & = & \mathit{True} \\
\mathit{conn}\ (x : \mathit{xs}) & = & \textbf{if}\ \mathit{marked}\ x\ \textbf{then} \\
 & & \quad \mathit{nm}\ \mathit{xs} \vee \\
 & & \quad (\underline{\mathit{marked}\ (\mathit{hd}\ \mathit{xs})}\ \wedge \mathit{conn}\ \mathit{xs}) \\
 & & \textbf{else}\ \mathit{conn}\ \mathit{xs}
\end{array}
$$

Example: Maximum Segment Sum Problem

The property is that all marked elements in a list should be adjacent (connected)

$$
\begin{array}{lcl}
conn\ [x] & = & True \\
conn\ (x : xs) & = & \textbf{if}\ marked\ x\ \textbf{then} \\
 & & \quad nm\ xs\ \vee \\
 & & \quad (\underline{marked\ (hd\ xs)}\ \wedge conn\ xs) \\
 & & \textbf{else}\ conn\ xs
\end{array}
$$

$$
\begin{array}{lcl}
nm\ [x] & = & not\ (marked\ x) \\
nm\ (x : xs) & = & not\ (marked\ x)\ \wedge\ nm\ xs
\end{array}
$$

Example: Maximum Segment Sum Problem

The property is that all marked elements in a list should be adjacent (connected)

$$
\begin{array}{lcl}
conn\ [x] & = & True \\
conn\ (x : xs) & = & \textbf{if}\ marked\ x\ \textbf{then} \\
 & & \quad nm\ xs\ \vee \\
 & & \quad (\underline{marked\ (hd\ xs)}\ \wedge conn\ xs) \\
 & & \textbf{else}\ conn\ xs
\end{array}
$$

$$
\begin{array}{lcl}
nm\ [x] & = & not\ (marked\ x) \\
nm\ (x : xs) & = & not\ (marked\ x)\ \wedge\ nm\ xs
\end{array}
$$

Easy: fusion + Tupling!

Example: Maximum Even-Segment Sum Problem

The property is that all marked elements in a list should be adjacent (connected) and the number of marked elements is even.

Example: Maximum Even-Segment Sum Problem

The property is that all marked elements in a list should be adjacent (connected) and the number of marked elements is even.

$$p\ xs = conn\ xs\ \wedge\ evens\ xs$$

Example: Maximum Even-Segment Sum Problem

The property is that all marked elements in a list should be adjacent (connected) and the number of marked elements is even.

$$p\ xs = conn\ xs\ \wedge\ evens\ xs$$

where *evens* can be defined by

$$\begin{array}{lcl} evens\ [x] & = & \textbf{if}\ marked\ x\ \textbf{then}\ False \\ & & \textbf{else}\ True \\ evens\ (x : xs) & = & \textbf{if}\ marked\ x\ \textbf{then} \\ & & \quad not\ (evens\ xs) \\ & & \textbf{else}\ evens\ xs \end{array}$$

Extension

We can have a more powerful theorem to solve wider class of maximum marking problems [Sasano et al.: SAIG'01]:

$$mmp'\ p\ f\ k\ = \uparrow_{sum \circ f*} /\ \circ\ p \triangleleft\ \circ\ gen\ k$$

where we allow:

- generation of possible ways of marking with k markers,
- more flexible objective function with f,
- p to be described as a finite higher-order foldr1

Example: Optimal Coloring Problem

- $k = 3$, where 1 represents "Red", 2 represent "BLUE", and 3 represents "YELLOW".
- The property: each sort of marked color does not appear adjacently.

$$
\begin{array}{lcl}
\mathit{indep}\ xs & = & \mathit{indep}'\ xs\ 0 \\
\mathit{indep}'\ [x]\ \mathit{color} & = & \mathit{markKind}\ x \neq \mathit{color} \\
\mathit{indep}'\ (x : xs)\ \mathit{color} & = & \mathit{markKind}\ x \neq \mathit{color} \\
 & & \wedge \mathit{indep}'\ xs\ (\mathit{markKind}\ x).
\end{array}
$$

- Definition of f

$$
\begin{array}{l}
f\ x = \textbf{case}\ \mathit{markKind}\ x\ \textbf{of} \\
\quad 1 \rightarrow \mathit{weight}\ x \\
\quad 2 \rightarrow -(\mathit{weight}\ x) \\
\quad 3 \rightarrow 0
\end{array}
$$