Generic Matroids

Athan Clark

July 23, 2015

Abstract

Matroids are a great tool for optimization - say we have a set of elements, and some function to measure each element. If we want to maximize the *total* measure of a subset of the input, we can do so quickly with matroids. The principal is simple - if we first have all paths to potential solutions at hand (the matroid itself), and a function to take elements of our set and give us some unit that we can compare each other against (for instance, the *Ord* type class in Haskell), then we can find the subset with a maximum total very quickly.

Here we assert the properties and definition of matroids, to better formalize and demonstrate their utility.

1 Overview

A Matroid can be seen as the "road map" to every possible solution. The potential solutions are based on an input set, and the set of all potential solutions are the roads. Formally, a matroid consists of a **ground** set E (the input), and a "family" I of **independent** subsets of E (the roads):

$$E : Set\ \ \delta,\ \ I : Set\ \ (Set\ \ \delta)\ \text{ where }\ I \subseteq \rho E$$

for some element type δ. I is a subset of the power set of E; I is a set of subsets of E.

2 Properties

For I to be seen as the routes we can take to a solution, we need some closure properties. An independence system supports the potential routes from an empty set to a solution, and the augmentation property lets us grow from a smaller solution to a larger one:

$$\begin{aligned}
&\forall i \in I.\ i \subseteq E && \text{(MATROID-SUBSET)}\\
&\forall i \in I.\ \rho i \subseteq I && \text{(MATROID-HEREDITARY)}\\
&\forall i, j \text{ where } |i| < |j| \in I.\\
&\quad \exists e \in i - j.\ i \cup \{e\} \in I && \text{(MATROID-GROWTH)}
\end{aligned}$$

MATROID-SUBSET and MATROID-HEREDITARY satisfy what is known as an "Independence System", a "Hereditary Subset System", or "Abstract Simplical Complex". This gives us the knowledge that for every subset $i \subseteq E$ in I, any *smaller variant* of i is also in I. MATROID-GROWTH is the "Augmentation Property" for matroids, letting us grow from a smaller element to a larger element, by implementing atomic inclusion of additional elements via union.

With these properties, not only can we leverage matroids as a means to check if a subset of E is a potential solution (in I), but we can also easily *add to* our solution, and see if that is also satisfactory. The *greedily* function relies on this, inductively proving that it strongly normalizes to a maximum "weight". That is, when using a *weigh* function as a metric for each element of type δ in E, *greedily* can find the subset of E that maximizes the total of this weight metric.

This is the nature of I - it is exhaustive in every opportinity that a subset of E can have to become a larger solution - all subsets of a potential solution will be a potential solution, and through augmentation we can approach a larger solution from a smaller one.

3 Weights

Matroids let us find optimal subsets of a particular set, with respect to a particular *metric*. We need some form of *weigh* function, which gives a measure for each element in E:

$$weigh \ : \ \delta \to \psi$$

For our purposes, we will also need some way to get a "total" value of any number of ψ values - in *any order*, too. This means that we need a binary $\otimes$ function, which satisfies an *abelian semigroup* over ψ:

$$\begin{aligned}
&\otimes : \psi \to \psi \to \psi\\
&\forall a, b \in \psi.\ a \otimes b \equiv b \otimes a && \text{(PSI-COMM)}\\
&\forall a, b, c \in \psi.\ (a \otimes b) \otimes c \equiv a \otimes (b \otimes c) && \text{(PSI-ASSOC)}
\end{aligned}$$

In the circumstance that we want to find the subset of E that *maximizes* the *total*[1], then ψ obviously needs to form a partial order. If we are to greedily

[1]Where *total* is akin to *concat* from Haskell, in any order.

find our maximum subset, then $\otimes$ should also be strictly increasing value when combining terms:

$$\forall p \in \psi.\ p \leq p \qquad \text{(PSI-REFL)}$$
$$\forall p, q \in \psi.\ p \leq q \wedge q \leq p \Rightarrow p \equiv q \qquad \text{(PSI-ANTI)}$$
$$\forall p, q, r \in \psi.\ p \leq q \wedge q \leq r \Rightarrow p \leq r \qquad \text{(PSI-TRANS)}$$
$$\forall p, q \in \psi.\ p \leq p \otimes q \qquad \text{(PSI-INC)}$$

Where PSI-REFL, PSI-ANTI and PSI-TRANS form the partial order, and PSI-INC shows that the total of any number terms should be larger than or equal to the total of any subset of those terms.

ψ then serves as an auxiliary type, with enough behaviour to ensure that *greedilyMax* will find our maximum subset.

4 Optimization

greedilyMax works by making successive "*pivots*", moving the maximum element from the ground set to the temporary result (only if the new result is in I). Put simply, *pivotMax* mutates the temporary result directly, and is stateful in the ground set:

```
pivotMax   : ( MonadState (Set δ) m
             , MonadReader (Set (Set δ)) m
             ) ⇒ Set δ → m (Set δ)
pivotMax x  =  do e ← takeMaximumWith weigh
                  i ← ask
                  if x ∪ e ∈ i then return x ∪ e
                               else return x
```

greedilyMax is then the fixpoint of *pivotMax*:

```
      greedilyMax  :  (Set δ, Set (Set δ)) → Set δ
greedilyMax (e, i)  =  runReader (runStateT (
                          almostFixM (null =<< get) pivotMax ∅
                       ) e) i
```

In this way, *greedilyMax* finds the maximum subset.

Theorem 1 (Greedily Maximum)**.** *greedilyMax finds the subset of E in I such that total of weigh mapped over the subset is maximal compared to all elements in $I \subseteq \rho E$.*

Proof. By induction on □