Accelerated Greedy Algorithms for Maximizing Submodular Set Functions

Michel Minoux, 1978

Gatsby Computational Neuroscience Unit, UCL

19 August, 2014

The Upshot

A simple and clever idea which is more widely applicable than is billed in the paper.
You might be able to use it in your research!

Outline

Set Functions

Consider a set $E = \{e_1, \ldots, e_n\}$ (where n is finite) and a function $f(S)$, $f : \mathcal{P}(E) \to \mathbb{R}$, where $\mathcal{P}(E)$ is the power set (set of all subsets) of E. f is thus called a set function.

We might want to find $S^* = \operatorname{argmax}_{S \in \mathcal{P}(E)} f(S)$.

This problem is in general combinatorially hard, because $f(S)$ may be arbitrary for each $S \in \mathcal{P}(E)$, and so these subsets must be exhaustively enumerated.

Submodular Set Functions

We hope real problems have benign structure, letting us find good, non-combinatorial solutions.
One case of interest is if $f(S)$ is *submodular*.

Definition (Submodular)

A set function $f : \mathcal{P}(E) \rightarrow \mathbb{R}$ is said to be *submodular* iff $\forall A \in \mathcal{P}(E), B \subset A, e_i \in E/A$,

$$f(A \cup e_i) - f(A) \leq f(B \cup e_i) - f(B).$$

This may be summarized as a property of "diminishing gains;" much like concavity.

Examples

- $f(S) = \sum_{s \in S} g(s)$ (Trivial, but submodular)
- $f(S) = \frac{|S|(|S|-1)}{2} + \sum_{s \in S} g(s)$ (Slightly less trivial)
- $f(S) = I(g; \mathbf{y})$ where $\mathbf{y} = \{y_1 = g(s_1) + \eta_1, \ldots\}$, $\eta_i \sim \mathcal{N}$, and g is a latent, Gaussian distributed vector.
- Minimum Spanning Tree: Consider a connected graph $G = [X, U]$, with node set X and edge set U. Let $w(u)$, $w : U \to \mathbb{R}$ be the (fixed) weight of each edge. Define $w(S) \triangleq \sum_{u \in S} w(u)$ for each $S \in \mathcal{P}(U)$. Further, define

$$f(S) = \begin{cases} -w(U/S) & \text{if } G_S = [X, U/S] \text{ is connected} \\ -\infty & \text{otherwise.} \end{cases}$$

f is submodular because $f(u \cup S) - f(S) = w(u)$ if G_S and $G_{S \cup u}$ are connected, and $-\infty$ if G_S is connected and $G_{S \cup u}$ is not (ignoring the case where G_S is not connected).

Contrasting Example

Consider a string of bits, where bits are in triplets $[e_1, e_2, e_3]$, where e_1, e_2, i.i.d. $\sim$ Uniform$(0, 1)$, carry the information and e_3 is a checksum, $e_3 = e_1 + e_2$.

$$f(S) \triangleq I(e_3; S),$$

where $E = \{e_1, e_2\}, S \in \mathcal{P}(E)$.
$f(\{e_1\}) = f(\{e_2\}) = 0$, but $f(\{e_1, e_2\}) = 1$ bit, so adding e_1 or e_2 to $S = \{\}$ produces less gain in f than adding e_1 or e_2 to $\{e_2\}$ or $\{e_1\}$; f is thus not submodular.

Greedy Algorithms

- In general, maximizing over combinatorial sets $\Longrightarrow$ combinatorial complexity.
- Greedy heuristic: iteratively add to S^k the element within the set E/S^k which provides the greatest gain in the objective function.
  - Provides an important alternative to doing the full optimization, but often no theoretical guarantees.
  - If objective function is submodular, guarantees may be obtained (e.g., Krause 2005, 2008).

Simple Greedy Algorithm

Standard greedy algorithm (SG)

(a) Take $S^\circ = \emptyset$; itération $k = 0$

(b) at step k, S^k is the current solution of cost $f(S^k)$ and $|S^k| = k$

(c) for all $e_i \in E - S^k$, compute :

$$\Delta^k(e_i) = f(S^k + \{e_i\}) - f(S^k).$$

(d) select e_{io} such that $\Delta^k(e_{io}) = \underset{e_i \in E-S^k}{\mathrm{Max}} \Delta^k(e_i)$.

if $\Delta^k(e_{io}) \leqslant 0$ Terminated = the current solution S^k is (loccaly) optimal.

otherwise :

(e) Set $S^{k+1} \leftarrow S^k + \{e_{io}\}$

$k \leftarrow k + 1$

and return to (b).

An Additional Problem

- The problem of actually determining which element of the decision set is truly the greedy choice may be computationally expensive.
- If there are n elements in E, we may have to evaluate $f(S \cup e_i), \forall e_i \in E/S$, once for each iteration of the algorithm (potentially n times); this scales as $O(n^2 c)$, where c is (assumed fixed) the cost of evaluating $f(S \cup e_i)$.
- Adaptive Greedy (AG) algorithms improve upon this.

Adaptive Greedy Algorithm

The accelerated greedy algorithm (AG)

(a) Take $S^\circ = \emptyset$ as a starting solution
step $k = 0$

(b) for every $e_i \in E$, compute :

$$\Delta(e_i) = f(\{e_i\}) - f(\emptyset)$$

(c) At step k, let S^k be the current solution, of cost $f(S^k)$
Select $e_{io} \in E - S^k$ such that :

$$\Delta(e_{io}) = \underset{e_i \in E - S^k}{\text{Max}} \left\{\Delta(e_i)\right\}$$

If e_{io} has already been selected once at step k set : $\delta = \Delta(e_{io})$ and go to (e)

(d) compute :

$$\delta = f(S^k + \{e_{io}\}) - f(S^k)$$

and set : $\Delta(e_{io}) \leftarrow \delta$

if $\delta < \underset{\substack{e_i \in E - S^k \\ e_i \neq e_{io}}}{\text{Max}} \left\{\Delta(e_i)\right\}$

return to (c)

otherwise :

(e) if $\delta \leq 0$ STOP : solution S^k is (locally) optimal - Otherwise ($\delta > 0$) :

(f) Set : $S^{k+1} \leftarrow S^k + \{e_{io}\}$

$\Delta(e_{io}) = 0$

$k \leftarrow k + 1$

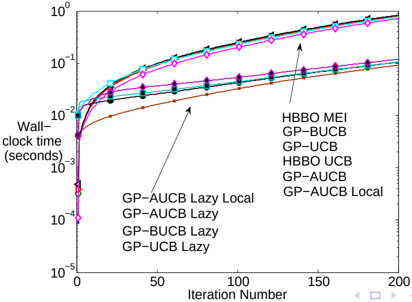
and return to (c).

If maximizer at each iteration is unique, same results as SG.

Surely that won't matter that much!

But it does!

- Minoux (1978) claims a reduction of hours to minutes (50-100x) on a test problem.
- And our results (Desautels, Krause, & Burdick, 2012 & 2014) using a similar technique show big speedups as well:



Minoux also provides a proof that the AG algorithm is computationally optimal within the class of algorithms which iteratively consider additions of only one element of E/S to S.

In conclusion:

- Minoux's trick is simple and easy, but exact.
- It's applicable whenever you're trying to greedily optimize a set function where the gain for adding $e_i \in E/S$ is strictly non-increasing with respect to S.