Multiplicative weights method:
A meta algorithm with applications to linear and semi-definite programming

Sanjeev Arora
Princeton University

Based upon:
Fast algorithms for Approximate SDP [FOCS ‘05]
√log(n) approximation to SPARSEST CUT in Õ(n²) time [FOCS ‘04]
The multiplicative weights update method and it’s applications [’05]
See also recent papers by Hazan and Kale.
Multiplicative update rule (long history)

\[ n \text{ agents} \quad \text{weights} \]

\[ w_1, w_2, \ldots, w_n \]

Update weights according to performance:

\[ w_i^{t+1} = w_i^t (1 + \varepsilon \cdot \text{performance of } i) \]

Applications: approximate solutions to LPs and SDPs, flow problems, online learning (boosting), derandomization & Chernoff bounds, online convex optimization, computational geometry, metric embeddings, portfolio management… (see our survey)
Simplest setting – predicting the market

N “experts” on TV

1$ for correct prediction

0$ for incorrect

Can we perform as good as the best expert?
Weighted majority algorithm [LW ‘94]

“Predict according to the weighted majority.”

Multiplicative update (initially all $w_i = 1$):
- If expert predicted correctly: $w_i^{t+1} \sim w_i^t$
- If incorrectly, $w_i^{t+1} \sim w_i^t(1 - \varepsilon)$

Claim: #mistakes by algorithm $\frac{1}{4} 2(1+\varepsilon)(#mistakes by best expert)$
- Potential: $\phi_t = \text{Sum of weights}= \sum_i w_i^t$ (initially $n$)
- If algorithm predicts incorrectly $\phi_{t+1} \cdot \phi_t - \varepsilon \phi_t / 2$
- $\phi_T \cdot (1-\varepsilon/2)^{m(A)} n$
- $\phi_T \cdot (1-\varepsilon)^{m_i}$
- $m(A) = # \text{ mistakes by algorithm}$
- $m_i$
Generalized Weighted majority

[A., Hazan, Kale ‘05]

n agents

Set of events (possibly infinite)

expert i

payoff

M(i,j)
Generalized Weighted majority
[AHK ‘05]

\( n \) agents

Set of events (possibly infinite)

\[ \begin{align*}
  p_1 & \quad \text{Algorithm: plays distribution on experts (} p_1, \ldots, p_n \text{)} \\
  p_2 \\
  \vdots \\
  p_n
\end{align*} \]

Payoff for event \( j \): \( \sum_i p_i M(i,j) \)

Update rule:
\[ p_i^{t+1} \overset{\cdots}{\longrightarrow} p_i^t (1 + \epsilon \cdot M(i,j)) \]

Claim: After \( T \) iterations,

Algorithm payoff \( \geq (1-\epsilon) \) best expert – \( O(\log n / \epsilon) \)
Game playing, Online optimization

Lagrangean relaxation

Gradient descent

Chernoff bounds

Games with Matrix Payoffs

Fast soln to LPs, SDPs
Common features of MW algorithms

- “competition” amongst n experts
- Appearance of terms like

\[ \exp(-\sum_t (\text{performance at time } t)) \]

- Time to get $\varepsilon$-approximate solutions is proportional to $1/\varepsilon^2$. 
Application 1: Approximate solutions to LPs (“Combinatorial”)

- Plotkin Shmoys Tardos ’91
- Young’97
- Garg Koenemann’99
- Fleischer’99

MW Meta-Algorithm gives unified view
Solving LPs (feasibility)

\[
\begin{align*}
\mathbf{w}_1 & \quad \mathbf{a}_1 \cdot \mathbf{x} & \quad \mathbf{b}_1 - \varepsilon \\
\mathbf{w}_2 & \quad \mathbf{a}_2 \cdot \mathbf{x} & \quad \mathbf{b}_2 - \varepsilon \\
& \quad \vdots & \quad \vdots \\
& \quad \vdots & \quad \vdots \\
\mathbf{w}_m & \quad \mathbf{a}_m \cdot \mathbf{x} & \quad \mathbf{b}_m - \varepsilon \\
\mathbf{x} & \quad 2\mathbf{P}
\end{align*}
\]

\[
\sum_k \mathbf{w}_k(\mathbf{a}_k \cdot \mathbf{x} - \mathbf{b}_k), 0
\]

\[
\mathbf{x} \quad 2\mathbf{P}
\]

\[\mathbf{P} = \text{convex domain}\]
Solving LPs (feasibility)

\[
\begin{align*}
[1 - \varepsilon (a_1 \cdot x_0 - b_1) / \rho] & \leq w_1 \\
[1 - \varepsilon (a_2 \cdot x_0 - b_2) / \rho] & \leq w_2 \\
\vdots & \\
[1 - \varepsilon (a_m \cdot x_0 - b_m) / \rho] & \leq w_m \\
\end{align*}
\]

Final solution = Average \( x \) vector

\[
\sum_k w_k (a_k \cdot x - b_k) , 0
\]

Oracle

\[
\rho = \text{width}
\]
Performance guarantees

- In $O(\rho^2 \log(n)/\varepsilon^2)$ iterations, average $x$ is $\varepsilon$ feasible.
- Packing-Covering LPs: [Plotkin, Shmoys, Tardos ’91]
  - $9? \times 2 \mathbb{P}$:
    - $j = 1, 2, \ldots, m$: $a_j \not\in x \cup 1$
  - Want to find $x \times 2 \mathbb{P}$ s.t. $a_j \not\in x \cup 1 - \varepsilon$
  - Assume: $8 \times 2 \mathbb{P}$: $0 \cdot a_j \not\in x \cdot \rho$
  - MW algorithm gets $\varepsilon$ feasible $x$ in $O(\rho \log(n)/\varepsilon^2)$ iterations
Connection to Chernoff bounds and derandomization

Deterministic approximation algorithms for 0/1 packing/covering problem \textit{a la} Raghavan-Thompson

- Solve LP relaxation of integer program. Obtain soln \((y_i)\)

- Randomized Rounding \(O(\log n)\) times:
  \(x_i \overset{\text{A}}{\leftarrow} 1\) w/ prob. \(y_i\)
  \(x_i \overset{\text{A}}{\leftarrow} 0\) w/ prob. \(1 - y_i\)

- Derandomize using pessimistic estimators
  \(\exp(\sum t \in f(y_i))\)

Young [95] “Randomized rounding without solving the LP.” MW update rule mimics pessimistic estimator.
Application 2:

Semidefinite programming (Klein-Lu’97)

\[ a_1 \preceq x \preceq b_1 \]
\[ a_2 \preceq x \preceq b_2 \]
\[ \vdots \]
\[ a_m \preceq x \preceq b_m \]
\[ x \in \mathbb{P} \]

\[ \mathbb{P} = \{x: x \text{ is psd; } \text{tr}(x) = 1\} \]

Oracle: \( \max \sum_j w_j (a_j \preceq x) \) over \( \mathbb{P} \)

(eigenvalue computation!)

\( a_j \) and \( x \): symmetric matrices in \( \mathbb{R}^{n \times n} \)
Next few slides: New Results (AHK’04, AHK’05)

Key difference between efficient and not-so-efficient implementations of the MW idea: **Width management**.

(e.g., the difference between PST’91 and GK’99)
Solving SDP relaxations more efficiently [AHK’05]

<table>
<thead>
<tr>
<th>Problem</th>
<th>Using Interior Point</th>
<th>Our result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXQP (e.g. MAX-CUT)</td>
<td>Õ(n^{3.5})</td>
<td>Õ(n^{1.5}N/\varepsilon^{2.5}) or Õ(n^{3}/\alpha^{*}\varepsilon^{3.5})</td>
</tr>
<tr>
<td>HAPLOFREQ</td>
<td>Õ(n^{4})</td>
<td>Õ(n^{2.5}/\varepsilon^{2.5})</td>
</tr>
<tr>
<td>SCP</td>
<td>Õ(n^{4})</td>
<td>Õ(n^{1.5}N/\varepsilon^{4.5})</td>
</tr>
<tr>
<td>EMBEDDING</td>
<td>Õ(n^{4})</td>
<td>Õ(n^{3}/d^{5}\varepsilon^{3.5})</td>
</tr>
<tr>
<td>SPARSEST CUT</td>
<td>Õ(n^{4.5})</td>
<td>Õ(n^{3}/\varepsilon^{2})</td>
</tr>
<tr>
<td>MIN UNCUT etc</td>
<td>Õ(n^{4.5})</td>
<td>Õ(n^{3.5}/\varepsilon^{2})</td>
</tr>
</tbody>
</table>
Recall: issue of width

MW

\[ a_1 \preceq x \leq b_1 \]
\[ a_2 \preceq x \leq b_2 \]
\[ \vdots \]
\[ \vdots \]
\[ a_m \preceq x \leq b_m \]

Oracle

\[ \sum_k w_k (a_k \preceq x - b_k) \leq 0 \]
\[ x \in 2P \]

\[ \tilde{O}(\rho^2/\varepsilon^2) \] iterations to obtain \( \varepsilon \) feasible \( x \)

\[ \rho = \max_k |a_k \preceq x - b_k| \]

\( \rho \) is too large!!
Issue 1: Dealing with width

Oracle

\[ \sum_k w_k (a_k \cdot x - b_k), 0 \]

MW

\[ a_1 \cdot x, b_1 \]
\[ a_2 \cdot x, b_2 \]
\[ \vdots \]
\[ a_m \cdot x, b_m \]

- A few high width constraints
- Oracle: separation hyperplane for dual
- Can run ellipsoid/Vaidya
- \( \text{poly}(m, \log(\rho/\epsilon)) \) iterations to obtain \( \epsilon \) feasible \( x \)
Dealing with width (contd)

Hybrid of MW and Vaidya

\(\tilde{O}(\rho_L^2/\epsilon^2)\) iterations to obtain \(\epsilon\) feasible \(x\)

\(\rho_L \leq \rho\)

MW

\[ a_1 \parallel x \parallel b_1 \]
\[ a_2 \parallel x \parallel b_2 \]
\[ \vdots \]

Dual ellipsoid/Vaidya

\[ a_m \parallel x \parallel b_m \]

Oracle

\[ \sum_k w_k (a_k \parallel x - b_k) \parallel 0 \]
\[ x \parallel 2P \]
Issue 2: Efficient implementation of Oracle: fast eigenvalues via matrix sparsification

\[ O(\sqrt{n\sum_{ij}|C_{ij}|/\varepsilon}) \] non-zero entries

\[ k \cdot C - C' \cdot k \cdot \varepsilon \]

- Lanczos effectively uses sparsity of C
- Similar to Achlioptas, McSherry [’01], but better in some situations (also easier analysis)
Online games with matrix payoffs
(Satyen Kale’06)

Payoff is a matrix, and so is the “distribution” on experts!

Uses matrix analogues of usual inequalities

\[ 1 + x \cdot e^x \quad I + A \cdot e^A \]

Used (together with many other tricks) to solve “triangle inequality” SDPs in \( O(n^3) \) time.
O(n^2)-time algorithm to compute O(plog n)-approximation to SPARSEST CUT

(v/s O(n^{4.5}) using interior point methods)
Sparsest Cut

The sparsest cut:

\[ \alpha(G) = \frac{2}{5} \]

- O(log n) approximation [Leighton Rao ’88]
- O(p log n) approximation [A., Rao, Vazirani’04]
- O(p log n) approximation in O(n^2) time. (Actually, finds expander flows) [A., Hazan, Kale’05]
MW algorithm to find expander flows

- **Events** – \{(s,w,z) \mid \text{weights on vertices, edges, cuts}\}
- **Experts** – pairs of vertices \((i,j)\)
- **Payoff:** (for weights \(d_{ij}\) on experts)
  \[
P_{ij} d_{ij} (s_i + s_j + l_{ij} i z_{ij})
  \]

Fact: If events are chosen optimally, the distribution on experts \(d_{ij}\) converges to a demand graph which is an “expander flow” [by results of Arora-Rao-Vazirani ’04 suffices to produce approx. sparsest cut]
Faster algorithms for online learning and portfolio management

(Agarwal-Hazan’06, Agarwal-Hazan-Kalai-Kale’06)

- Framework for online optimization inspired by Newton’s method (2\textsuperscript{nd} order optimization).
  (Note: MW $\frac{1}{4}$ gradient descent)

- Fast algorithms for Portfolio management and other online optimization problems
Open problems

- Better approaches to width management?

- Faster run times?

THANK YOU
Connection to Chernoff bounds and derandomization

- Deterministic approximation algorithms *a la* Raghavan-Thompson
- Packing/covering IP with variables $x_i = 0/1$
  
  \[
  \text{9? x 2 P: } \sum_{j=2}^{8} f_j(x) \geq 0
  \]
  
  - Solve LP relaxation using variables $y_i \in [0, 1]$
  - Randomized rounding: w.p. $y_i$
    \[
    x_i = \begin{cases} 
      1 & \text{w.p. } y_i \\
      0 & \text{w.p. } 1 - y_i
    \end{cases}
    \]

- Chernoff: $O(\log m)$ sampling iterations suffice
Derandomization [Young, ’95]

- Can derandomize the rounding using $\exp(t\sum_j f_j(x))$ as a pessimistic estimator of failure probability
- By minimizing the estimator in every iteration, we mimic the random expt, so $O(\log m)$ iterations suffice
- The structure of the estimator obviates the need to solve the LP: *Randomized rounding without solving the Linear Program*
- Punchline: resulting algorithm is the MW algorithm!
Weighted majority [LW ‘94]

- If lost at $t$, $\phi_{t+1} \cdot (1 - \frac{1}{2} \varepsilon) \phi_t$

- At time $T$: $\phi_T \cdot (1 - \frac{1}{2} \varepsilon) \#\text{mistakes} \phi_0$

\[
(1_i \cdot ")^{m_i} = w_i^T \cdot w_i^T = \mathbb{C}_T
\]

- Overall:

\[
\#\text{mistakes} \cdot \log(n)/\varepsilon + (1 + \varepsilon) m_i
\]
Vectors $\mathbf{a}_j$ and $\mathbf{x}$: symmetric matrices in $\mathbb{R}^{n \times n}$

- $\mathbf{x} \succeq 0$

Assume: $\text{Tr}(\mathbf{x}) \cdot 1$

Set $\mathbf{P} = \{\mathbf{x}: \mathbf{x} \succeq 0, \text{Tr}(\mathbf{x}) \cdot 1\}$

Oracle: $\max \sum_j w_j (\mathbf{a}_j \& \mathbf{x})$ over $\mathbf{P}$

Optimum: $\mathbf{x} = \mathbf{v}\mathbf{v}^T$ where $\mathbf{v}$ is the largest eigenvector of $\sum_j w_j \mathbf{a}_j$
Efficiently implementing the oracle

- Optimum: \( x = vv^T \)
  - \( v \) is the largest eigenvector of some matrix \( C \)
- Suffices to find a vector \( v \) such that \( v^T Cv \geq 0 \)
- Lanczos algorithm with a random starting vector is ideal for this
- Advantage: uses only matrix-vector products
  - Exploits sparsity (also: sparsification procedure)
- Use analysis of Kuczynski and Wozniakowski [’92]