Efficient Memoization for Approximate Function Evaluation over Sequence Arguments

AAIM 2014

Tamal Biswas and Kenneth W. Regan
University at Buffalo (SUNY)

9 July 2014

The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).

The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.

The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals.

The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for M = millions of x.
3. Each eval of f is **expensive**. Many repetitive evals.
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:

The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals.
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
    - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.

The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals.
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.
   - Could be non-linear but well-behaved, e.g., logistic regression.

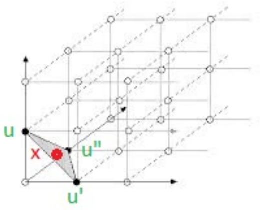
The Problem

1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals.
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.
   - Could be non-linear but well-behaved, e.g., logistic regression.
5. Two other helps: f is smooth and bounded.

The Problem

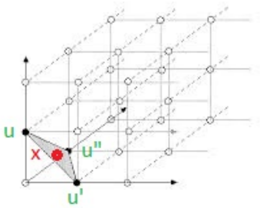
1. Space of points $x = (x_1, \ldots, x_N)$, where N is not small ($N \approx 50$).
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals.
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.
   - Could be non-linear but well-behaved, e.g., logistic regression.
5. Two other helps: f is smooth and bounded.
6. And we need good approximation to $\mu(\cdots)$ (only) under distributions $D(x)$ controlled by a few model-specific parameters,

The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

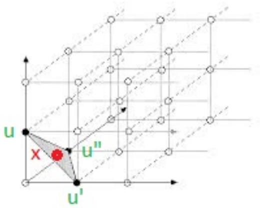
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*

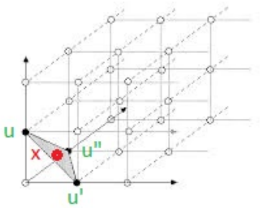
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$.

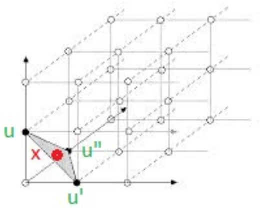
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$. *Seems better. But N is not small.*

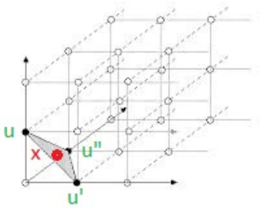
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$. *Seems better. But N is not small.*
3. Use any neighbor u and do Taylor expansion.

The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$. *Seems better. But N is not small.*
3. Use any neighbor u and do Taylor expansion.

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 50x data!

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 50x data!
3. Main Question: Can we "cheat" by shortcutting the partials?

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 50x data!
3. Main Question: Can we “cheat” by shortcutting the partials?
4. If f were linear, obviously $\frac{\partial f}{\partial x_i} = \text{constant}$.

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 50x data!
3. Main Question: Can we "cheat" by shortcutting the partials?
4. If f were linear, obviously $\frac{\partial f}{\partial x_i} = \text{constant}$.
5. What if the grid is warped "similarly" to f?

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \ldots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \ldots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E
4. Defines *fallible agent* $P(s, c, \ldots)$.

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \ldots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E
4. Defines *fallible agent* $P(s, c, \ldots)$.
5. Main Output: Probabilities $p_{t,i}$ for $P(s, c, \ldots)$ to select option i at time t.

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \dots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E
4. Defines *fallible agent* $P(s, c, \dots)$.
5. Main Output: Probabilities $p_{t,i}$ for $P(s, c, \dots)$ to select option i at time t.
6. Derived Outputs:
   - Aggregate statistics: *move-match* MM, *average error* AE, ...
   - Projected confidence intervals for those statistics.
   - "Intrinsic Performance Ratings" (IPR's).

How the Model Operates

1. Use analysis data and parameters $s, c, \ldots$ to compute "perceived inferiorities" $x_i \in [0.0, 1.0]$ of each of N possible moves. Let $a_i = 1 - x_i$.

$$(x_1 = 0.0 \leq x_2 \leq x_3 \leq \cdots \leq x_N) \equiv (a_1 = 1.0 \geq a_2 \geq \cdots \geq a_N \approx 0)$$

How the Model Operates

1. Use analysis data and parameters $s, c, \ldots$ to compute "perceived inferiorities" $x_i \in [0.0, 1.0]$ of each of N possible moves. Let $a_i = 1 - x_i$.

$$(x_1 = 0.0 \leq x_2 \leq x_3 \leq \cdots \leq x_N) \equiv (a_1 = 1.0 \geq a_2 \geq \cdots \geq a_N \approx 0)$$

2. For a fixed function h, solve $\frac{h(p_i)}{h(p_1)} = a_i$ subject to $\sum_{i=1}^{N} p_i = 1$.

How the Model Operates

1. Use analysis data and parameters $s, c, \ldots$ to compute "perceived inferiorities" $x_i \in [0.0, 1.0]$ of each of N possible moves. Let $a_i = 1 - x_i$.

$$(x_1 = 0.0 \leq x_2 \leq x_3 \leq \cdots \leq x_N) \equiv (a_1 = 1.0 \geq a_2 \geq \cdots \geq a_N \approx 0)$$

2. For a fixed function h, solve $\frac{h(p_i)}{h(p_1)} = a_i$ subject to $\sum_{i=1}^{N} p_i = 1$.
3. It suffices to compute p_1; then $p_i = h^{-1}(a_i h(p_1))$ is relatively easy.

How the Model Operates

1. Use analysis data and parameters $s, c, \ldots$ to compute "perceived inferiorities" $x_i \in [0.0, 1.0]$ of each of N possible moves. Let $a_i = 1 - x_i$.

$$(x_1 = 0.0 \leq x_2 \leq x_3 \leq \cdots \leq x_N) \equiv (a_1 = 1.0 \geq a_2 \geq \cdots \geq a_N \approx 0)$$

2. For a fixed function h, solve $\frac{h(p_i)}{h(p_1)} = a_i$ subject to $\sum_{i=1}^{N} p_i = 1$.
3. It suffices to compute p_1; then $p_i = h^{-1}(a_i h(p_1))$ is relatively easy.
4. Model uses $a_i = e^{-(\frac{\delta_i}{s})^c}$, where δ_i is the *scaled* difference in value between the best move and the i-th best move. Also fairly cheap.

How the Model Operates

1. Use analysis data and parameters $s, c, \ldots$ to compute "perceived inferiorities" $x_i \in [0.0, 1.0]$ of each of N possible moves. Let $a_i = 1 - x_i$.

$$(x_1 = 0.0 \leq x_2 \leq x_3 \leq \cdots \leq x_N) \equiv (a_1 = 1.0 \geq a_2 \geq \cdots \geq a_N \approx 0)$$

2. For a fixed function h, solve $\frac{h(p_i)}{h(p_1)} = a_i$ subject to $\sum_{i=1}^{N} p_i = 1$.
3. It suffices to compute p_1; then $p_i = h^{-1}(a_i h(p_1))$ is relatively easy.
4. Model uses $a_i = e^{-(\frac{\delta_i}{s})^c}$, where δ_i is the *scaled* difference in value between the best move and the i-th best move. Also fairly cheap.
5. But $y = p_1 = f(x)$ may require expensive iterative approximation.

How the Model Operates

1. Use analysis data and parameters $s, c, \ldots$ to compute "perceived inferiorities" $x_i \in [0.0, 1.0]$ of each of N possible moves. Let $a_i = 1 - x_i$.

$$(x_1 = 0.0 \leq x_2 \leq x_3 \leq \cdots \leq x_N) \equiv (a_1 = 1.0 \geq a_2 \geq \cdots \geq a_N \approx 0)$$

2. For a fixed function h, solve $\frac{h(p_i)}{h(p_1)} = a_i$ subject to $\sum_{i=1}^{N} p_i = 1$.
3. It suffices to compute p_1; then $p_i = h^{-1}(a_i h(p_1))$ is relatively easy.
4. Model uses $a_i = e^{-(\frac{\delta_i}{s})^c}$, where δ_i is the *scaled* difference in value between the best move and the i-th best move. Also fairly cheap.
5. But $y = p_1 = f(x)$ may require expensive iterative approximation.
6. Note f is *symmetric*, so x can be an ordered sequence.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_N$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_N} = y.$$

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_N$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_N} = y.$$

5. For $N = 1$, simply $p = \sqrt[b]{y}$. So this generalizes taking roots.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_N$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_N} = y.$$

5. For $N = 1$, simply $p = \sqrt[b]{y}$. So this generalizes taking roots.
6. We have $y = 1$ and $b_1 = 1$. Can this be solved **without** iteration?

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_N$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_N} = y.$$

5. For $N = 1$, simply $p = \sqrt[b]{y}$. So this generalizes taking roots.
6. We have $y = 1$ and $b_1 = 1$. Can this be solved **without** iteration?
7. Also: Current Expansion uses data for each depth d.

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.
4. **Axiom:** Influence of *poor* moves tapers off:

$$\text{As } x_i \longrightarrow 1.0, \quad \frac{\partial f}{\partial x_i} \longrightarrow 0.$$

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.
4. **Axiom:** Influence of *poor* moves tapers off:

$$\text{As } x_i \longrightarrow 1.0, \quad \frac{\partial f}{\partial x_i} \longrightarrow 0.$$

5. **Axiom:** Influence of *lower-ranked* moves becomes less:

$$\text{For } i < j, \ \frac{\partial f}{\partial x_j} < \frac{\partial f}{\partial x_i}.$$

(Not quite what was meant. . .)

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.
4. **Axiom:** Influence of *poor* moves tapers off:

$$\text{As } x_i \longrightarrow 1.0, \quad \frac{\partial f}{\partial x_i} \longrightarrow 0.$$

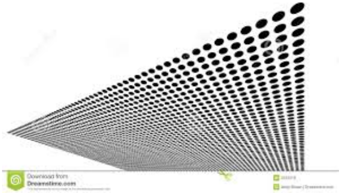
5. **Axiom:** Influence of *lower-ranked* moves becomes less:

$$\text{For } i < j, \ \frac{\partial f}{\partial x_j} < \frac{\partial f}{\partial x_i}.$$

   (Not quite what was meant. . .)
6. **"Universal Guess"**: In the first Taylor term, use

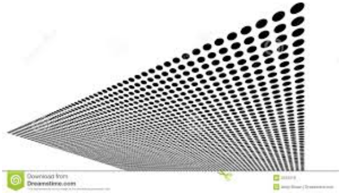
$$\frac{\partial f}{\partial x_i} \approx \frac{1}{i} a_i = \frac{1}{i}(1 - x_i).$$

The Tapered Grid



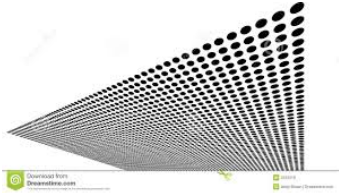
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.

The Tapered Grid



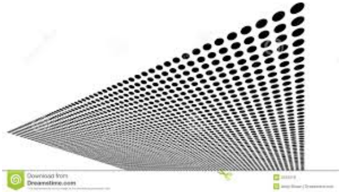
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.

The Tapered Grid



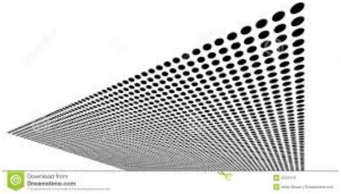
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Given $x = (x_1, \ldots, x_N)$, how to define "nearest" gridpoint $u = (u_1, \ldots, u_N)$?

The Tapered Grid



1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Given $x = (x_1, \ldots, x_N)$, how to define "nearest" gridpoint $u = (u_1, \ldots, u_N)$?
4. How to define a good bounding set $u, v, \ldots$?

The Tapered Grid



1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Given $x = (x_1, \ldots, x_N)$, how to define "nearest" gridpoint $u = (u_1, \ldots, u_N)$?
4. How to define a good bounding set $u, v, \ldots$?
5. How to make the computation of nearby gridpoints efficient?

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. "**Nearest Neighbor**" is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. "**Nearest Neighbor**" is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **"Nearest Neighbor"** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. "**Nearest Neighbor**" is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.
5. Like a heuristic for solving Knapsack problems.

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. "**Nearest Neighbor**" is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.
5. Like a heuristic for solving Knapsack problems.
6. Refinements which we have not yet fully explored include working backward from $i = N$ (too).

Strategies

Given $x = (x_1, x_2, \ldots, x_N)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. "**Nearest Neighbor**" is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.
5. Like a heuristic for solving Knapsack problems.
6. Refinements which we have not yet fully explored include working backward from $i = N$ (too).
7. Combine with "universal gradient" idea, or even ignore said idea.

Results So Far...

1. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

Results So Far. . .

1. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

2. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.

Results So Far...

1. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

2. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.
3. Results under D_ϵ are *good*: 3-place precision on $\mu(\dots)$ given 2-place to 1-place precision on grid.

Results So Far...

1. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

2. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.
3. Results under D_ϵ are *good*: 3-place precision on $\mu(\dots)$ given 2-place to 1-place precision on grid.
4. Results on real chess data...

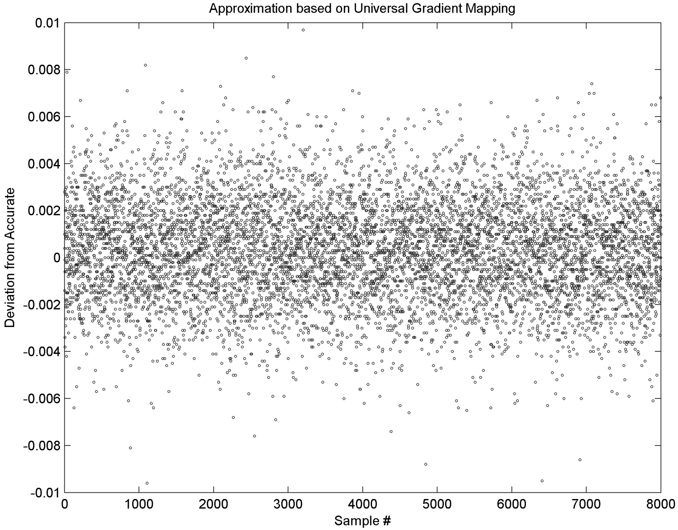
Results So Far. . .

1. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

2. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.
3. Results under D_ϵ are *good*: 3-place precision on $\mu(\dots)$ given 2-place to 1-place precision on grid.
4. Results on real chess data. . . still a work in progress.

Results for NN+UG



Results for Just NN