On-Line Algorithms and Competitiveness

Richard Borie
University of Alabama
June 20, 2005

Definitions

- Off-line algorithms run in batch mode
  - Know the entire sequence of requests in advance
- On-line algorithms are interactive
  - Must process each request before knowing future requests
- An on-line algorithm A is c-competitive if
  - For all sequences R = (R1, R2, R3, ..., Rm) of requests,
    \[ \text{cost}_A(R) \leq c \times \text{cost}_{OPT}(R) + b \]
  - OPT is an optimal off-line algorithm
  - b is a constant that is independent of R (usually 0)

Adversary

- When we try to prove that our on-line algorithm is c-competitive, we must assume an adversary knows how our algorithm works, and always issues requests R1, R2, R3, ..., Rm that will make our algorithm perform as poorly as it possibly can

Example: Rent or buy?

- Renting an item (such as tuxedo or skis) costs r dollars
- Purchasing the same item costs p dollars, where p > r
- For simplicity, assume p is a multiple of r
- Should you rent or buy?
- Assume you don't know whether or how often you'll ever need this item again
- Goal: minimize the total cost

Rent or buy? (continued)

- If you rent every time, then after m rentals, your ratio = m*r/p (unbounded)
- If you buy the first time, and then never use the item again, ratio = p/r
- Here is a (2-r/p)-competitive algorithm:
  - If (total_so_far + r < p) then rent else buy
- Why is this (2-r/p)-competitive?
  - If OPT < p then we pay exactly OPT
  - Otherwise we pay exactly 2*p-r and OPT = p

Rent or buy? (continued)

- There is no c-competitive on-line algorithm for this problem for any c < 2-r/p
  - The algorithm that always rents has unbounded ratio
  - So suppose we rent exactly k times before buying
  - OPT = \( \min(p, r*(k+1)) \)
  - Ratio = \( \frac{r*(k+1)}{OPT} \)
  - If p <= r*(k+1), then ratio >= \( \frac{r*(k+1)}{p} \) >= \( \frac{2*p-r}{p} \)
  - If r*(k+1) <= p, then ratio >= \( \frac{r*(k+1)/r*(k+1)}{r*(k+1)} \) >= 1 + (p-r)/(r*(k+1)) >= 1 + (p-r)/p >= 2-r/p
Another example: paging

- Let $k = \#$ pages that can fit in the cache
- Least Frequently Used (LFU) heuristic is not $c$-competitive for any constant $c$
  - Suppose $k = 2$. Request $q$ accesses to page 1. Next request $2q$ accesses that alternate between pages 2 and 3.
  - $\text{cost}_{\text{LFU}}(R) \geq 2q = 2m/3 = \Omega(m)$
  - $\text{cost}_{\text{OPT}}(R) \leq 3$

Paging (continued)

- Least Recently Used (LRU) heuristic is $k$-competitive
  - Suppose LRU generates $z$ page faults at these times: $T_1, T_2, T_3, \ldots, T_z$
  - Consider any time $T_a$ such that $a \geq k+1$
  - At least $k+1$ different pages are accessed during the time interval $[T_{a-k}, T_a]$
  - So $\text{OPT}$ must generate at least one page fault during the interval $[T_{a-k}, T_a]$

Paging (continued)

- No on-line algorithm $A$ for paging can be $c$-competitive for any constant $c < k$
  - Suppose $k+1$ pages in the working set.
    Adversary always requests a page that is not in the cache.
  - $\text{cost}_A(R) = m$
  - $\text{cost}_{\text{OPT}}(R) \leq m/k$, because $\text{OPT}$ is omniscient and always discards a page that won’t be needed until farthest in future

Example: Self-organizing list

- List $L$ that contains $n$ items
- Operation $\text{Access}(x)$ has cost $\text{rank}(x)$, which is position of $x$ from head of list
- Optionally, we may choose to reorder $L$ at the end of any access operation by exchanging adjacent items
  - Each exchange incurs unit cost
  - Goal: minimize the total cost

Analysis

- Worst case:
  - Adversary always accesses the tail item of $L$
  - $\text{Cost} = \Omega(m * n)$ in worst case
- Average case:
  - Suppose item $x$ is accessed with probability $p(x)$
  - Expected cost $= \sum_{x \in L} p(x) * \text{rank}(x)$
  - The minimum occurs when $L$ is arranged in decreasing order with respect to $p(x)$
  - But we do not know the distribution in advance

Heuristic 1

- Single Exchange (SE):
  - After accessing $x$, exchange $x$ with its predecessor (if $x$ was not at the front of $L$)
    - $\text{Cost} = \text{rank}(x) + 1$
- Not $c$-competitive for any constant $c$:
  - Alternately request the last two items in $L$
    - $\text{Cost}_{\text{SE}}(R) = m*\lceil n/2 \rceil + \Omega(m*n)$
    - $\text{Cost}_{\text{OPT}}(R) \leq 2^2*2^n + (m/2)^1 + (m/2)^2 = O(m*n)$
Heuristic 2

- Frequency Count (FC):
  - Count the frequency of access for each item, and maintain L arranged in decreasing order by frequency
- Not c-competitive for any constant c:
  - Access each of the n items n consecutive times
    - $\text{Cost}_{\text{FC}}(R) \leq (n/2) \times n \times (n/2) = \Omega(n^3)$
    - $\text{Cost}_{\text{OPT}}(R) \leq n \times (2n + n-1) = O(n^2)$

Heuristic 3

- Move to Front (MTF):
  - After accessing x, move x to front of L using exchanges
    - Cost = $2 \times \text{rank}(x) - 1$
  - Want to show that MTF is c-competitive for some constant c

MTF is 4-competitive

- Theorem: MTF is 4-competitive for self-organizing lists.
- Proof:
  - Let $L_k$ denote MTF's list after $k$th access
  - Let $L'_k$ denote OPT's list after $k$th access
  - Let $c_k$ = MTF's cost for $k$th operation
  - Let $c'_k$ = OPT's cost for the $k$th operation
    - $\Phi(L_k) = 2 \times (\text{number of inversions at time } k)$
    - Example:
      - $L_k = <E,C,A,D,B>$
      - $L'_k = <C,D,A,B,E>$
      - Inversions: (E,C),(E,A),(E,D),(E,B),(A,D)
      - $\Phi(L_k) = 2 \times 5 = 10$
    - Note:
      - $\Phi(L_0) = 0$ for every $k$
      - $\Phi(L_0) = 0$ if MTF and OPT start with the same list

- An inversion is a pair of items $(x,y)$ where $x$ precedes $y$ in $L_k$ and $y$ precedes $x$ in $L'_k$
- Define a potential function $\Phi$:
  - $\Phi(L_k) = 2 \times (\text{number of inversions at time } k)$
  - Example:
    - $L_k = <E,C,A,D,B>$
    - $L'_k = <C,D,A,B,E>$
    - Inversions: (E,C),(E,A),(E,D),(E,B),(A,D)
    - $\Phi(L_k) = 2 \times 5 = 10$
  - Note:
    - $\Phi(L_k) = 0$ for every $k$
    - $\Phi(L_0) = 0$ if MTF and OPT start with the same list

- How much does $\Phi$ change per exchange?
  - Each exchange either creates or destroys one inversion
    - So $\Delta \Phi$ is either 2 or -2
  - Let $c^*_k =$ amortized cost of MTF for $k$th operation
    - $c^*_k = c_k + \Phi(L_k) - \Phi(L_{k-1})$
    - Note $\Phi(L_k) - \Phi(L_{k-1}) = 2 \times (\#\text{inversions created} - \#\text{inversions destroyed})$

- Group all items into 4 sets as follows:
  - A = items that precede x in $L_{k-1}$ and precede x in $L'_{k-1}$
  - B = items that precede x in $L_{k-1}$ and follow x in $L'_{k-1}$
  - C = items that follow x in $L_{k-1}$ and precede x in $L'_{k-1}$
  - D = items that follow x in $L_{k-1}$ and follow x in $L'_{k-1}$
- When x is moved to front
  - # inversions created = |A|
  - # inversions destroyed = |B|
- Each exchange by OPT creates at most one inversion
- Thus:
  \[ c_k^* \leq c_k + \Phi(L_k) - \Phi(L_{k-1}) \]
  \leq 2 \times (|A| + |B| + 1) + 2 \times (|A| - |B| + e_k)
  \leq 4 \times |A| + 2 + 2 \times e_k
  \leq 4 \times (c' + e_k)
  \leq 4 \times c'_k

Hence \( c_k^* \leq 4c'_k \), and therefore:

\[
\text{Cost}_{\text{MTF}}(R) = \sum_{1 \leq k \leq m} c_k \\
\leq \sum_{1 \leq k \leq m} (c_k^* + \Phi(L_{k-1}) - \Phi(L_k)) \\
\leq \sum_{1 \leq k \leq m} (4c'_k + \Phi(L_0) - \Phi(L_m)) \\
\leq 4 \times \text{Cost}_{\text{OPT}}(R)
\]

[End of proof]

More examples

- Bin packing
  - First Fit (FF) is 2-competitive
    - Because no two bins can be combined
  - Decreasing First Fit (DFF) is off-line

- Financial markets (stocks, bonds)
  - Dollar cost averaging is a useful heuristic
  - No competitive algorithm known in general
  - Only if upper/lower bounds exist on prices