Bin Packing and Paging

Hsiang-Hsuan (Alison) Liu

1 Bin Packing

Bin Packing problem

Assume that there are unlimited number of size-1 bins. Given a sequence R of items $s_1, s_2, \cdots$ with each request $0 < s_i \leq 1$ represents the size of the item that must be assigned to some bin. The goal is to minimize the number of bins needed to assign all the items without exceeding the capacity of any bin.

As this problem is naturally an online problem, our aim is minimizing the competitive ratio. That is, we hope to minimize the ratio between the cost incurred by our algorithm to the offline optimal cost under the worst case.

1.1 The `FirstFit` Algorithm

A lot of algorithms for the Bin Packing problem have been studied. Many of these algorithms are "greedy-based": `FirstFit`, `NextFit`, `BestFit`, `WorstFit`, etc. In the following, we focus on the `FirstFit` algorithm:

`FirstFit` Algorithm

When an item arrives, pack it into the first bin that can accommodate it. If there is no such bin, open a new one.

Theorem 1. *`FirstFit` algorithm for the Bin Packing problem is 2-competitive when the optimal solution is large.*

Proof. Assume that the `FirstFit` algorithm uses k bins. Let B_i denote the total size of the items assigned to the i-th bin by `FirstFit`. We first claim the following:

Claim: there is at most one bin j with $B_j \leq 1/2$.

Followed by this Claim, at least $k-1$ bins has $B_i > 1/2$, and the total size of all items is at least $(k-1)\cdot(1/2)$. Because the optimal solution needs at least $\frac{\text{Total size of the items}}{1}$ bins to accommodate all items,

$$\text{OPT}(R) \geq \text{Total size of the items} > \frac{k-1}{2} \text{ for any set of items } R.$$

Since `FirstFIt` $= k$, for any instance R,

$$\texttt{FirstFit}(R) < 2\cdot\text{OPT}(R) + 1.$$

And hence the competitive ratio of `FirstFit` is 2 when the optimal cost is very large.

The last thing is to prove the correctness of the Claim. This can prove by contradiction: suppose on the contrary that there are two bins i and j, where bin j is opened later, such that $B_i \leq 1/2$ and $B_j \leq 1/2$. Consider the first item assigned to the bin j, it can be assigned to bin i by `FirstFit`. Therefore, a contradiction occurs and there is at most one half-empty bin.

□

Theorem 2. *`FirstFit` algorithm for the Bin Packing problem is at least 1.667-competitive.*

Proof. For some large k and small ε, consider the adversarial instance R consists of $6k$ size-$(\frac{1}{6} - 2\varepsilon)$ items, $6k$ size-$(\frac{1}{3} + \varepsilon)$ items, and $6k$ size-$(\frac{1}{2} + \varepsilon)$ items (in order). The items can be packed such that each bin consists of one item from each of the three groups of items and consumes $6k$ bins. Therefore, the optimal cost is at most $6k$.

`FirstFit` will pack the first group of items in 6, the second group of items in 2, and each item in the last group occupies one bin. In total, `FirstFit` uses $k + 3k + 6k = 10k$ bins. Therefore, $\frac{\texttt{FirstFit}(R)}{\mathrm{OPT}(R)} \geq \frac{10k}{6k} \approx 1.667$. $\square$

1.2 Bin Packing problem lower bound (Optional)

Theorem 3. *The* Bin Packing *Problem is at least* $\frac{4}{3}$*-competitive*

Proof. We prove this by contradiction. Suppose on the contrary that there exists an online algorithm ALG that attains a competitive ratio $\frac{4/3}{-}\epsilon$. Consider the adversarial input R which first release m size-$(\frac{1}{2} - \epsilon)$ items. The optimal solution is $\frac{m}{2}$. Since ALG is $\frac{4}{3} - \epsilon$-competitive, $\mathrm{ALG}(R) < \frac{4}{3} \cdot \mathrm{OPT}(R) = \frac{4}{3} \cdot \frac{m}{2} = \frac{2m}{3}$. Let a_1 and a_2 denote the number of bins with 1 item and 2 items in the $\mathrm{ALG}(R)$ assignment, respectively. By the definition of a_1 and a_2, we know A) $a_1 + a_2 = \mathrm{ALG}(R)$, and B) $a_1 + 2 \cdot a_2 = m$. By A and B, $\mathrm{ALG}(R) = m - a_2$. Furthermore, by the assumption that ALG is $\frac{4}{3} - \epsilon$-competitive, $\mathrm{ALG}(R) = m - a_2 < \frac{2m}{3}$, and $a_2 > \frac{m}{3}$.

Now, release another m $\frac{1}{2} + \epsilon$ items. Let R' denote the new instance (with $2m$ items). For the new instance R', one can pack one $\frac{1}{2} - \epsilon$ item and one $\frac{1}{2} + \epsilon$ in one bin. Therefore, $\mathrm{OPT}(R') \leq m$. However, ALG has to build its new assignment upon the old assignment for the first m items in R. It can pack some $\frac{1}{2} + \epsilon$ items in some of the a_1 bins. Let x be the $\frac{1}{2} + \epsilon$ items that are NOT packed in a_1 bins (in this case, each of these jobs occupies one bin). The algorithm ALG now uses $a_1 + a_2 + x$ bins. As there are m $\frac{1}{2} + \epsilon$ items, $a_1 + x \geq m$. Since ALG is $(\frac{4}{3} - \epsilon)$-competitive, $\mathrm{ALG}(R') < \frac{4}{3} \cdot \mathrm{OPT}(R') = \frac{4}{3} \cdot m$. Therefore, $a_2 < \frac{m}{3}$. It contradicts to the fact that $a_2 > \frac{m}{3}$. Hence, the theorem is proven. $\square$

2 Paging

In the modern *virtual memory system*, there are two levels of memory. The first level is a slow *memory*, which is big but cannot be accessed directly. The second level is a fast *cache*, which can be accessed directly, but its size is very limited. The data is partitioned into equal-size memory units, *page*. Once a page p is requested, the system has to make p available in the cache (called *hit*). Otherwise, if the page is not in the cache (called *miss*), the system incurs one *page fault* and must copy the page from the memory to the cache. If p is not in the cache, and the cache is full at the moment when p is requested, the system has to decide which page in the cache to evict such that p can be copied to the cache. Formally, the Paging problem is defined as follows.

Paging problem

There are N pages $\{1, 2, 3, \cdots, N\}$ stored in the slow memory and a fast cache with size $k < N$. Given a sequence R of requests $r_1, r_2, \cdots$, where $r_i \in [1, N]$, the online algorithm incurs a page fault if the requested page is not in the cache. The goal is to serve the sequence R with minimum page faults.

Algorithms According to how the pages are evicted, there are some famous algorithms:

- `FIFO` (First-In-First-Out): When eviction is necessary, replace the page that has been in the cache the longest.
- `LIFO` (Last-In-First-Out): Replace the page that has been copied into the cache the most recently.

- `LFU` (Least-Frequently-Used): Replace the page that has been requested the least since starting of the requests.
- `LRU` (Least-Recently-Used): Replace the page whose most recent request was the earliest.
- `CLOCK` (CLOCK-replacement): An approximation to `LRU` where a single bit replaces the implicit time stamp of `LRU`.
- `LFD` (Longest-Forward-Distance): Replace the page whose next request is latest.

2.1 The `LFU` Algorithm

Theorem 4. *The `LFU` algorithm has unbounded competitive ratio.*

Proof. Let m be a large number and consider the following instance. First request page 1 for m times, page 2 for m times, $\cdots$, page $k-1$ for m times. Then, the adversary takes turns and requests page k, page $k+1$, page k, page $k+1$, $\cdots$ (repeated for m times).

`LFU` incurs one page fault for pages 1 to $k-1$. For the later requests on page k and page $k+1$, `LFU` incurs one page fault whenever they are requested. In total, *LFU* incurs $k-1+2m$ page faults.

However, one can evict one of the pages 1 to $k-1$ when the first time page $k+1$ is requested and incur total $k+1$ page faults. Therefore, $\frac{\texttt{LFU}(R)}{\text{OPT}(R)} \geq \frac{k-1+2m}{k+1}$. The competitive ratio grows with the instance size $m(k+1)$ and is unbounded. □

2.2 The `LRU` Algorithm

Theorem 5. *The `LFU` algorithm is at most $O(k)$-competitive.*

Proof. For any request sequence R, partition it into phases as follows:

- Phase 0 is empty
- Phase i is the maximal sequence following phase $i-1$ that contains at most k distinct page requests (that is, phase $i+1$ begins on the request that is the $k+1$-th distinct page)

Next, we have the following claims:

Claim (a): In phase i, `LRU` only incurs at most k page faults

Claim (b): For any i, OPT incurs at least 1 page fault among the period from the second request in phase i till the first request in phase $i+1$

Assume that there are p phases. By these two claims, we can show that

$$\frac{\texttt{LRU}(R)}{\text{OPT}(R)} \leq \frac{p \cdot k}{(p-1)\cdot 1} = \frac{(p-1)\cdot k + k}{p-1} = O(k).$$

Now we prove the two claims. For Claim (a), since `LRU` evicts the page that was used the longest time ago, it never evicts a page that was requested earlier in the same phase. Hence, `LRU` incurs at most k page faults in a phase as there are at most k distinct pages in each phase.

For Claim (b), consider the cache just after the first request r in phase i. At this moment, there are $k-1$ pages in the cache, not counting r. Since the first request in phase $i+1$ is different from any of the pages in phase i, there are k distinct requests in the interval from the second request in phase i till the first request in phase $i+1$. Hence, any feasible algorithm has to evict one page to accommodate these requests, and so does OPT. Therefore, OPT has at least one page fault in this interval.

□

2.3 Paging problem lower bound

Theorem 6. *The* Paging *Problem is at least* $\Omega(k)$*-competitive.*

Proof. Assume that the cache size is k. Consider any algorithm ALG and design the adversary as follows: first request pages $1, 2, 3, \cdots, k, k+1$. At this moment, ALG evicts a page i. Then, the adversary requests page i. The adversary repeatedly requests the page evicted by ALG for $n-1$ rounds. In this instance, each request incurs a page fault for ALG. Therefore, ALG costs n. Because there are only $k+1$ pages involved, OPT incurs at most 1 page fault per k pages. Therefore,

$$\frac{\mathrm{ALG}(R)}{\mathrm{OPT}(R)} \geq \frac{k+n}{k+n/k} \approx \Omega(k).$$

□

Corollary 1. *LRU algorithm is optimal for the* Paging *problem*