1 Introduction

In this chapter we briefly describe a number of empirical studies of competitive algorithms: applications of ski-rental including cache coherence, virtual circuit holding times and mobile computing, paging, routing and admission control.

An algorithm is said to be on-line if it receives its input in a sequence of requests, and must service each request as it arrives, without any knowledge of the future. Many fundamental problems in computer science are inherently on-line, including memory management, processor scheduling, load balancing and routing. A commonly used method of analyzing on-line algorithms is competitive analysis [23], where the performance of the on-line algorithm is compared to the performance of the optimal off-line algorithm.

An on-line algorithm is said to be \( c \)-competitive if on each input its performance is within a factor of \( c \) of the performance of the optimal off-line algorithm on that input. Competitive analysis of algorithms has advantages over other more traditional methods of evaluating algorithms: Worst-case analysis can be overly pessimistic, while average case analysis requires a statistical model of the input. It is difficult to devise realistic statistical models, since input patterns tend to change dynamically with applications and over time.

Note that competitive analysis is still a worst-case measure, since we ask that the algorithm have a cost at most \( c \) times optimal on every input. Another way of saying this, is that the performance of the algorithm must be within a factor of \( c \) on inputs generated by a worst-case adversary.
The cornerstone of competitive analysis, the paging problem, is of long-standing interest to the operating systems and architecture communities. The setting for this problem is a two-level store consisting of a fast memory (the cache) that can hold \( k \) pages, and a slow memory that can store \( n \) pages. The \( n \) pages in slow memory represent the virtual memory pages. A paging algorithm is presented with a sequence of requests to virtual memory pages. If the page requested is in fast memory (a hit), no cost is incurred; but if not (a fault), the algorithm must bring it into the fast memory at unit cost. The algorithm must decide which of the \( k \) pages currently in fast memory to evict in order to make room for the newly requested page.

Many other problems that have been studied by the competitive algorithms community have direct bearing on computer systems. In this article we briefly describe several empirical studies of competitive algorithms. We begin with a number of applications of the "ski-rental problem" including cache coherence, virtual circuit holding times and mobile computing. We then touch on experimental work on paging, and on routing and admission control. These performance studies lead to the following conclusions: First, algorithms optimized to work against the worst-case adversary tend not to work well in practice. Second, the standard competitive algorithms give insight leading to the "correct" algorithm. Finally, nearly all successful empirical studies use algorithms that adapt in some way to the input distributions.

2 Applications of ski rental

2.1 The ski rental problem

We begin by reviewing the ski rental problem.\(^1\) Suppose you are about to go skiing for the first time in your life. Naturally, you ask yourself whether to rent skis or to buy them. Renting skis costs, say, $30, whereas buying skis costs, say $300. If you knew how many times you would go skiing in the future (ignoring complicating factors such as inflation, and changing models of skis), then your choice would be clear. If you knew you would go at least 10 times, you would be financially better off by buying skis right from the beginning, whereas if you knew you would go less than 10 times, you would be better off renting skis every time.

Alas, the future is unclear, and you must make a decision nonetheless – you are faced with an on-line problem, and must deal with events (in this case ski trips) as they arrive without knowledge (or with only limited knowledge) of future events.

For the ski-rental problem, we readily observe that if an adversary determines exactly at what point you will never go skiing again, it can ensure that you pay at least twice what you could have paid, regardless of your strategy. Indeed, suppose your algorithm is to continue renting until you have skied \( j \) times and

\(^1\) This analogy was originally suggested by Larry Rudolph in the context of the work on competitive snoopy caching [16].