The Incremental Constraint of k-Server

by

Caelyn Burnham McAulay

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Online algorithms are characterized by operating on an input sequence revealed over time versus a single static input. Instead of generating a single solution, they produce a sequence of incremental solutions corresponding to the input seen so far. An online algorithm's ignorance of future inputs limits its ability to produce optimal solutions. The incremental nature of its solutions is also an obstacle. The two factors, referred to as the adversarial and incremental constraints respectively, can be differentiated by examining the corresponding incremental algorithm, which has knowledge of future inputs, but must still provide a competitive solution at each step.

We examine the k-server problem under this framework, and determine that its incremental constraint is bounded below by 2. We also improve the lower bound on the incremental constraint of the vertex cover problem from $\frac{7}{6}$ to $\frac{3}{2}$.

We introduce several new characterizations and extensions to the definition of the incremental constraint and introduce an alternate definition for the adversarial constraint.

Acknowledgements

I would like to thank Jonathan Buss for supervising and guiding the research presented in this thesis. I would also like to thank Alex Lópex-Oritz and Mark Sherry for interesting discussions and Alex López-Oritz and Ian Munro for serving as readers for this thesis.

Contents

Chapter 1

Introduction

Online algorithms (such as those for the k-server problem [29]) have been traditionally evaluated using the competitive ratio [33]. In the work of Sharp [32], it was shown that the competitive ratio of an online algorithm is composed of two distinct contributing factors. The first is that online algorithms have no knowledge of their future inputs. Consequently, an adversary can choose sequences of inputs on which a given online algorithm will perform poorly. The second, and generally overlooked factor, is that the solutions generated by online algorithms must build upon their earlier solutions. These two factors were respectively named the adversarial constraint and the incremental constraint.

To isolate the effects of the two factors on the competitive ratio, we study the incremental version of an online problem. Incremental algorithms have knowledge of the entire sequence of input; but their performance is evaluated at every step, instead of on the entire sequence. Since incremental algorithms are effected only by the incremental constraint, comparing the competitiveness of the two types of algorithms on the same problem allows us to gain a lower bound on the adversarial constraint of an online problem.

This thesis extends that approach to the k-server problem and also provides some criticisms of the manner in which the incremental and adversarial constraints are calculated. It formalizes some definitions and results concerning the relationship between incremental algorithms, online algorithms and offline algorithms. It also provides formal proofs of the unbounded competitiveness of the Greedy algorithm and the Retrospective algorithm for the k-server problem in the appendix.

Since its introduction in [29], the k-server problem has been the subject of the active research. While there exist more general online problems (e.g. the tasks systems of Borodin, Linial, and Saks [7, 8]), the k-server problem, with its natural and general definition and deep results has been a very influential problem.

The competitive ratio is, for some online problems, to gross a measure. It fails to distinguish between good, bad or mediocre online algorithms. The offline adversary is far more powerful than the online algorithm we are attempting to

evaluate. In addition to the advantage of foreknowledge, since the comparison is being made over all possible input sequences, the online algorithm is potentially being evaluated on those isolated perverse sequences on which it performs in a pessimal manner.

These problems with the competitive ratio have long been known and there have been many efforts to generate metrics that correct for them. Some try to increase the power of the online algorithm (such as giving it a finite amount of look ahead), while others try to weaken the adversary (such as the random-order performance ratio, which prevents the adversary from dictating the ordering of the input sequence).

The incremental constraint (or incremental ratio) attempts to level the playing field by the first method. An incremental algorithm, in contrast to an online algorithm, will know what the sequence of input is. However, unlike either an offline or an online algorithm, it is evaluated not just on the entire sequence; but at each step. This stricter means of evaluation might seem too great a handicap for incremental algorithms. Could an online algorithm use its looser form of evaluation to beat an incremental algorithm? Over all possible input sequences, this is not the case. Since online algorithms base their decisions only on what inputs have been seen, they cannot, in fact, beat an incremental algorithm on an entire problem space. A proof of this is given in the chapter on the background material for the incremental constraint.

In this thesis, we prove a lower bound of 2 on the incremental constraint of the standard k-server problem. For the more general problem of k-servers on undirected graphs, we include a proof that the incremental constraint is, like the competitive ratio, unbounded. By considering paging as a sub-problem of k-server, since its incremental constraint is 1 (i.e. non-existent), while its adversarial constraint is k (and thus in the case of 2-servers is 2), we show that the conventional means of calculating the adversarial and incremental constraints of a problem are misleading in the case of k-server.

In addition, we improve the lower bound on the incremental constraint of the vertex cover problem and prove that the global incremental ratio of a problem does not exceed its global competitive ratio.

1.1 Overview of Chapters

Chapter 2 will cover the background material on the incremental constraint. The definition of the incremental constraint and the incremental ratio will be presented and given a motivating example in the form of a bin-packing problem. Some of the complexity properties of incremental algorithms will be discussed. There will also be some original work in this chapter in the form of a proof that the incremental constraint will never exceed the competitive ratio of an online problem is presented. This chapter will also summarize the current results known about the incremental constraint of various online problems.

Chapter 3 will cover the background material on the k-server problem. The problem itself will be first formally defined and then a brief survey of the results

from its introduction to the present will be discussed, with some illustrative proofs sketched. These results will be divided into results pertaining to the general k-server problem and those pertaining to various special cases.

Chapter 4 will present the almost all of the new results of this thesis. First, the results concerning the lower bounds of the incremental constraint of the general k-server problem will be presented and proven. The next section addresses the incremental constraint of the k-server problem on directed graphs. There is then a proof of an improved lower bound on the incremental constraint of the vertex cover problem. The problem of adding an asymptotic factor to the incremental constraint is explored. Then there will be a discussion of the relationship between the incremental constraint and the adversarial constraint of online problems. In particular, the incremental constraint and the adversarial constraint for the 2-server problem will be analyzed. This chapter will conclude with discussion of potential directions for future work.

Chapter 2

The Incremental Constraint

2.1 Definition of the Incremental Constraint

2.1.1 Definitions and an Illustrative Example

The concept of incremental algorithms, the metrics by which they are evaluated, and their application to the study of online algorithms, all originates in the work of Sharp [32]. All definitions in this section come from that work.

To better illustrate what the incremental constraint is, we will working with the following concrete example. Consider an instance of the bin-packing problem, in which there are 12 items to be packed into bins (of size 1), 6 of size $\frac{2}{5}$ and 6 of size $\frac{3}{5}$. This problem can then be viewed in three ways: as offline problem (as in classic bin-packing), as an online problem and as an incremental problem. For both the online and incremental instances of this problem, an ordering on the items is required as well. We shall assume all 6 smaller items arrive first, followed by all 6 larger items. The objective, for both the offline and online version of the problem, is to use as few bins as possible. for the incremental case, the objective is more complicated, and is described below.

As an offline problem, the solution is straightforward, with 6 bins being needed, each containing one item of size $\frac{2}{5}$ and one item of size $\frac{3}{5}$.

In the online case, the online algorithm must assign each item, as they arrive one by one, to a bin, without knowledge of future items. While no online bin-packing algorithm (randomized or deterministic) has a competitive ratio better than 1.54 ([9],[34]), this problem instance is not a worst case sequence, and the Best-Fit algorithm (place each new item into the fullest bin into to which it can fit, open a new bin if this fails) has a competitive ratio of $\frac{3}{2}$. Its solution, consists of 9 bins: 3 with two items of size $\frac{2}{5}$ and 6 with one item of size $\frac{3}{5}$.

Before discussing the incremental case, we need to define what an incremental problem is. In general, a combinatorial optimization problem has a static input and a single corresponding solution. An incremental problem can be thought of as a series of combinatorial optimization problems, with inputs that are prefixes of later problems. An incremental algorithm then produces a series

of solutions, one for each problem, with solutions that build incrementally. In the case of the incremental bin-packing problem, we have a series of bin packing problems and our solutions cannot rearrange already packed bins.

Definition (Sharp [32]) An **incremental problem** is defined as a series of combinatorial optimization problems $\{P_1, P_2, P_3, ...\}$ with the *incremental constraint* that $P_i \subset P_{i+1}$ (or in the cases where the problems already consist of a sequences of inputs: P_i is a prefix of P_{i+1}). An **incremental algorithm** takes an instance of an incremental problem and produces a sequence of solutions $\{S_1, S_2, S_3, ...\}$ satisfying the *feasibility constraint* that S_i is a solution to problem P_i and the *incremental constraint* that $S_i \subset S_{i+1}$ (or in the cases where solutions are themselves sequences: S_i is a prefix of S_{i+1}).

How can the performance of an incremental algorithm be evaluated? One approach is the aggregate value function, which is defined as the sum of the costs of each of the partial solutions.

Definition (Sharp [32]) The **aggregate value** function is defined for a given instance of an incremental problem $\{P_1, P_2, P_3, ...\}$ and corresponding solution $\{S_1, S_2, S_3, ...\}$ as $\sum_i \mathrm{cost(S_i)}$.

The cost function will vary from problem to problem. In the case of the bin-packing problem, it is defined as the number of bins used. Thus $\mathrm{cost(S_i)}$ is defined as the number of bins being used by solution S_i.

Under this metric, the offline bin-packing algorithm has a value of 47, while the online Best-Fit algorithm has a value of 51 (see diagram on the next page, in which, for example, OPT uses 1 bin on the first step, 2 on the second step and so forth, for a total of $1 + 2 + 3 + 4 + 5 + 6 + 6 + 6 + 6 + 6 + 6 + 6 = 47$). However, this metric does not take the balance amongst the solutions into the account. While the online Best-Fit algorithm has a slightly higher aggregate value, its solution ends up using $\frac{3}{2}$ as many bins on the sixth step. A better metric can be defined by taking the cost of each partial solution and comparing it against the optimal solution at that step. Minimizing this value is called the min-ratio problem for incremental algorithms.

Definition (Sharp[32]) The **incremental ratio** for a given instance of an incremental problem $\{P_1, P_2, P_3, ...\}$ and corresponding solution $\{S_1, S_2, S_3, ...\}$ is $\max_i \frac{\mathrm{cost(S_i)}}{\mathrm{cost}(OPT(\mathrm{P_i}))}$, where OPT is the optimal offline algorithm for the given combinatorial optimization problem.

We note that the above definitions have assumed a combinatorial minimization problem. These definitions can be trivially changed to apply to combinatorial maximization problems.

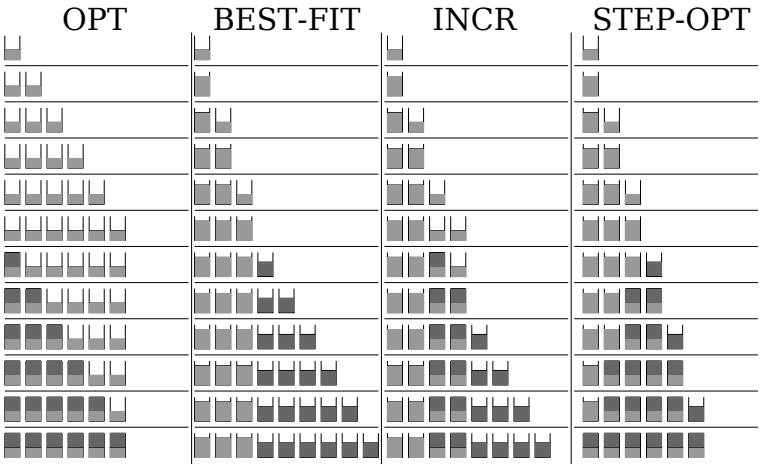
Analogous to the use of OPT above to denote the offline optimum solution to a problem, *INCR* will denote the optimal solution to a given incremental problem instance. The optimal solution is the one with the lowest possible incremental ratio. If there exist multiple 'optimal' solutions, then *INCR* will produce the one that has the lowest competitive ratio for the entire sequence. If these two conditions do not uniquely define a single solution, then *INCR* will select one at random. We will, after this point, refer to *INCR* as the incremental algorithm.

Like the competitive ratio (introduced in [33]), the incremental ratio is applied to both discrete problem instances and to entire problem spaces. When applied to a single problem, the incremental ratio will refer to the maximum incremental ratio achieved by the incremental algorithm (*INCR*). When the incremental ratio or the competitive ratio is applied over a problem space, we will refer to it as the **global incremental ratio** or **global competitive ratio** respectively. When the global competitive ratio of an online algorithm is α, we will sometimes refer to it as being an α-competitive algorithm. When that competitiveness is tight, the problem itself maybe be referred to as an $\alpha-$competitive problem. Similarly, if the global incremental ratio of a problem is α and its adversarial constraint has been calculated as β, it may be called an $\alpha-$incremental problem or a $\beta-$adversarial problem.

The global incremental ratio is also referred to as the incremental constraint of a problem. To isolate the contribution being made to the global competitive ratio of an online algorithm by the incremental constraint of the problem, we compute the global incremental ratio of the incremental version of the same problem. This global incremental ratio then represents the incremental constraint's contribution to the global competitive ratio. The adversarial constraint is then calculated by computing the difference between the two. A critique of this approach is included in the fourth chapter.

Intuitively, one might assume that when the offline version of a problem already produces solutions that satisfy the incremental constraint, the incremental ratio for that problem will be 1. As the counter-example of the k-server problem illustrates, however, this is not the case.

With an incremental algorithm and its means of evaluation defined, we now define the incremental solution to our bin-packing problem. Unlike the online algorithm Best-Fit, *INCR* is not blind to the future and is aware that if it pairs all 6 items of size $\frac{2}{5}$ off, the 6 items of size $\frac{3}{5}$ will each need their own bin. At the same time, unlike *OPT*, *INCR* needs to provide good solutions at every step; not just the last one.



In the diagram on the previous page, the first column corresponds to the behaviour of *OPT*, the second to *BEST-FIT* and the third to *INCR*. The final column, *STEP-OPT*, displays an optimal solution at that step (in some cases more than one arrangement could have achieved the minimum).

In this particular case, *INCR* achieves an incremental ratio of $\frac{4}{3}$. Had *INCR* paired off the third pair of small items, its behaviour would have matched that of the Best-Fit algorithm and thus would have achieved a greater incremental ratio of $\frac{3}{2}$. By contrast, if *INCR* had not paired off the second pair of small items, on the fourth step, it would have used 3 bins, versus the 2 that *OPT* used, for an incremental ratio of, again, $\frac{3}{2}$. Even worse would have been not pairing the very first pair of small items. That course of action would have led to an incremental ratio of 2 on the second step, as *INCR* would have used 2 bins to the 1 bin being used by *OPT*.

2.1.2 Incremental Algorithms and Online Algorithms

Incremental algorithms and online algorithms share many similarities. Both operate on a series of inputs, producing feasible solutions at each step. However, online algorithms are blind to the future, while incremental algorithms have perfect knowledge of the entire sequence of inputs. Furthermore, while online algorithms are only evaluated on their performance over the entire sequence, incremental algorithms are evaluated at every step.

This metric may appear too strict. Is it possible for an online algorithm to display a global competitive ratio that is strictly less than the global incremental ratio of the incremental algorithm? The answer to this question is no. This is, to the knowledge of this author, the first proof of this.

Theorem The global incremental ratio of the incremental algorithm for any given problem will not exceed the global competitive ratio for an online algorithm for the same problem.

Proof Given a problem instance and an online algorithm, compute the incremental ratio of the solution produced by the online algorithm. By definition, the incremental ratio of the online algorithm's solution must be greater than or equal to the incremental ratio of the solution produced by the incremental algorithm. The incremental ratio of an online algorithm's solution to a problem is equal to the maximum competitive ratio that that online algorithm exhibits on a prefix of that problem. Thus the global competitive ratio of an online algorithm is equal to its global incremental ratio. Thus, since on every problem instance, the incremental ratio of the online algorithm is greater than or equal to the incremental ratio of the incremental algorithm, it must follow that the global competitive ratio of an online algorithm is greater than or equal to the global incremental ratio of the incremental algorithm. ■

While this theorem holds for the global competitive ratio and the global incremental ratio, it does not necessarily hold for the competitive ratio and incremental ratio in every problem instance. Recalling the bin-packing example from the previous section (bins of size 1, 6 items of size $\frac{2}{5}$ followed by 6 items of size $\frac{3}{5}$, incremental ratio of $\frac{4}{3}$ and for the Best-Fit online algorithm a competitive

ratio of $\frac{3}{2}$), imagine an additional 12 items being added to the input sequence, all of size $\frac{1}{2}$. Now the competitive ratio of Best-Fit is $\frac{5}{4}$ (since it uses a total of 15 bins to the offline optimal solution of 12); but the incremental ratio of *INCR* is still $\frac{4}{3}$ (while it ends up using a total of 14 bins at the end of the sequence, its behaviour on the first 12 items will be unchanged). To elaborate on this simple counter-example, there exist cases where, while the initial part of the input sequence is very challenging for both online and incremental algorithms, the rest of the sequence is straight forward. For an online algorithm, these 'good' subsequences give it the chance to improve its competitive ratio. For incremental algorithms, since they are evaluated at each step, 'good' subsequences can never make up for 'bad' subsequences, unless the good subsequences come before the bad.

Incremental algorithms can be understood to be intermediate between the computational power of the offline optimal algorithm and online algorithms.

An alternate way of considering an incremental algorithm, is to model it as a online algorithm with access to an oracle. Assume we wish to model an incremental problem an input of length n. The corresponding oracle can be asked what additional input will arrive at an arbitrary step i. For steps $n+1$ onward the oracle will respond that no additional input will arrive on that step. For all other steps, the oracle will respond with the additional input. However, it may be lying. The online algorithm, augmented with an oracle, will actually be serving a prefix of the entire sequences. This prefix is chosen by an adversary to produce the worst possible competitive ratio for the online algorithm.

Online algorithms are commonly evaluated using the competitive ratio, introduced by Sleator and Tarjan in [33], in which the performance of the algorithm is compared against the offline optimal. This can be modeled as a game in which there is an adversary attempting to choose a sequence which will 'trip' the online algorithm up, producing a bad competitive ratio. This is referred to as the *adversarial constraint*. However, as was introduced in the work of Sharp [32], there are two factors contributing to the competitive ratio of an online problem: the *adversarial constraint* (as mentioned) and the *incremental constraint*. The later, as explained above, is the constraint that solutions generated by an online algorithm must build on the previous solutions.

Sharp's work goes on to explain that the competitive ratio does not differentiate between these two factors and thus cannot tell us which one is contributing more to the poor performance of an online algorithm. As the incremental ratio for an incremental algorithm is affected only by the incremental constraint; by comparing the gap between the global competitive ratio for the online version of a problem to the global incremental ratio of the incremental version of a problem, we can sort out the relative importance of the two factors in the competitive ratio.

When the gap between incremental ratio and the competitive ratio is small, it seems we can conclude that the adversarial constraint is contributing little and better online performance can only be found by improved incremental algorithms. More importantly, this would seem to indicate that improving input prediction is unlikely to lead to better performance. When the gap is large, it

seems we can conclude that the adversarial constraint is a significant factor, and improving input prediction is likely to lead to improvements in the performance of online algorithms. Unfortunately, as is explained in the final chapter of this thesis, these elegant conclusions are misleading.

2.1.3 Properties of Incremental Algorithms

Incremental algorithms are, in and of, themselves quite interesting, and display interesting increases in complexity in comparison with the classic algorithms on which they are based. While this thesis is primarily concerned with the insights into online algorithms and online problems that the study of incremental algorithms grants us, we wish to present some of these interesting complexity results.

Obviously, the incremental version of an NP-hard problem (such as network flow) contains, as a trivial special case, its non-incremental version and is therefore NP-Hard. Much less obviously, there exists polynomial time classic problems (such as bipartite matching [25]) for which the incremental version is NP-hard [32].

Curiously, the complexity of an incremental problem depends heavily on its exact formulation. Obviously, the formulation of the classic problem on which it is based matters; but the choice of varying parameter is also important.

To illustrate this, consider the problem of maximum flow. In its classic formulation, we are given a graph $G = (V, E)$, a source node s, a sink node t and a capacity function c defined on E, and we need to find the largest s-t flow f, such that the flow on any edge e does not exceed its capacity $c(e)$.

In its incremental (or as it is referred to in [23], hierarchical) formulation, we are given, in addition to the graph $G = (V, E)$ and our source and sink nodes (s and t respectively), a non-decreasing sequence of k capacity functions c_i (i.e. for all edges $e \in E$, for all i such that $2 \leq i \leq k$, we must have that the inequality $c_{i-1}(e) \leq c_i(e)$ holds). We then need to find k s-t flows f_i, such that the flow f_i on any edge e ($f_i(e)$) both does not exceed $c_i(e)$ and is at least $f_{i-1}(e)$ (i.e. for all $e \in E$, $f_{i-1}(e) \leq f_i(e) \leq c_i(e)$. Let $|f_i|$ denote the total amount of f_i flow sent from s to t.

While the classic problem of maximum flow is has a polynomial time algorithm [14], and its optimal solutions are always integral (in the sense that if the capacities of edges are integers the resulting maximum flow will be as well), neither is true for the incremental version. The optimal solutions to the incremental version of the problem (in which we attempt to maximize the expression $\min_i \frac{|f_i|}{d_i}$, where d_i is the value of the optimal flow for G with capacity function c_i) are potentially fractional.

Finding this optimal fractional solution has a polynomial time solution for the cases of directed graphs and bidirectional undirected graphs (i.e. graphs in which flow on an edge can be carried in either direction provided the total flow does not exceed the edge's capacity). In the case of unidirectional undirected graphs (i.e. graphs in which flow on an edge can be carried in one direction only)

the problem is NP-complete starting at $k \geq 3$ [23]. Meanwhile, if we restrict our possible intermediate solutions to being integral, the problem is NP-complete, for all three possible types of graphs, as soon as there is even one additional step (i.e. $k \geq 2$)[23].

Thus we have an example of a classic problem with a polynomial time solution that is NP-complete in its incremental formulation. Furthermore, while the different models for the base problem (fractional versus integral, directed versus undirected) were unimportant in its complexity as a classic problem; these distinctions were important for the complexity of the incremental version of this problem.

To illustrate the effect that changing the varying parameter in an incremental problem can have, we will be examining the minimum spanning tree problem. In its classic offline form, finding the minimum spanning tree of a graph $G = (V, E)$ is, depending on the algorithm chosen, a $O(E \log(V))$ or $O(E + V \log(V))$ time operation [14]. To develop an incremental version of this problem, there are several approaches; but we shall examine just two. The first is called the *node-incremental spanning tree* problem and the second is called the *cost-incremental spanning tree* problem.

In the node-incremental spanning tree problem, the input is a graph $G = (V, E)$ with a sequence of vertices $V_1 \subseteq V_2 \subseteq ...V_k \subseteq V$, or, as we'll refer to them for the rest of this discussion, nodes, arrive over time. Our goal is then to find an incremental sequence of trees $(T_1, T_2, ..., T_k)$ where $T_i \subseteq T_{i+1}$ and T_i spans the vertices of V_i (and no other vertices). Let $E_i \subseteq E$ be the set of edge with both endpoints in V_i and let c_i^{min}, c_i^{max} denote the cost of the minimum-weight and maximum-weight edges in E_i respectively. We then define $r_n^* = \max_i \frac{c_i^{max}}{c_i^{min}}$. For this problem there exists a simple incremental algorithm, with a global incremental ratio of r_n^* (which is the best possible global incremental ratio possible) that is very fortunately also an online algorithm [32].

In the cost-incremental spanning tree problem, the input is a graph $G = (V, E)$ with a non-decreasing sequence of cost functions c defined over E (or, to put it another way, the costs of the edges increase over time). Our goal is to find a single tree T that is good for every cost function. To put it more formally, we wish to minimize the expression $\max_i \frac{c_i(T)}{c_i(T_i^*)}$, where $c_i(T)$ is the cost of a tree T under the i^{th} cost function, and T_i^* is a minimum spanning tree under the i^{th} cost function. Similarly to c_i^{min}, c_i^{max} and r_n^* above: we shall define c_i^{min}, c_i^{max} to the cost of the minimum-weight edge and the maximum weight edge respectively under the i^{th} cost function in c and $r_c^* = \max_i \frac{c_i^{max}}{c_i^{min}}$. The global incremental ratio for the cost-incremental spanning tree is, similarly to above, r_c^* [32]. However, unlike node-incremental spanning tree, computing this bounded incremental ratio is not done online. In fact, the cost-incremental spanning tree problem is reducible to the NP-Hard problem Partition [32].

Thus we have an example of a problem where the choice of varying parameter used to generate its incremental version effected the complexity of the resulting incremental problem considerably.

2.2 Current Results

Previous work on the incremental constraint has focused on various combinatorial minimization and maximization problems, with the assumption that problems whose offline version seem to include an incremental constraint (such as paging, the ski rental problem and k-server) would have a global incremental ratio of 1 in their online form. But what does it mean for an offline problem to include the incremental constraint of a problem? At first inspection, if an offline version of a problem is order-independent and if its online version does not have a 1-competitive algorithm, then its incremental version likely has a non-trivial global incremental ratio. Notably, all of the problems outlined below share this set of properties.

Does it follow that problems, whose offline formulation do take the ordering of inputs into account, must have trivial global incremental ratios? The counter-example of k-server, unfortunately, disproves this simple heuristic.

Fortunately, there is another, more robust way, to characterize problems where the incremental constraint is a contributing factor to its global competitive ratio. For this we need to consider, given a sequence of inputs, what the optimal solutions, for each prefix of the sequence, look like. In cases where there does not exist a sequence of optimal solutions satisfying the incremental constraint, we must have a non-trivial global incremental ratio. In other words, when we have an input sequence ρ where there exist at least two steps, i and j, $i < j$, such that all optimal solutions for ρ_i are 'incompatible' with those of ρ_j, then we have non-trivial incremental constraint. We will call this the incremental property.

If we recall the bin-packing example of the previous section, such a pair of steps occur at steps 6 and 8. At step 6, the sole optimal solution has all six items of size $\frac{2}{5}$ paired off. At step 8, the sole optimal solution has two items of size $\frac{3}{5}$ paired with two items of size $\frac{2}{5}$. Since bin assignment is irrevocable, there is no possible solution to this problem that is optimal at both steps, and thus the bin-packing problem has a non-trivial global incremental ratio.

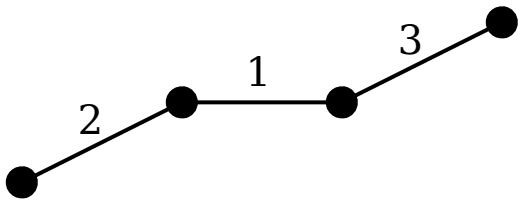
With this understanding of the incremental constraint in mind, here are the current results on the global incremental ratios of a variety of different problems. This choice of presentation is based heavily off that the work of Sharp [32], in that I've grouped the problems by the relative contribution of the incremental and adversarial constraints to their competitive ratios. The first two problems, bipartite matching and colouring are effected by both. The next three, Steiner tree, vertex cover and set cover are also effected by both; but only to a very limited degree by the incremental constraint. In the final two problems, the network flow and metric k-center problems, the competitive ratio is found to be entirely due to the incremental constraint.

2.2.1 Bipartite Matching

The bipartite matching problem consists of the following: given a graph (V, E), find the maximum set $M \subseteq E$ such that no two edges of the *matching* M share

a common endpoint. In the online and incremental versions of this problem, the edges of the graph arrive one at a time. As each edge arrives, the online (or incremental) algorithm must decide whether to include it in the matching or not. Karp et al.[27] give a tight $\frac{1}{2}$-competitive algorithm for the online problem. For the incremental problem, Sharp [32] gives a tight $\frac{2}{3}$-competitive algorithm.

To see that the bipartite matching problem satisfies the property that characterizes non-trivial incremental problems that we outlined above, we need only consider the following graph that was used, by Sharp, in proving the lower bound of the tight $\frac{2}{3}$-competitive algorithm mentioned above. The edges of this graph arrived in the order they are numbered in.

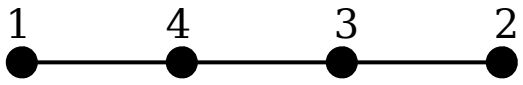


Obviously, at step 1, the maximum matching consists of edge 1, while at step 3, the maximum matching consists of edges 2 and 3.

2.2.2 Minimum Colouring

In the minimum colouring problem, given a graph $G = (V, E)$, one attempts to find an assignment (or colouring) $f : V \rightarrow C$ of colours to vertices such that adjacent vertices are assigned different colours and $|C|$ is minimized. It is NP-complete [18] and the best-known polynomial-time algorithm for approximate minimum colouring is $O(\frac{n(\log\log n)^2}{\log^3 n})$ [21]. In its online and incremental forms, the vertices arrive one at a time and must be coloured as soon as they arrive. In the online version of the problem, the colour of v_i is assigned only by looking at the subgraph of G induced by $\{v_1, v_2, ..., v_i\}$. It has been shown that any deterministic online colouring algorithm has competitive ratio of at least $\frac{2n}{\log^2 n}$ [22]. For the incremental case, the existence of a $O(\frac{n(\log logn)^2}{\log^3 n})$-competitive algorithm was shown by Sharp [32].

To see that the minimum colouring problem satisfies the property discussed above, consider the following small instance of the problem. The vertices of the graph arrived in the order in which they are labeled.



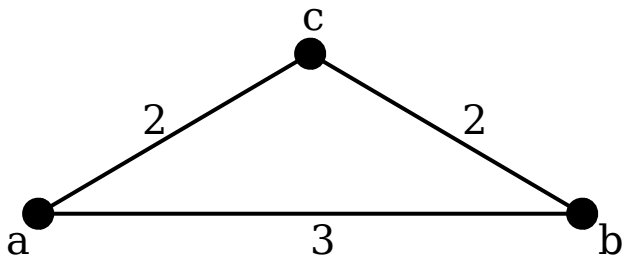
Obviously, at step 2, the smallest colouring uses only 1 colour, and colours both outside vertices the same colour. This is incompatible with the optimal solution at step 4, which uses two colours, and colours those two vertices differently.

2.2.3 Steiner Tree

The Steiner tree problem consist of finding, for a weighted graph $G = (V, E)$ and a terminal set $T \subseteq V$, a minimum cost tree that spans all of T. This problem is also NP-complete [18]. The best known polynomial time approximation algorithm produces a 1.55-approximation [31].

In online (and incremental) Steiner tree, the elements of T, the terminal vertices, arrive one at a time. As each new terminal vertex arrives, the solution tree is expanded to include it. In the online case, $\Omega(\frac{\log n}{\log \log n})$ is an almost tight lower bound on the competitive ratio [2]. In the incremental case, A 6.2-competitive ((4×1.55)-competitive) algorithm exists [32].

To see that the Steiner tree problem satisfies our incremental property, here is an illustrative instance of the problem. We assume that T is equal to V, and that the vertices arrive in order a, b, c.



Obviously, at step 2, the optimal spanning tree is to take the edge from a to b, while at step 3, the optimal spanning consists of the other two edges.

2.2.4 Vertex Cover

In the vertex cover problem, given a graph $G = (V, E)$, one finds the minimum set of vertices $V' \subseteq V$ such that at least one edge point of every edge is in V'. It is a NP-complete problem [18]. The best known polynomial time algorithm for vertex cover is a $(2 - \frac{\log \log n}{2 \log n})$-approximation [30]. It has also been shown in [24], that for sufficiently high vertex degree no approximation can beat a factor of $\frac{7}{6}$.

In online (and incremental) vertex cover, it is the edges of E that appear one at a time, with the set V being known from the beginning. At each step, the existing cover is extended to cover the new edge as necessary. There is a simple 2-competitive algorithm, for both the online and incremental cases, based on the 2-approximation algorithm, in which as each uncovered edge appears, both

of its end-points are added to the cover. Sharp proved that the incremental algorithm is bounded below by $\frac{7}{6}$ [32]. This lower bound is improved to $\frac{3}{2}$, in the fourth chapter of this thesis, by way of an example satisfying the incremental property.

2.2.5 Set Cover

The set cover problem consists, given a set X of n elements, of finding the minimum number of subsets, from m subsets $S \subseteq 2^X$, such that their union is equal to X. It is an NP-complete problem [18]. The best know polynomial time algorithm has an approximation factor of $\Theta(\log n)$[15].

For the online (and incremental) version of set cover, a subset $X' \subseteq X$ of elements is given to the algorithm one-by one, with the existing cover being extended with sets of S as each new element appears. While X and S are known to the online algorithm, X' is not. It has been shown that for all non-trivial values of m and n, that the competitive ratio of any deterministic algorithm for online set cover is at least $\Omega(\frac{\log n \log m}{\log\log m + \log\log n})$ [1]. The incremental set cover problem has been shown to have a $\Theta(\log n)$-competitive algorithm [32].

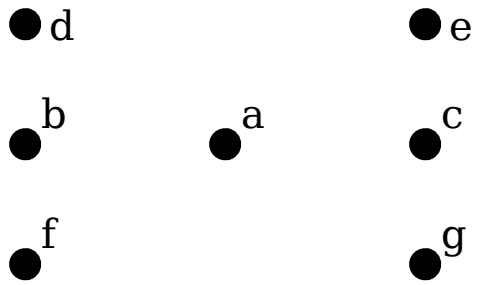
To see that the set cover problem satisfies the incremental property, we need only consider the following small instance of the problem. Let $X' = X = \{1, 2, 3, 4\}$, with the items arriving in their numbered order. Let our subsets be $\{1, 2\}$, $\{1, 3\}$ and $\{2, 4\}$. Obviously, at step 2, the optimal set cover consists of $\{1, 2\}$; but at step 4, the optimal cover will be $\{1, 3\}$ and $\{2, 4\}$.

2.2.6 Metric k-Center

The k-center problem begins with n points in a metric space. It then selects k of those points as *centers* so as to minimize the distance from any point to its closest center. It is known to be NP-complete [18]. The best known approximation algorithm is a 2-approximation [19] and it cannot be approximated to within 2-ϵ for any $\epsilon > 0$ [26].

Online center and Incremental center begin with same input as k-center (i.e. a set of n points in a metric space). Then, the number of selected centers increases. The 2-approximation mentioned above is also a 2-competitive algorithm for both cases. Since incremental center cannot have have a 2-ϵ-competitive algorithm for any $\epsilon > 0$, by the result mentioned above, the competitive ratios of the incremental and online versions of the two problems are identical, as was shown in the work of Sharp [32].

That this problem also has the incremental property, is easily illustrated by the following example. Here we assume (despite the scale of this diagram) that the distance between all adjacent points is equal (i.e. b is equidistant from a,d and f, a is equidistant from both b and c and c is equidistant from a,e and g).

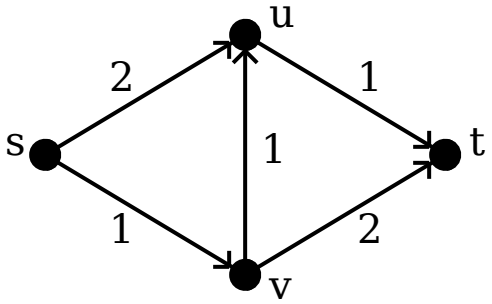


If $k = 1$, the optimal center to choose is a; but if $k = 2$, the optimal solution is to choose b and c.

2.2.7 Network Flow

The network flow problem has already been defined and complexity results relating to its incremental form discussed above. As mentioned above it can be solved in polynomial time. The model, for both online and incremental flow, is, again, the one from above where edge capacities increase over time, with a goal of deciding, as the capacities grow, how to send more flow from the source node s to the sink node t. There exists a strict $\frac{2}{n}$-competitive algorithm for the online flow problem [32]. Incremental flow also has a strictly $\frac{2}{n}$-competitive algorithm and this competitive ratio is optimal [32].

To see that the network flow problem satisfies that incremental property, consider the following small network flow problem instance, which is used in discussion of incremental flow by Sharp [32]. In this diagram, we assume all edges have the same capacity, with the numbers corresponding to which step this edge capacity 'arrives'. In other words, the edge capacity from s to u is 0 at step 1, and 1 at step 2, while the edge capacity from s to v is 1 at both steps.



At step 1, the maximum flow possible is 1, and it is achieved by taking all three available edges. At step 2, the maximum flow possible is 2, and it is achieved by taking all edges except the edge from v to u. Since an edge cannot decrease its used capacity between steps, these two flows are incompatible.

Chapter 3

Background Material on k-Server

3.1 Introduction

The k-server problem, is defined on a graph $G = (V, E)$. On this graph reside k servers, each located on a vertex in V. A request sequence ρ is a sequence of vertices in V. Servicing a request consists of moving a server to the requested vertex along the edges of the graph. The cost of such a move is equal to the sum of the edge weights traveled along. If a server is already on the requested vertex, it can serve the request without moving, for a cost of 0.

If the graph G is an undirected graph, the problem is then referred to as the symmetric k-server problem. If G is, by contrast, a directed graph, the resulting problem would be the asymmetric k-server problem. As is proven in the final chapter of this thesis, the global incremental ratio of the asymmetric k-server problem is unbounded and consequently the competitive ratio of the asymmetric k-server problem is also unbounded. The results on the symmetric k-server problem are far more intriguing; generally the symmetric k-server problem is referred to simply as the k-server problem. This convention will be followed in this thesis and all references to the k-server problem hereafter can be understood to refer to the symmetric k-server problem.

In the offline k-server problem, the goal is to minimize sum of the server moves needed to serve a given request sequence. In the online k-server problem, requests arrive one by one and an online k-server algorithm must decide which server to use to serve each request without knowledge of future requests. Its goal is still to minimize the sum of the costs of the moves used. The incremental k-server problem is similar, in that requests arrive one at a time; however the entire request sequence is known in advance. The performance of a solution to the incremental k-server problem is evaluated after each request is served. This is done by comparing the cost that has accrued while serving the prefix of the request sequence ρ ending at that step with the cost that the offline optimal

algorithm would have accrued serving just that prefix.

An alternate definition is to place the k-server problem on a metric space. The initial position of the servers and the requests in the request sequence then correspond to points on that metric space. The cost of server moves is the computed with a distance function defined on the metric space. This distance function must be symmetric and must satisfy the triangle inequality.

The k-server problem was introduced in [29]. It was in that paper that the k-server conjecture was first proposed. This conjecture states that there exists a k-competitive algorithm for the k-server problem. What follows is a summary of the progress made in the intervening 24 years towards proving this conjecture. Instead of organizing these results in chronological order, they will be grouped by type instead. We have divided them into two classes. The first are results relating to the general problem, while the second are results relating to special cases. The special cases can themselves be broken into two sub-classes: restrictions on the graph G, and finite values of k.

Attacking the problem by way of special cases was established in the initial paper on the k-server conjecture, where it was shown to hold in two special cases: when k is equal to 2 and when k servers are serving only $k+1$ distinct points. That the conjecture holds when k is equal to 1 is obvious. That it holds when the number of servers is equal to or greater than the number of distinct points is also obvious.

A further special case, that of the paging problem (which is equivalent to the k-server problem on a metric space where all points are equidistant), has already been shown to have a k-competitive online algorithm in [33].

3.2 General k-Server Results

3.2.1 The Lower Bound

In the paper that introduced the k-server problem [29], a lower bound of k for the competitive ratio for the k-server was proven. It in fact proves a stronger result, showing that for any k-server online algorithm A, there is an h-server algorithm B (with $h \leq k$) and a set of requests ρ_i such that $C_A(\rho_i) > \frac{k}{k-h+1} C_B(\rho_i)$. Unsurprisingly, given that the paging problem is a special case of the k-server problem, the proof is heavily based on the proof of the lower bound of the competitiveness of online paging algorithms from [33]. The proof of the simpler theorem is included here, as it is both elegant and illustrative.

Theorem (Manasse et. al. [29]) The competitive ratio of the k-server problem is bounded below by k.

Proof Given a graph G, with at least $k+1$ distinct vertices, and an initial configuration for k servers, an arbitrary sub-graph of $k+1$ vertices is chosen to contain all k vertices covered by the initial configuration. Obviously, there are $k+1$ possible configurations of k severs on $k+1$ vertices. Let A be any given online algorithm for the k server problem. We then compare A against k offline algorithms $B_1, ..., B_k$. These $k+1$ distinct algorithms maintain the

following invariant: at the beginning of any given request in the input sequence, the configurations of the algorithm A and the k algorithms $B_1, .., B_k$ are all different (and thus cover all $k+1$ possible configurations of k servers on the subspace).

Initially, the servers of the offline algorithms B_i are moved from the original configuration to all other configurations (this fixed cost can be ignored, as it depends only on the initial configuration). The first request r_1 is then chosen to be the one vertex that A does not cover; but that is, of course covered by all k of the offline algorithms. To serve request r_1, A must move one its servers from some vertex. Assume this vertex is a. Once it has done so, its new configuration matches that of some B_i. This algorithm, B_i, then moves its server from r_1 to a. Thus while the cost of A has been $d(a, r_1)$, the combined cost of all k offline algorithms has been also been $d(a, r_1)$. This process then repeats.

Prior to the start of each step, the invariant concerning the configurations of A and B_1 through B_k still holds, and the cost of A is equal to the sum of the costs of all k offline algorithms. Thus, we can conclude that A must be at least k times the cost of one of the offline algorithms. Label this algorithm B_i. Since B_i can have a total cost of no more than that of OPT, it follows that the competitive ratio of A is at least k. ■

This lower bound on the competitive ratio of any online algorithm for the k-server problem is applicable to any graph with $k+1$ vertices. Thus, there exist no non-trivial special cases amongst graphs for which the competitive ratio is less than k.

3.2.2 Upper Bounds

When this lower bound had been established, the question of whether there was even a finite upper bound was still open. The first result to demonstrate the existence of an online algorithm with a finite competitive ratio for the k-server problem in all metric spaces was found by Rabani, Fiat and Ravid in [17]. Their algorithm was the Expand-Contract algorithm and it is relatively complex. At a value of $\Theta(k!^3)$, the competitive ratio for the Expand-Contract algorithm was, unfortunately exponential in k.

It was shown by Grove (in [20]), that the Harmonic algorithm has a competitive ratio of $O(k2^k)$, which, by the derandomization technique of [6], generates a deterministic competitive ratio of $O(k^2 4^k)$. This result of Grove [20] was then improved in [3], by himself and Bartel, to $2^k \log k$, for a deterministic competitive ratio of $4^k \log^2 k$. Results concerning the Harmonic algorithm on special cases will be discussed in the following section.

The Harmonic algorithm is a memoryless randomized algorithm. When an uncovered vertex is encountered in the request sequence, the algorithm chooses a server at random to move to the requested vertex. The probability that a given vertex is chosen is inversely proportional to the distance it will need to travel.

The competitiveness of memoryless online algorithms, as a distinct class of algorithm, is of interest as they take fewer computational resources to run.

Obviously, any memoryless deterministic algorithm for k-server will have unbounded competitiveness. As is mentioned below in the section on results for special cases, there exists graphs for which the Harmonic algorithm is $\frac{k(k+1)}{2}$. Currently, the conjecture is that the Harmonic algorithm is $\frac{k(k+1)}{2}$ competitive on all graphs. This is called the Harmonic conjecture.

Papadimitriou and Koutsoupias showed that the Work-Function algorithm had competitive ratio of $2k-1$ [28]. To better understand the algorithm, it is useful to consider how one might solve the offline k-server problem. An obvious approach would be to use dynamic programming. We could define a function w, that takes in two configurations and a sequence of requests. If, for instance, the initial configuration of the graph was C_0, then $w(C_0; r_1, r_2, .., r_n, X)$ would be the value of the optimal solution that begins in configuration C_0, serves requests r_1 through r_n and ends in configuration X. Formally:

$$w(C_0; r_1, .., r_n; X) = \min\{\sum_{i=0}^{n} d(C_i, C_{i+1}) : r_i \in C_i, C_{n+1} = X\} \tag{3.1}$$

This will be denoted as $w_{c_0;r_1,..,r_n}$ or w_n when C_0 and the request sequence $\rho = r_1, ..., r_n$ are understood. Since $w_i(X) = \min\{w_{i-1}(Z) + d(Z, X), r_i \in Z\}$, this function can be computed dynamically, using $w_0(X) = d(C_0, X)$ as bootstrapping values.

The Greedy Algorithm, that simply moves the closest server to serve each request, is the most natural online algorithm. To put it another way, it chooses the configuration at step i that minimizes the expression $d(C_{i-1}, C_i)$. Unfortunately, it has an unbounded competitive ratio. As a proof sketch, consider a small graph of three vertices, with two vertices A and B quite close and a third vertex C only slightly farther away from both, with one server on B and one on C and a request sequence consisting entirely of As and Bs. The Greedy algorithm would keep moving the server from A to B and back again running up an arbitrarily large cost, while the offline optimal would have simply moved the server from C over to A for finite cost. This proof is formally presented in the appendix.

Another, equally extreme and even more foolish, algorithm is the Retrospective Algorithm. At each step it chooses to move its servers into the configuration that minimizes the current work function (i.e. on step i, it moves its servers into the configuration X that minimizes $w_i(X)$). By inspection, like the Greedy Algorithm, the Retrospective Algorithm has an unbounded competitive ratio. A proof of this is presented in the appendix.

The Work Function Algorithm attempts to balance between these two extremes. At step i, it chooses configuration C_i that minimizes the expression $w_i(C_i) + d(C_{i-1}, C_i)$.

The proof that this algorithm is $2k-1$-competitive, uses a potential function argument like that used in the proof that the double coverage algorithm on a line is k-competitive, which is presented in the section on k-server results on restricted graphs. Here the potential function is composed of expressions using

the work function. The proof then uses various properties of the work function that are induced by its structure.

It is believed that the Work-Function algorithm is in fact k-competitive [28]; but a tight upper bound on its competitiveness remains an open problem.

3.3 Results on Special Cases of k-Server

3.3.1 Results for Finite k

Results for finite k first appear in the paper introducing the k-server problem [29]. In that paper, it was shown the RES algorithm was 2-competitive for the 2-server problem. RES, as it names alludes to, is a residue based algorithm, much like the Balance algorithm, detailed below, from the same paper.

The RES algorithm is intended strictly for the 2-server case. First, it always keeps the two servers on different vertices and, since one of those vertices must represent the last request served, we must have that any offline solution must also cover one of the two vertices. Let us label the last requested vertex as A, and let the other be labeled B. This leads us to the identities of the maintained residues. On a graph of N vertices, the offline solution must have the 'B' server on one of the other N-1 vertices. RES maintains residues for each of these N-1 cases. For a vertex i, the residue R_i is found by computing the cost of an offline optimal algorithm that ends with servers on vertex A and i (for the very first step, we assume $R_i = 2d(B, i)$). If the next requested vertex is k, then the server on vertex A is chosen to serve it if and only if $\min_i R_i + 2d(i, k) \geq 2d(A, i) + d(A, B)$. Otherwise we move the server on B to serve the request at vertex k.

The proof of the competitiveness of RES is in showing that the inequality $R_i + R_j \geq a + 2 \cdot \max(\mathrm{d(A,B)} + \mathrm{d(i,j)}, \mathrm{d(A,i)} + \mathrm{d(B,j)}, \mathrm{d(A,j)} + \mathrm{d(B,i)})$ holds for all possible i and j, where a is a constant dependant on the initial conditions of the graph. This inequality allows us to conclude that the competitive ratio is bounded below by 2.

Another step for finite k related results came in [12], in which it was shown that the Equipoise algorithm was 11-competitive for 3 servers. The details of the algorithm itself relate both to calculations done on graphs generate by the 3 servers configuration on the graph and the use of virtual server moves to generate 'credit' assigned to the corresponding 'actual' unmoved server. The proof of the competitiveness is made not against the actual offline optimal but against what is referred to in the paper as a turbo-adversary whose performance is occasionally better than optimal. The result for 3 servers however, was surpassed by the proof that the Work Function Algorithm, discussed above, is $2k - 1$-competitive, giving a 5-competitive algorithm for the 3-server problem on arbitrary metric spaces.

An additional result for 3-server was shown in [4], in which it was shown that the Work Function algorithm (discussed above in the section on general k-server results) is 3-competitive for the 3-server problem in the Manhattan plane. As

a corollary, the paper also shows that the same algorithm is $3\sqrt{2}$-competitive on the Euclidean plane (since the Euclidean metric can be approximated by the Manhattan plane).

For higher values k, there has been no progress except in the form of the general results for unbounded values of k.

3.3.2 Results for restricted graphs

As mentioned above, in the first paper on the k-server problem, the k-server conjecture was shown to hold in the special of case of a graph consisting of $k+1$ vertices. The specific online algorithm that was shown to be k-competitive in that case was the Balance algorithm. This algorithm tracks the total distance travelled by each server. As each request arrives the Balance algorithm chooses to move the server that will accrue the smallest cumulative cost after moving. Unfortunately, this straightforward algorithm was found not to be k-competitive on general graphs. The proof of its competitiveness on graphs of $k+1$ vertices relied on invariants involving the residues computed using the cost of the OPT and the cost of the Balance algorithm. This method was essentially extended to develop the RES algorithm detailed in the previous section.

For the case of graphs consisting of $k+2$ vertices, it was shown that the Work Function algorithm satisfies the k-server conjecture. This result is included in the paper that introduced the Work Function algorithm [28]. The proof is similar to that of the proof that same algorithm is $2k-1$-competitive on general graphs; but uses a different potential function.

While these results on graphs of size $k+1$ and $k+2$ are encouraging, the first result involving metric spaces of unbounded size was found by Chrobak et. al. [10]. In the cited paper they showed that the double coverage algorithm satisfied the k-server conjecture for graphs in the form of a line. They also provided some interesting results concerning the asymmetric k-server problem: proving that the competitive ratio was unbounded for the general k-server problem and presenting a competitive algorithm for the special asymmetric case of weighed caching. Their results concerning the double coverage algorithm were then extended to trees in [11].

We will now present the double coverage algorithm and a sketch of the proof that it is k-competitive on a line as it is an excellent illustration of the potential method used to prove many of the results concerning the k-server problem; both in the general case and in the case of restricted graphs. These results from the paper by Chrobak et. all. [10].

Picture a line containing two or more servers. When a request is made, if it is not already covered by a server, it must fall between two servers or to the extreme right or left of the line. In the latter case, the double coverage algorithm moves the single adjacent server to serve the request. When a request falls between two servers, both adjacent servers are moved an equal distance towards the server until the request is served.

For example, if a requested point C falls between two servers on points A and B (with A assumed to be on the left of C), with C at a distance of 2 from A

and a distance of 3 from B, then the server on A is used to serve the request for C. However, the server on B is still moved 2 units to the left from its starting position.

The double coverage algorithm is the first algorithm here discussed that is not a 'lazy' algorithm. Lazy k-server algorithms do not move their servers unless actively serving a request. All non-lazy algorithms can be converted into lazy algorithms: a conversion that can only improve the competitiveness of an algorithm. In this case, the conversion is easy and all that is required is to keep around a potential associated with any servers that would have been moved unnecessarily.

To extend our example, assume it was carried out with a lazy version of the double coverage algorithm. The server on B would not move; but would be noted to have a 'free' move of 2 units to the left available to it. Thus, if the next request D were to fall between the server on C and the server on B, at a distance of one unit from C and two from B, since B would have a 'free' move to the left of two units, we would, under the lazy double coverage algorithm, choose to move the server on B to serve the request for D, instead of the server on C, despite the latter server being closer to the request.

While the lazy version of the double coverage algorithm is superior, we shall assume that we are working with the non-lazy version as is greatly simplifies the proofs that follow. The other advantage of the non-lazy version of the double coverage algorithm is that it is memoryless algorithm, and as was discussed earlier in the section on general k-server results, memoryless algorithms are desirable for their speed of execution.

Theorem(Chrobak et. al. [10])

The double coverage algorithm for the k-server problem is k-competitive when applied to a graph in the form of a line.

Proof Sketch

The proof here relies on what is referred to as the *potential function method*. We define a potential function $\Phi(C_t, C_t^{'})$ where C_t is the configuration of the servers on the line after t steps under the double coverage algorithm, while $C_t^{'}$ is the configuration of the servers after t steps under the offline optimal algorithm. If OPT_t represents the move made by the offline optimal to serve the tth request and A_t the same for the online algorithm, we then use our potential function to show that $\text{cost}(A_t) - \rho \cdot \text{cost}(OPT_t) \leq \Phi(C_{t-1}, C_{t-1}^{'})$. Once this is established, by summing all m terms in a given sequence, we can conclude that $\sum_{t=1}^{m} \text{cost}(\text{A}_t)$ - $\rho \sum_{t=1}^{m} \text{cost}(OPT_t) \leq \Phi(C_0, C_0^{'})$, which implies that A has a competitive ratio of ρ.

In the case of the double coverage algorithm for the k-server problem on a line, our potential function is defined as follows. We first label the k online servers $a_1, .., a_k$ and the k offline servers as $b_1, .., b_k$. Without loss of generality, this labeling applies left to right. Since at no point, in either the offline or the online solution will a server move past another server, the ordering of the servers

will be constant. The potential function is then $\Phi = k \sum_{i=1}^{k} |a_i - b_i| + \sum_{i<j} (a_j - a_i)$. ■

The proof was extended to cover trees as well, in [11], essentially by changing 'adjacent' to the term 'unobstructed', which is defined to mean that there are no other servers on the unique path from a server a_i to a request r_i.

When the double-coverage approach was extended on the circle, it was, unfortunately, not found to be k-competitive. Here, our circle is continuous and has an infinite set of points. Discrete circles fall into the category of resistive metric spaces, which are discussed further down. When the double coverage algorithm is extended to the circle, it is $O(k^3)$-competitive [16]. In the cited paper, the same proof method and even the same potential function were used; but due the properties of a circle, the resulting competitiveness was significantly higher.

A very different approach was taken in the case of resistive graphs. In the work of Coppersmith et. al. [13], a randomized algorithm with competitiveness of k for the k-server problem on resistive graphs was presented. The result presented was even slightly more general than that, applying if every $(k+1)$-node subgraph of a graph was resistive. Before we can discuss resistive graphs or even define them, we need to present an alternate way of looking at the graphs that the k-server problem operates on.

A graph can be thought of as a matrix of costs. Such a cost matrix C is obviously square, with one row (and one corresponding column) for each vertex in a graph, with the entry c_{ij} corresponding to the cost to travel for vertex v_i to vertex v_j (c_{ii} is 0 for all i). We can then consider a random walk on a graph G, with the probabilities of each step governed by a probability matrix P, where p_{ij} is the probability that a walk moves from v_i to v_j. We then define e_{ij} to the expected cost of a random walk starting at v_i and terminating at v_j (with e_{ii} corresponding to the expected cost of a round trip from v_i).

We then say a random walk has *stretch* c if there exists a constant a such that for any sequence of vertices $v_{i_0}, v_{i_1}, ..., v_{i_l}$, $\sum_{j=1}^{l} e_{i_{j-1}i_j} \leq c \cdot \sum_{j=1}^{l} c_{i_{j-1}i_j} + a$. To find the optimal probability matrix that generates a random walk with optimal stretch, we turn our attention to electric network theory.

Given a graph, we can model it as as network of resistors, with a *conductance* σ_{ij} between v_i and v_j (i.e. v_i and v_j are connected by a resistor with *branch resistance* $\frac{1}{\sigma_{ij}}$). Let the effective resistance between v_i and v_j be labeled R_{ij} (i.e. $\frac{1}{R_{ij}}$ is the current that would flow from v_i to v_j if one volt were applied).

From this we can define the probability matrix for *resistive random walk* by letting $p_{ij} = \frac{\sigma_{ij}}{\sum_k \sigma_{ik}}$. In the above mentioned paper [13], it was shown that this random walk has a stretch of $n-1$ in the graph with the cost matrix defined by $c_{ij} = R_{ij}$. In the same paper, a stretch of $n-1$ was proven optimal. Thus to find, for a graph G, its random walk with optimal stretch, one needs to simply compute the resistive inverse (σ_{ij}) of the cost matrix C. Unfortunately, not all cost matrices have resistive inverses. Those that do we shall label as *resistive*

graphs.

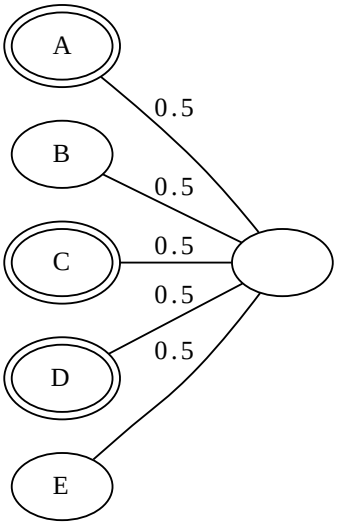
To simplify, if a graph could have a physical counterpart in the form of a network of resistors, with the effective resistance between each pair of nodes corresponding to cost of travelling between those vertices in the graph, then there exists a k-competitive randomized algorithm for the k-server problem on that graph.

The randomized algorithm, that is presented as k-competitive for resistive graphs in [13], is essentially to apply the *resistive random walk* algorithm at each step to the subgraph for $k + 1$ vertices consisting of the k vertices covered by the k servers and the additional requested vertex. The proof, like the previous three cases discussed, uses the potential function method.

Chapter 4

The Incremental Constraint of k-Server

As was mentioned above, paging can be modeled as a special case of the k-server problem. To see this, consider the diagram below.



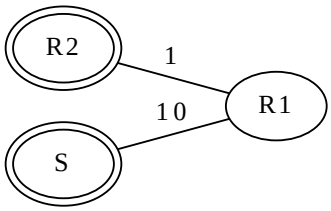
Each of the labeled nodes corresponds to possible page that might be requested. Each node with a second circle corresponds to a node with server on it. Thus this diagram corresponds to an instance of the paging problem with three spaces in memory with pages A, C, D in memory and pages B, E not in memory. To remove a page from memory and replace it with another (for ex-

ample if one wanted to eject page A and replace it with page B), the server is moved from the ejected page (in our example A) to the replacement page (B). This has identical cost regardless of which two pages of memory are being exchanged.

It thus appears that paging and k-server are very closely related. However, while paging has an incremental constraint of 1 (a proof of this is given below in the section on the criticism of the incremental constraint), k-server most definitely does not (as shown below). The incremental constraint is, to the best of the knowledge of this author, the first metric to separate the k-server problem from the paging problem. While the paging problem has an incremental constraint of 1, as in is shown in this chapter, the incremental constraint of the k-server problem is at least 2.

4.1 A Lower Bound on the Incremental Constraint of the Symmetric k-Server Problem

Consider the following graph:



Assume that the request sequence associated with this graph is R_1, R_2 repeated 100 times. Obviously, the offline optimal solution is to move the server from S to R_1 for the first request and at that point we can begin serving all remaining requests for free. This gives us a total cost of 10 for the entire sequence. However, this request serving sequence, on the very first request, has an incremental ratio of $10 = \frac{10}{1}$, as the first request could have been served for a cost of 1 by moving the server from R_2 to R_1. Similarly, the second request can be served for an additional cost of 1 by moving the server from R_1 to R_2 and so forth.

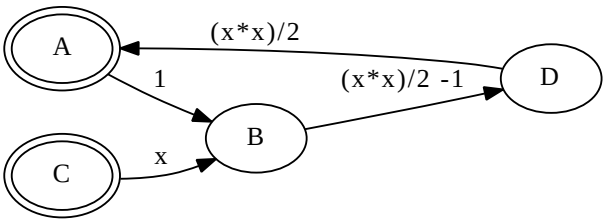
If we wait until the ith request (with $i \leq 10$) to move the server from S, we have an incremental ratio of $\frac{10+2\lfloor \frac{i}{2} \rfloor}{i}$. If we wait until after the 10th request to move the second server, the ratio on that step will we $\frac{10+2\lfloor \frac{i}{2} \rfloor}{10}$. Obviously, we can minimize the ratio by maximizing i in the first expression, which translates to moving the second server (the server that is on node S in the initial configuration) to serve the 10th request, giving us a incremen-

tal ratio of $\frac{10+2\lfloor\frac{10}{2}\rfloor}{10} = \frac{10+2(5)}{10} = 2$. One step previously, the incremental ratio would have also been $\frac{10+2\lfloor\frac{9}{2}\rfloor}{9} = \frac{10+2(4)}{9} = 2$, while just one more step previously: $\frac{10+2\lfloor\frac{8}{2}\rfloor}{8} = \frac{10+2(4)}{8} > 2$. On the eleventh step, again the ratio is unchanged at $\frac{10+2\lfloor\frac{11}{2}\rfloor}{10} = \frac{10+2(5)}{10} = 2$ and one step after that, again increases: $\frac{10+2\lfloor\frac{12}{2}\rfloor}{10} = \frac{10+2(6)}{10}$. Thus we have demonstrated a lower bound of 2 on the incremental constraint of the k-server problem.

4.2 The Incremental Constraint of the Asymmetric k-Server Problem

As was shown in [10], the competitive ratio of the asymmetric k-server problem is unbounded. The following result shows that the incremental constraint of the asymmetric k-server problem is also unbounded.

On a directed graph, the incremental constraint of the k-server problem has no upper bound. Consider the following graph:

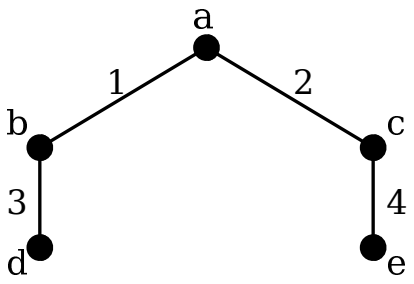


Assume we begin with servers on nodes A and C and have a request sequence consisting of just two requests: B, A. There are only two request serving sequences. In the first, we move the server from A to B to serve the request for B and then move from B to D back to A to serve the request for A. In the second, we move the server from C to B to serve the request for B and then, since there is already a server on node A, serve the request for A for free. This second sequence has a cost of x and is the offline optimal way to serve the entire request sequence. However, it has a incremental ratio of x, as the first request, B, can be served for a cost of 1. The first sequence also has an incremental ratio of x, as it has a total cost of $1 + \frac{x^2}{2} - 1 + \frac{x^2}{2} = x^2$, in comparison with the total cost of x for the second sequence.

Since x is an arbitrary value, we can conclude that the incremental constraint for directed graphs is unbounded.

4.3 A Lower Bound for Vertex Cover

Consider the following graph.



In the online and incremental versions of the vertex cover problem, as an edge arrives, the vertex cover must be extended to cover the new edge. Here the edges arrive in the order in which they are numbered. By inspection, at step 2, the optimal solution is a cover consisting of the vertex a; but by step 4, the optimal solution consists of b and c.

At step 1, if b were chosen instead of a, at step 2, a second vertex, either a or c would have to be chosen. Thus, the incremental optimal solution chooses a on the first step so as to avoid having an incremental ratio of 2 on the second step.

Once a has been chosen, no additional vertices are needed until the third step. On the third step, one of b or d must be chosen. This third step marks the point where the optimal solution must begin using two vertices, since the third and second edges share no vertices in common. One of these vertices can still be a.

On the fourth, and final, step, there exists an optimal solution with only two vertices, b and c. Regardless of which vertex was chosen at step three (since neither b nor d represent ends points for the fourth edge), the incremental optimal solution must choose a third vertex. Thus on this on this problem instance, the incremental optimal solution has an incremental ratio of $\frac{3}{2}$.

Since no series of chosen vertices can achieve an incremental ratio of less than $\frac{3}{2}$, we can conclude that the global incremental ratio of this problem is bounded below by $\frac{3}{2}$, which represents a significant improvement on the previous lower bound of $\frac{7}{6}$ [32].

4.4 Asymptotic Incremental Constraint

Through out this thesis, and in all previous work on the incremental constraint, the incremental ratio has been presented as analogous to the competitive ratio. However, as defined, the incremental ratio is, more precisely, analogous to the strict competitive ratio.

There are two ways in which an asymptotic component might be included

in the incremental ratio. The first is to allow a constant additive term, as is the case in the competitive ratio. The second is to 'ignore' some constant number points in the sequence when computing the incremental ratio. The first is more intuitive; but given that incremental ratio is calculated by looking at the ratio on every step, would the second produce a lower, more accurate, incremental ratio?

For the second to produce a lower ratio, first there must exist single steps which increase the cost of a solution by an arbitrarily large value. If the increase to the cost on a single step is bounded above by a constant, then in a constant number of steps, the cost can only increase by a constant additive term.

To put it more formally, given a request sequence ρ, let $A(\rho, i)$ be some solution for the entire sequence ρ at step i and let ρ_i denote the length i prefix of ρ. In the case where there exists a constant upper bound, c, on the additional cost of a step, if the step on which the incremental ratio of A is achieved, i.e. the step j for which $\frac{\text{cost(A}(\rho,\text{j}))}{\text{cost(OPT}(\rho_\text{j}))} = \max_i \frac{\text{cost(A}(\rho,\text{i}))}{\text{cost(OPT}(\rho_\text{i}))} = \alpha$ holds, is excluded, then the ratio at step $j-1$ is bounded below by $\frac{\text{cost(A}(\rho,\text{j}))-\text{c}}{\text{cost(OPT}(\rho_\text{j}))}$, since the difference in cost between two steps is at most c and the sequence $\text{cost}(\text{OPT}(\rho_\text{i}))$ is a monotonically increasing. When more than one step in a row is excluded, the same method can be expanded by changing c in the previous expression to $d \cdot c$, where d is the number of steps excluded.

This condition alone eliminates many of the problems discussed in this thesis, including bin-packing, vertex cover, set cover, minimum colouring. The above discussion assumed the existence of an underlying combinatorial minimization problem. In the case of maximization problems, similar definitions and arguments results in the same conclusions, allowing us to also eliminate the problem of bipartite matching.

But what of problems with unbounded single steps? Could the exclusion of a constant number of steps from the calculation of the incremental ratio result in a lower global incremental ratio than allowing a constant additive term? We would next need that the number of steps on which the previous global incremental ratio is achieved is always constant for a given problem instance. If we can describe a request sequence of arbitrary length in which the global incremental ratio is achieved an unbounded number of times, then the exclusion of a constant number of steps obviously does not result in an improved global incremental ratio.

The next constraint takes more work to be shown to apply to a given problem; but, it can be shown to apply to the rest of the problems discussed in this thesis. For problems like Steiner tree, network flow and k-server, simply repeating larger scale worst case solutions, will, with the exclusion of a finite number of steps, will result in the same global incremental ratios.

To illustrate, in the case of k-server, we define a request sequence that begins in the same manner as that in the proof of the lower bound, we then add another copy of the same graph (potentially connected to the original three vertex graph; but at an extremely large distance), but with the long edge increased from 10 to some much larger number (for any given ϵ, we can find an edge length n, such

that $\frac{n+20}{n} < 1 + \epsilon$), and the number of repetition of two vertices corresponding to R_1 and R_2, similarly increased so as to force the incremental solution to eventually take this very long edge. This extension is then added an arbitrary number of times, so that the incremental ratio is bounded below by $2 - \epsilon$ on an arbitrary number of steps.

For k-center, instead of a repeating sequence of ever-larger problems, we instead describe a series of repeating ever-smaller problems. Consider a simple pair of points, and take the point midway between those two. Obviously, on the first step, when $k = 1$, the optimal solution is to take that middle point. If we then extend the diagram by adding in two more points, each midway between the middle point and the two original points, this doesn't change the identity of the optimal solution on the first step; but instead defines the optimal solution on the second step. Similarly, if we extended the diagram again by adding four new points, at the four midway points between the five points we have already described, the identity of the optimal solutions on the first and second steps are unchanged; but these four points define the optimal solution at the fourth step. We can repeat this process an indefinite number of times. At each step whose value is a power of 2, the incremental algorithm is forced into solutions that are increasingly close to twice the optimal.

This then eliminates all of the problems for which the incremental constraint has been studied. It is still possible that there is some problem for which allowing a constant number of steps to be excluded would generate a lower incremental constraint than allowing a constant additive term; but given the combination of required constraints identified so far (unbounded single steps, bounded number of 'bad' steps on all possible instances), this seems unlikely.

With the exclusion of a constant number steps from the computation of the incremental ratio excluded as a means to add an asymptotic factor to the global incremental ratio, we turn our attention back to the use of an additive constant. At this point, we will provided a formal definition.

Definition We say that a problem has an **asymptotic global incremental ratio** of α if there exists a constant c, such that for all request sequences ρ, and all possible prefix lengths i of ρ, $\alpha \cdot OPT(\rho_i) + c \geq INCR(\rho, i)$ holds.

As it turns out, for almost all of the problems discussed in this thesis, the asymptotic global incremental ratio is equal to the global incremental ratio. For almost all problems here discussed, including k-sever, vertex cover, set cover, minimum colouring, Steiner tree, bipartite matching and network flow, separate worst case instances of the problem can be repeated in the same request sequence, as is used in [32] in the proof of the lower bound of the global incremental ratio of bipartite matching. For graph problems, these separate instances can take the form of disconnected graph components. For the set cover problem, they can take the form of disjoint sets. Since a constant additive term can only ameliorate the incremental ratio in a finite number of these worst case instances, in all of these cases, the asymptotic global incremental ratio will match the global incremental ratio.

The case of k-center is different, as separate instances of the problem cannot be described in the same request sequence. However, by simply enlarging a

worst case instance of the problem, any given constant factor can be made to reduce the global incremental ratio by ϵ.

Does this mean that a constant additive term can never decrease the global incremental ratio for a problem? Fortunately, the one remaining problem we have to examine does seem to exhibit a difference between the its global incremental ratio and its asymptotic global incremental ratio. The problem in question is that of bin-packing.

The global incremental ratio of the bin-packing problem is bounded below by $\frac{3}{2}$. While the example explored in chapter 2 yielded an incremental ratio of $\frac{4}{3}$, this can be improved upon by simply looking at a smaller example. If, instead of 6 items of sizes $\frac{2}{5}$ and six items of $\frac{3}{5}$, we consider four items of size $\frac{2}{5}$ and four items of size $\frac{3}{5}$, then the incremental ratio increases to $\frac{3}{2}$. Such a worst case example is easily defeated with a constant factor of as little of 2 (in particular, allowing a constant factor of 2 or more, gives this example an asymptotic incremental ratio of 1).

As for the slightly larger example used in chapter 2, since extending the number of requests in the same pattern generates the same incremental ratio, we can bound the asymptotic global incremental ratio of the bin-packing problem below by $\frac{4}{3}$.

To see this, consider the midway point in a request sequence of $6n$ items of size $\frac{2}{5}$, followed by $6n$ items of size $\frac{3}{5}$. The *STEP-OPT* solution and the *BEST-FIT* solution both used $3n$ bins, each consisting of two items of size $\frac{2}{5}$, while the *OPT* solution uses $6n$ bins, each with a single item of size $\frac{2}{5}$. *INCR* has chosen to not pair some proportion of the $3n$ pairs of items of size $\frac{2}{5}$. Call this proportion α, then at this mid-step, the ratio between the step-optimal solution and the incremental solution will be $1 + \alpha$.

At the final step, the optimal solution will consist of $6n$ bins, each containing one item of size $\frac{2}{5}$ and one of $\frac{3}{5}$, while the solution produced by *BEST-FIT* will consist of $9n$ bins, $3n$ with pairs of items of size $\frac{2}{5}$ and $6n$ with a single item of size $\frac{3}{5}$. As for *INCR*, for each pair of unpaired items of size $\frac{2}{5}$, it will be using one less bin than *BEST-FIT*, for a total of $9n - 3n\alpha$. Thus the ratio at the final step will be $\frac{9n-3n\alpha}{6n} = \frac{3-\alpha}{2}$.

The midway point and the final point represent the points at which *INCR* will perform the worst on this sequence, attempting to minimize the maximum of the two leads us to the optimal value for α, which is $\frac{1}{3}$ by $1 + \alpha = \frac{3-\alpha}{2} \Rightarrow 2 + 2\alpha = 3 - \alpha \Rightarrow 3\alpha = 1$. Thus the incremental ratio for this arbitrarily long sequences is $\frac{4}{3}$. Thus given a constant additive term to ignore, by choice of large enough n, we can find a request sequence with an asymptotic incremental ratio of $\frac{4}{3} - \epsilon$, for arbitrary $\epsilon > 0$.

We cannot conclude at this point that the asymptotic global incremental ratio and global incremental ratio necessarily differ for the bin-packing problem, as both proofs are only for lower bounds. However, given that two measures cannot be shown to be identical for this problem by way of a trivial construction proof, unlike the above problems, the possibility is still open.

4.5 Criticisms of the Adversarial Constraint

As has been previously mentioned, in the work of Sharp [32], the competitive ratio is found to be comprised of two distinct contributing factors, the incremental constraint and the adversarial constraint. Previously, online algorithms have been thought to be out-performed by offline algorithms because they are blind to the future and so can make decisions that, while sensible at the time that they are carried out, are non-optimal for the entire sequence. This is what is referred to as the adversarial constraint. Sharp points out that this is not the only handicap that an online algorithm works under. Online algorithms also have to provide intermediate solutions. While they are ultimately judged on their solution for the entire sequence of input, online algorithms have to provide solutions that are reasonable on each step since they do not know how long the sequence of input is going to be.

This leads us to incremental algorithms. To determine how much of the global competitive ratio of a problem is a result of having to satisfy this incremental constraint, we compute the global incremental ratio for that problem, and then determine the adversarial constraint as the difference between the two. For example: if an online problem has a global competitive ratio of α, and its incremental version has a global incremental ratio of β, then the incremental constraint of the online problem is β and its adversarial constraint is $\gamma = \alpha - \beta$.

The first apparent shortcoming with this method of computing the adversarial constraint become apparent when computing it for the paging problem, a $k-$competitive problem that has an global incremental ratio of 1.

Theorem

The incremental paging algorithm is 1-competitive.

Proof

Consider the Bélády-min algorithm [5] for finding the ideal sequence of page replacements in a paging problem. For each cache miss in which a page must be ejected from memory, choose to eject the page whose next request is furthest into the future (obviously if there is a page currently in memory that is never requested again in the input sequence of page requests, eject that one).

This produces the optimal sequence is terms of cache misses. Further more, note that at no point where a page miss occurs was there a way to prevent a page miss at that point without causing one or page misses earlier in the sequence. If it is the first time a page x has been requested, obviously the only way to have avoided that page miss would have been to take on the cost of loading x into memory before the request for x, which would put the incremental ratio of such a solution above 1 on the step at which the unnecessarily early step occurred. If x had been in memory before; but has since been kicked out, since we only would have kicked out x when it was the furthest request away, if at the time, another page, say y, had been kicked out instead, a request for it would have occurred before x, and thus the incremental ratio of such a solution would also exceed 1 on the step at which y was requested. Thus, this algorithm has an incremental ratio of 1. ■

Given then that the paging problem is 1-incremental, and k-competitive

[33], does that mean that it is $k-1$-adversarial? While the definition of the adversarial constraint would say yes, our intuitive interpretation of what we want the relationship between the incremental constraint and the adversarial constraint to mean says no. What a problem being 1-incremental tells us is that the incremental constraint does not contribute anything to the competitive ratio. What we want is for the conclusion to be that paging is k-competitive, with all of its 'poor' performance being due to it being blind to the future and thus subject to the manipulations of an adversary. What we want is for paging to thus be k-adversarial. Even with 1-incremental problems treated as a special case, does it make sense to consider the adversarial constraint of a say, 6-competitive algorithm to be 3 if it is also a 3-incremental problem? 2-adversarial would seem to make more sense.

Seeing as the competitive ratio represents a factor of multiplication over the original, a more sensible means of computation might be to use division instead, i.e. given a $\alpha-$competitive and $\beta-$incremental problem, we should conclude that it is $\gamma = \frac{\alpha}{\beta}$-adversarial. We note here that this will work for both maximization and minimization problems. Obviously, if $\alpha \geq \beta \geq 1$, (as is the case in a minimization problem) then $\alpha \geq \gamma = \frac{\alpha}{\beta} \geq 1$ as well. As for maximization problems, when $\alpha \leq \beta \leq 1$, it follows that $\alpha \leq \gamma = \frac{\alpha}{\beta} \leq 1$ as well.

Using this new method of computation, here are the revised values of the adversarial constraints of the problems discussed in chapter 2.

	Original	Revised
Bipartite Matching	$\frac{1}{6}$	$\frac{3}{4}$
Minimum Colouring	$\frac{2}{\log^2 n} - O(\frac{n(\log\log n)^2}{\log^3 n})$	$O(\frac{\log n}{n(\log\log n)^2})$
Steiner Tree	$\Omega(\frac{\log n}{\log\log n})$	$\Omega(\frac{\log n}{\log\log n})$
Vertex Cover	0.834	1.715
Set Cover	$\Omega(\frac{\log n \log m}{\log\log n+\log\log m}) - \Theta(\log n)$	$\Omega(\frac{\log m}{\log\log n+\log\log m})$
Metric k-center	0	1
Network Flow	0	1

For the rest of this discussion, we shall assumed we are using the revised version of the adversarial constraint and the incremental constraint.

In both the original work on incremental constraints, and in the above revisions to it, the assumption has been that the incremental and adversarial constraints on a problem are exhibited on every problem instance. Again, paging is the source of the counter-example.

As is discussed in the beginning of this chapter, paging can be considered as a special case of k-server problem. Paging also has an incremental constraint of 1, as is proven as the beginning of this section. Thus, because the paging problem with a cache size of 2, it is 2-competitive. It also has an incremental constraint of 1 and thus an adversarial constraint of 2.

Meanwhile, in the proof of the lower bound of the incremental constraint of the k-server problem, we have a problem instance with two servers and an incremental constraint of 2. Since the global competitive ratio for the 2-server problem is already known to be 2 [29], and the global incremental ratio cannot exceed the global competitive ratio (as was proven back in the second chapter), we can conclude that for the 2-server problem the incremental constraint must be 2.

So, we find ourselves with two types of problem instance for the 2-server problem, one type with an incremental ratio of 2 and the other with an incremental ratio of 1, and both with a competitive ratio of 2. If we just look at the first type of problem, we would conclude that the adversarial constraint is 1 and that improved prediction cannot help us improve the performance of any online algorithms for the k-server problem. If we looked at the second, we would conclude that improved prediction was the only way to improve performance.

The former view can be summarized as the idea that a small to non-existent gap between the global incremental ratio and the global competitive ratio indicates improved performance cannot come from improved predictions. If by improved performance, we strictly mean an improved competitive ratio, then this view is entirely correct. However, again as paging well-illustrates, there is often a significant gap between the theoretical worst-case performance of an online algorithm and its performance in practice. In practice, improved prediction aids implementations of paging, and by extension, k-server.

Thus, while the computed size of the adversarial constraint provides a bound on the possible decreases (or increases) to the competitive ratio of the problem through improved predictions, these limitations may not apply with sufficient likelihood in practice. Since the incremental ratio is not necessarily constant over all problem instances, there may still be practical improvements to be made to the performance of online algorithms.

In the case of 2-server, the problem instances divide themselves nicely into two cases, those with an incremental constraint of 1 and that with an adversarial constraint of 1; but can we find a better, more general, method? Operating from the assumption that the competitive ratio is composed of both incremental and adversarial components, instead of treating the adversarial component as being constant across the entire problems space, let us instead compute it on a per problem instance basis, taking the maximum over the space of problems.

Definition The **adversarial ratio** for a request sequence ρ and an online algorithm A is defined as $\frac{A(\rho)}{INCR(\rho)}$. The **global adversarial ratio** is then defined as $\max_{\rho} \frac{A(\rho)}{INCR(\rho)}$.

Let us apply this definition to some of the problems that have been examined so far.

In the case of the paging problem, and by extension the k-server problem, since every problem instance of the standard paging problem has a solution with incremental ratio of 1, and every online algorithm for the paging problem is no better than k-competitive, we can conclude that both paging and k-server, have a global adversarial ratio of k. This follows what we wanted our definition to

achieve in that it identifies that there are problem instances for which a factor of k is being contributed to the competitive ratio due to the online algorithm's obliviousness.

In the standard online algorithm for the vertex cover problem, as each edge arrives, if it is not already covered by a vertex, both of the edge's vertices are added to the cover. This algorithm is known to be 2-competitive. It is not difficult to devise a simple example in which the incremental ratio is 1; but this algorithm produces a solution which is 2-competitive. As an example, consider three edges joined together to form a line, with the edges arriving in order from left to right. As the first edge arrives, both of its vertices are added to the cover. The second edge, already covered, adds no new vertices to the cover. As the third edge arrives, as it is uncovered, both of its vertices are added to the cover. Thus this cover will consist of 4 vertices, while the incremental algorithm would have added only the interior vertices for both the first and third edges, for a total of 2.

Turning our attention to a very different problem, what would the global adversarial ratio of the k-center problem be? Consider that if a problem is 1-incremental, it implies that there exists a series of solutions, for each value from 1 though k, each of which is optimal on that step. In the case of the online k-center problem, while the ultimate number of centers that will be placed is unknown, the position of all points in the space is known at the outset. By choosing the optimal solution at each step, choosing to break ties in favour of whichever would be compatible with optimal solutions on additional steps, an online algorithm would, necessarily, match the incremental optimal algorithm in cases where the incremental ratio is 1. Unlike earlier cases, we are unable to find request sequences with incremental ratios of 1 that display the worst case competitive ratio. Does this mean that the adversarial ratio of k-center is 1? No; but it does inform us that in cases where the online algorithm has a competitive ratio of greater than 1, the incremental constraint must also be greater than 1.

To see that the adversarial constraint of the k-center problem is in fact greater than one, consider the simple case of three points on a Euclidean plane, with the one point midway between the other two. Our request sequence will stop after the placement of the second center point. Obviously, the optimal solution on the first step is to place our center on top of the middle point. On the second step, the optimal solution consists of points midway between the midway point and the two outer points. Thus an online algorithm that chooses the optimal point it can on each step, having placed its first center on the midway point, would find itself with a competitive ratio of 2 when the sequence ended on the next step, as there would be nowhere for it to place its second center so as to decrease the maximum radius. The following diagram illustrates what the incremental optimal algorithm would do in this situation.

The numbered points correspond to the centers placed by the incremental optimal algorithm. If we assume that the distance from the midway point to the outer points is r, then let the distance from the midway point to the two centers be αr. Thus on the first step, *INCR* would display an incremental ratio of $1+\alpha = \frac{r+\alpha r}{r}$, and on the second an incremental ratio of $2-2\alpha = \frac{r-\alpha r}{\frac{r}{2}} = \frac{2r-2\alpha r}{r}$. Minimizing these ratios leads us to a value of α of $1 + \alpha = 2 - 2\alpha \Rightarrow 1 = 3\alpha \Rightarrow \alpha = \frac{1}{3}$. Thus while the incremental ratio of this problem instance is $\frac{4}{3}$, the competitive ratio of the online algorithm is 2, leading to an adversarial ratio of $\frac{3}{2}$.

For online problems that are not competitive, i.e. problems that do not have bounded competitive ratios, such as minimum colouring, set cover and Steiner tree, calculation of the adversarial constraint is unlikely to add any new insights, particularly in the case of Steiner tree where a bounded incremental algorithm is known, while the problem itself has no bounded competitive ratio.

4.6 Directions for Future Work

The immediately obvious direction for future work is to find an upper bound on the incremental constraint of k-server. I suspect that the value of the incremental constraint on the symmetric k-server problem is either two or k; but have been unable to prove it. Some possible intermediate steps prior to proving an upper bound in the general case include proving an upper bound in restricted sub-cases, perhaps following the same progression of sub-cases as results on the competitive ratio of k-server, i.e. starting with the (n-1)-case, or on a line.

Bibliography

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 100–105, 2003.

[2] N. Alon and Y. Azar. On-line steiner trees in the euclidean plane. *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 337–343, 1992.

[3] Yair Bartel and Edward Grove. The harmonic k-server algorithm is competitive. *Journal of the ACM*, 47(1):1–15, 2001.

[4] Wolfgang Bein, Marek Chrobak, and Lawrence L. Lamore. The 3-server problem in the plane. *Theoretical Computer Science*, 287:387–391, 2002.

[5] L. A. Bélády. A study of replacement algorithms for a virtual storage computer. *IBM Systems Journal*, 5:78–101, 1966.

[6] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Widgerson. On the power of randomization in on-line algorithms. *Proceedings 22nd annual symposium on Theory of Computing*, pages 379–386, 1990.

[7] Allan Borodin, Nathan Linial, and Michael Saks. An optimal online algorithm for metrical task systems. *Proceedings 19th annual Symposium on Theory of Computing*, pages 373–382, 1987.

[8] Allan Borodin, Nathan Linial, and Michael Saks. An optimal online algorithm for metrical task system. *Journal of the ACM*, 39:745–763, 1992.

[9] B. Chandra. Does randomization help online bin-packing? *Information Processing Letters*, 43(1):15–19, 1992.

[10] Marek Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1990.

[11] Marek Chrobak and Lawrence L. Larmore. An optimal online algorithm for k-servers on tress. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[12] Marek Chrobak and Lawrence L. Lawrence. Generosity helps or an 11-competitive algorithm for three servers. *Journal of Algorithms*, 16:234–263, 1994.

[13] D. Coppersmith, P. G. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs and applications to online algorithms. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 369–378, 1990.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, third edition, 2009.

[15] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[16] A. Fiait, Y. Rabini, Y. Ravid, and B. Schieber. A deterministic o(k^3)-competitive k-server algorithm for the circle. *Algorithmica*, 11:572–578, 1994.

[17] A. Fiat, Y.Rabani, and Y. Ravid. Competitive k-server algorithms. *Proceedings 31st annual symposium on Foundations of Computer Science*, 2:454–463, 1990.

[18] M. R. Gary and D. S. Johnson. *Computers and Intractibility; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[19] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[20] Edward Grove. The harmonic k-server algorithm is competitive. *Proceedings 23rd annual symposium on Theory of Computing*, pages 260–266, 1991.

[21] M. M. Halldórsson. A still better performance gaurantee for approximate graph coloring. *Information Processing Letters*, 12(1):19–23, 1993.

[22] M. M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete algorithms*, pages 211–213, 1992.

[23] J. Hartline and A. M. Sharp. Hierarchical flow. *Proceedings of the 2nd International Netowrk Optimization Conference*, pages 681–687, 2005.

[24] J. Hästad. Some optimal inapproximability results. *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 1–10, 1997.

[25] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[26] W. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Math*, 1:209–215, 1979.

[27] Richard M. Karp, U. V. Vazirani, and V.V. Vazirani. An optimal algorithm for online bipartitie matching. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[28] Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. *Proceedings 26th annual Symposium on Theory of Computing*, pages 507–511, 1994.

[29] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for online problems. *Proceedings of the 20th annual symposium on the Theory of Computing*, pages 322–333, 1988.

[30] B. Monien and E. Specenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22(1):115–123, 1985.

[31] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

[32] A. M. Sharp. *Incremental Algrothms: Solving Problems in a Changing World*. PhD thesis, Cornell University, August 2007.

[33] Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[34] A. Van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43(5):274–288, 1992.