Greedy Algorithms and Why Simple Algorithms Can Be Complex

Allan Borodin

Department of Computer Science, University of Toronto

University of California at Irvine

February 6, 2009

Based on years of pestering colleagues and papers with

Michael Alekhnovich [3]

Spyros Angelopoulos [5]

Josh Buresh-Oppenheim [3]

David Cashman [13]

Stephanie Horn [32]

Russell Impagliazzo [19]

Brendan Lucier [43]

Avner Magen [3], [13]

Morten Nielson [14]

Toniann Pitassi [3]

Charles Rackoff [14]

Yuli Ye [49]

A Theory for All Algorithms

Discrete computation : Church-Turing computability

Computation over the reals : Blum, Shub, Smale model vs the oracle bit Turing machine model.

Computer Science is the systematic study of algorithms.

-Richard Karp in "Computing, 2016: What Won't Be Possible?" symposium.

From Steve Cook's 1999 CRM-Fields talk "The Achievements and Challenges of Computational Complexity"

Proving time **upper** bounds

Rich, well-developed methodology:

Linear Programming
Dynamic Programming
Greedy Algorithms
Divide and Conquer Algorithms,...

Proving time **lower** bounds

Methods: Diagonalization; Reduction; Counting

No success for common practical problems which appear difficult

Scheduling and other NP-complete problems
Factoring integers ...

Almost Ten Years Later: Half full or half empty?

We (in theory) continue to equate:

"efficient" with polynomial time by a Turing machine
"very efficient" with quasi-linear time $O(n \log^k n)$ by a RAM

Complexity theory has been very successful in many regards ....but
We still do not have provable results showing that some explicit NP problem *cannot* be computed in say time $O(n \log n)$.

Algorithm design does indeed have well-developed and widely applicable methodologies but ...

How well do we understand the well developed algorithmic methodology?

Ambitious goal: A systematic study of *"simple"*, *"combinatorial"* algorithms for combinatorial search and optimization problems.

More modest goal: Just try to understand the limitations and power of "greedy" and "greedy-like" algorithms.

Results are from a worst case perspective. But the definitions of the algorithmic models and their potential applicability do not depend on the worst case perspective, and hopefully can go beyond discrete search and optimization.

Motivation

Conceptually simple algorithms are usually the first thing one tries and sometimes these simple algorithms provide good (or even optimal) results or at least benchmarks. In some applications, simplicity may almost be necessary. Do such simpe algorithms have *provable* limtations?

The Design and Analysis of Algorithms is arguably the main undergraduate *theory course* in most CS departments. Many courses and texts organize the material in terms of undefined (albeit intuitively understood) conceptually simple algorithmic paradigms (or meta-algorithms) such as greedy, dynamic programming, ... This is a theory course, where the basic objects of study, the algorithmic paradigms, are generally not defined! (But we do prove theorems about particular algorithms for particular problems.)

Occasionally, good students may ask (or I pretend they ask):

- Is Dijkstra's shortest path algorithm a greedy algorithm or is it dynamic programming?
- What is the difference between divide and conquer and dynamic programming?
- Do we need dynamic programming for shortest paths when edge costs can be negativei or is there a Dykstra-like algorithm? Or is there a greedy algorithm?
- Can we efficiently compute maximum (bipartite) matching by dynamic programming?
- Is there any relation between combinatorial optimization and numerical optimization?

Anti-motivation part 1: Why do this?

" .... trying to define what may be indefinable. .... I shall not today attempt further to define the kinds of material...But *I know it when I see it ...* " U.S. Supreme Court Justice Potter Stewart in discussing obscenity, 1964.

Samuel Johnson (1709-1784): All theory is against freedom of the will; all experience for it.

Anonymous students (and colleagues): Who cares what kind of an algorithm it is?

Bellman [11] 1957: (in the spirit of Samuel Johnson)

We have purposely left the description a little vague, since it is the spirit of the approach to these processes that is significant, rather than a letter of some rigid formulation. It is extremely important to realize that one can neither axiomatize mathematical formulation nor legislate away ingenuity. In some problems, the state variables and the transformations are forced upon us; in others, there is a choice in these matters and the analytic solution stands or falls upon this choice; in still others, the state variables and sometimes the transformations must be artificially constructed. Experience alone, combined with often laborious trial and error, will yield suitable formulations of involved processes.

Anti-motivation 2: And hasn't it all been done before?

Socrates: I know nothing except the fact of my ignorance.

The next few slides attempts to show that my ignorance is not complete. But (literally) every day I discover more relevant work, some recent and some not so recent.

Small sample of some previous and more current work along these lines.

Some restricted classes of algorithms:

Graham [27] (list scheduling algorithms 1966), Chvátal [17] (branch and bound 1980), Helman and Rosenthal, Helman [29, 28] (DP + branch and bound 1985, 1989), Khanna, et al [35] (local search 2000), Lovász and Schrijver (1991), Sherali and Adams (1990), Lasserre(2001), Arora, et al [7, 8] [2002,2006] (lift and project 2002,2006), Orecchia et al [40] (cut-matching game 2008).

Related area of restricted models:

comparison based algorithms; oracle models ; online algorithms; data stream algorithms.

Proof complexity of restricted proof systems

For example, resolution refutations.

A well developed theory of greedy algorithms?

The term *greedy algorithms* (attributed by Edmonds to Fulkerson) seems to have been first used in print by Edmonds [22]. Hoffman [31] describes such algorithms in the context of LP as follows:
There are a small number of (say maximizing) linear programming problems which can be solved very easily. After (possibly) renumbering the variables, the algorithm successively maximizes $x_1, x_2, \ldots$ meaning that if $\bar{x}_1, \ldots, \bar{x}_k$ have already been determined then $\bar{x}_{k+1}$ is the largest value of x_{k+1} such that this partial solution can be extended to a vector in the desired polytope.
Hoffman states that "in ancient times" (e.g. in inventory theory) such algorithms were called *myopic*.

Matroids, greedoids, k-independence systems, polymatroids, submodular functions and *the* natural greedy algorithm.

Beautiful development starting in the 1950's with the work of Rado [46], Gale [21] and Edmonds [22, 23], (extended by Korte and Lovász [37, 38], and others) as to contexts in which *"the natural"* greedy algorithm will produce an optimal solution. In particular, matroids characterize those hereditary set systems for which the natural greedy algorithm (determined by the order $c_1 \geq c_2 \ldots$ for maximization) will optimize a linear objective function $\sum c_i x_i$ for a maximum independent set $\{i : x_i = 1\}$ in a matroid $M = (E, \mathcal{I})$ where $\mathcal{I}$ are the independent subsets of E. Here the best known example is perhaps the minimum (or maximum) spanning tree problem where the edges of a graph are the elements and the indepedent sets are forests in the graph. Kruskal's greedy algorithm is the natural greedy MST algorithm.

Greedoids are set system relaxations of matroids for which *the* greedy algorithm has been studied in terms of optimal algorithms. Polymatroids extend matroids from $\{0, 1\}$ vectors to real vectors providing a linear programming framework which can be solved optimally by a natural matroid greedy algorithm extension. Series-parallel network flows can be solved by the same natural greedy algorithm which can be viewed as a combinatorial gradiant method.

Transportation problems, Monge sequences and greedy algorithms. In a related development, beginning with the 1963 work of Hoffman [30] on the transportation problem, a necessary and sufficient condition (the existence of a Monge sequence) determines when the transportation problem can be solved greedily (using the fixed order of the Monge sequence).

Maximizing a submodular function

Canonical greedy for the maximization of modular and submodular functions subject to a matroid (and more generally k-independence) constraint following Edmonds and starting with Jenkyns [33], Korte and Hausmann [36], and Fisher, Nemhauser and Wosley [45]. Recent interest due to application to combinatorial auctions (CAs).

$S := \emptyset$
While $E \neq \emptyset$
    Let $e^* = argmax_{e \in E} f(S \cup \{e\}) - f(S)$
    If $S \cup \{e^*\} \in \mathcal{I}$ *then* $S := S \cup \{e^*\}$
    $E := E - \{e^*\}$
End While

This canonical greedy algorithm provides a k (resp. $(k+1)$)-approximation for maximizing modular (resp. submodular) objective functions subject to a k-independence set system $(E, \mathcal{I})$.

Greedy is an adjective

However, the term greedy algorithm has taken on an intuitive meaning and this is my starting point for thinking about a more systematic study of algorithms. Greedy (and local search) algorithms are perhaps the simplest algorithms to design yet not easy to understand their ultimate power and limitations.

I will present a precise model, *priority algorithms* (Borodin, Nielsen, Rackoff [14]), that I believe captures many (but defintely not all) common greedy algorithms. In hindsight, our priority model can be viewed as directly building on the work of Dechter and Dechter [20] which was recently brought to our attention. Extending the priority algorithm formulation, we can also obtain models for some basic types of dynamic programming, backtracking, and primal dual algorithms.

Priority algorithms to model greedy algorithms and as a starting point for other algorithmic frameworks.

"Informally, a greedy algorithm solves a global optimization problem by *making a sequence of locally optimal decisions*".

But decisions about what?

I claim that most greedy algorithms make decisions about input items and then we can rephrase the essential aspect of greedy algorithms as follows: Consider input items (e.g. jobs in a scheduling problem) in some order and make an irrevocable decision (e.g. if and where to schedule a job) for each input item.

More formally, U = universe of possible input items; D = (finite) set of decisions about items; problem $\mathcal{F}$ = is a family of functions $\{F_n : (U \times D)^n \to \Re \cup \{+\infty, -\infty\} | n \geq 1\}$; $I \subset U, |I| = n$ is an input instance of size n; a solution to instance $I = \{u_1, \ldots, u_n\}$ is an assignment of decisions $\{(u_1, d_1), \ldots, (u_n, d_n)\}$.

Contrasting view in DasGupta, et al text: "Greedy algorithms build up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit." But what does this mean for unit profit or min makespan?

To make the priority algorithm framework precise for a problem:

1. Specify the input item representation; e.g. for graph problems, items can be vertices (represented by adjacency list) or edges; for CNF-SAT, items are variables represented by the clauses in which they appear; for scheduling, items are jobs represented by their parameters (e.g. processing time, deadline).
2. Specify a set of possible irrevocable decisions (about an input item); clarify "greedy" decisions as "live for today" (locally optimal) decisions. Hence we view greedy as a special case of priority (i.e. myopic).

3. Specify how input items can be ordered; fixed order (e.g. Kruskal's MST, Graham's online and LPT greedy makespan.) vs adaptive order (e.g. Prim's MST, Dijkstra's shortest path, greedy set cover, submodular function maximization).

   What orderings are allowed? We allow any "locally definable ordering". In hindsight, this turns out to be precisely those orderings that satisfy Arrow's Independence of Irrelevant Attributes (IIA) axiom. Notwithstanding the controversial nature of the IIA axiom in social choice theory, IIA orderings seem quite permissive. For example, we can use any ordering induced by sorting according to the values of *any* function r : {possible input items}$\rightarrow \Re$.

   In an adaptive ordering, this can depend on items (and the corresponding decisions) that have already been seen (as well as those that we know will not be seen).

Avoiding the Turing tarpit

The priority framework (as in the competitive anaylsis of online algorithms where adversarial "nature" determines the ordering) does not have any explicit complexity constraints. Limits on optimality and approximation are independent of the $\mathcal{P} \neq \mathcal{NP}$ conjecture. In practice, greedy and myopic (priority) algorithms tend to be very efficient. Unlike online algorithms and some other previous restricted models, the model definition is not dictated and has to be justified.

Socrates strikes: Independent of [14], priority model is implicitly used in Gonen and Lehmann's [26] multi-unit weighted set packing "polynomial time greedy" inapproximation.

A few examples of algorithms that fit the model

- Kruskal's and Prim's optimal MST algorithms for MST.
- Huffman optimal prefix coding (with tree nodes as items).
- Optimal greedy algorithm for unweighted interval scheduling and 2-approximation algorithms for job interval selection.
- Graham's [27] online and LPT makespan approximation.
- The greedy H_n-approximation set cover algorithm [34, 16, 42].
- The myopic (random SAT threshold) search algorithms [1].
- Various 2-approximation algorithms for vertex cover.
- The (1.52)-approximation for uncapacitated metric facility [44].
- Truthful $O(\sqrt{m})$-approximation for single minded CAs [41].

The "obvious" greedy algorithm doesn't always work well

Johnson, Lovász, Chvátal show that a natural greedy algorithm for weighted vertex cover (set cover greedy algorithm applied to vertex cover) has approximation $\approx H(d)$ for (unweighted) degree d graphs:

$S := \emptyset$

For all $v \in V$

$dc(v) = degree(v)$ % $dc(v)$ will denote current degree

End For

While V is not empty

Let $v = argmin_{u \in V} \frac{w(u)}{dc(u)}$

For all u such that $(u, v) \in E$

$dc(u) := dc(u) - 1$

End For

$S := S \cup \{v\}; V := V - \{v\} - \{u : dc(u) = 0\}$

End While

Clarkson's [18] modified greedy algorithm for vertex cover

$S := \emptyset$;

For all $v \in V$

$dc(v) = degree(v)$ % $dc(v)$ will denote current degree(v)

$wc(v) = w(v)$ % $wc(v)$ will denote current weight(v)

End For

While V is not empty

Let $v = argmin_{u \in V} \frac{w(u)}{dc(u)}$

For all u such that $(u, v) \in E$

$dc(u) := dc(u) - 1$

$wc(u) = wc(u) - \frac{wc(v)}{dc(v)}$

End For

$S := S \cup \{v\}; V := V - \{v\} - \{u : dc(u) = 0\}$

End While

Complicating a simple "solved" problem.

Interval scheduling on m identical machines. In this problem, an input $\mathcal{I}$ is a set of intervals $\{I_1, I_2, \ldots, I_n\}$. The usual (but not the only) representation of an input interval I_i is (s_i, f_i, w_i) where s_i (resp. f_i) is the starting (resp. finishing) time of the i^{th} interval and w_i is the weight or profit of the i^{th} interval (if scheduled). In the unweighted case, $w_i = 1$ for all i. The goal is to maximize the profit of a "feasible subset". Unweighted interval scheduling has an $O(n \log n)$ time optimal greedy algorithm. Weighted case is not quite as easy. The somewhat more general JISP, unweighted job interval scheduling problem, is a MAX-SNP-hard problem where in some sense the same greedy algorithm provides the best known polynomial time approximation ratio.

The Holy Grail: Does the framework lead to new insights and new algorithms?

Greedy for unweighted m machine interval scheduling. [25]

Sort $\mathcal{I} = \{I_1, \ldots, I_n\}$ so that $f_1 \leq f_2 \leq \ldots f_n$

$S := \emptyset$

For $j = 1..m$

    $t_j := 0$ % t_j is the current finishing time on machine j

End For

For $i = 1..n$

    If there exists j such that $t_j \leq s_i$ then

        schedule I_i on that machine j which minimizes $s_i - t_j$;

        $t_j := f_i$

    End If

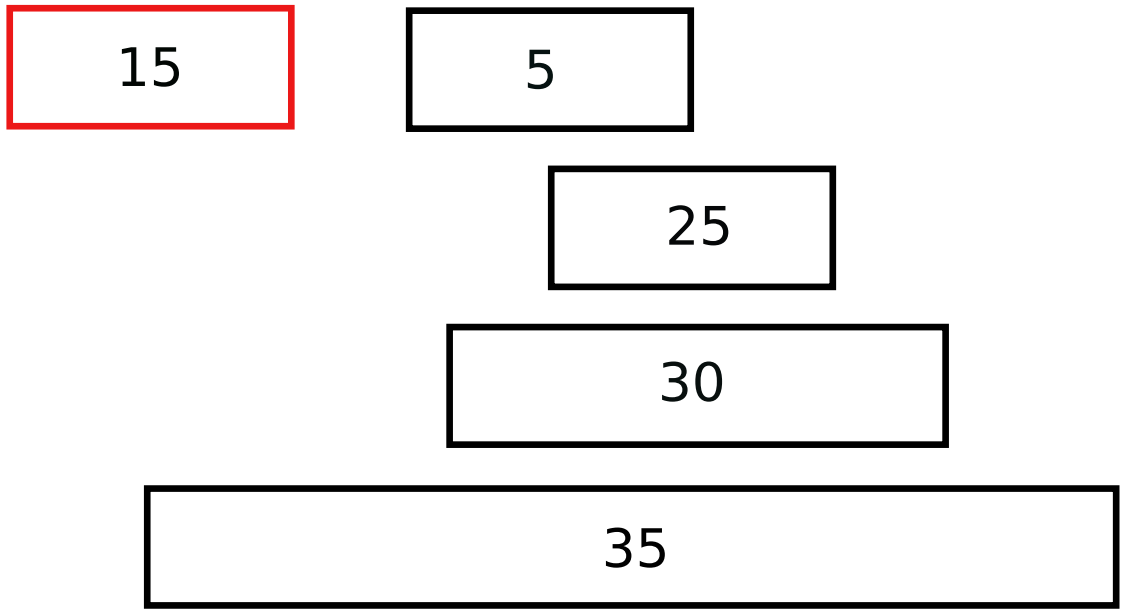
End For

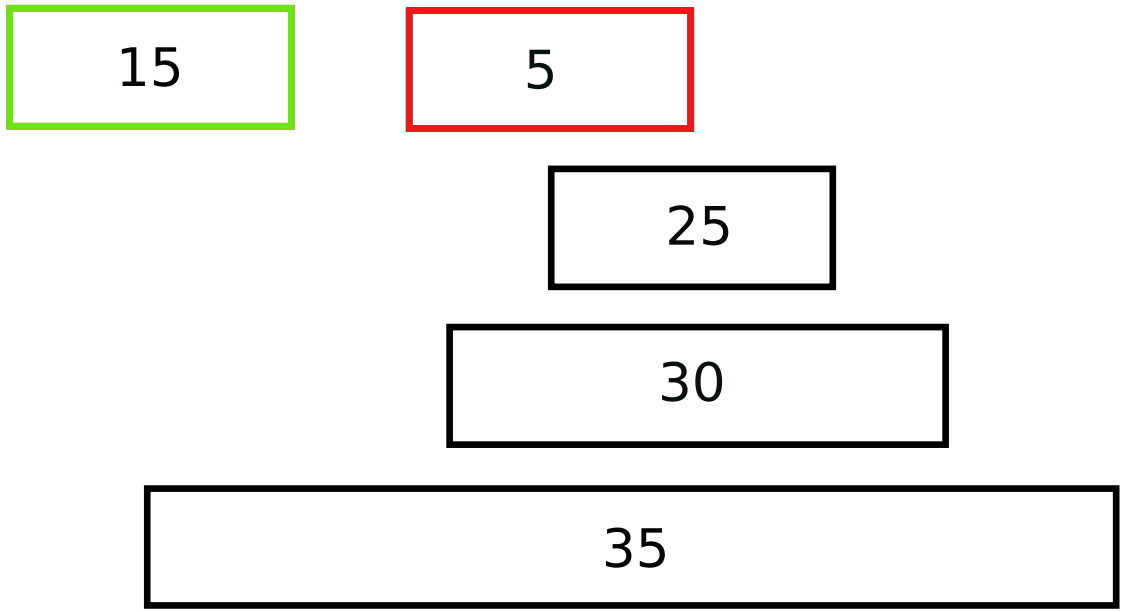
Is this the obvious greedy algorithm?

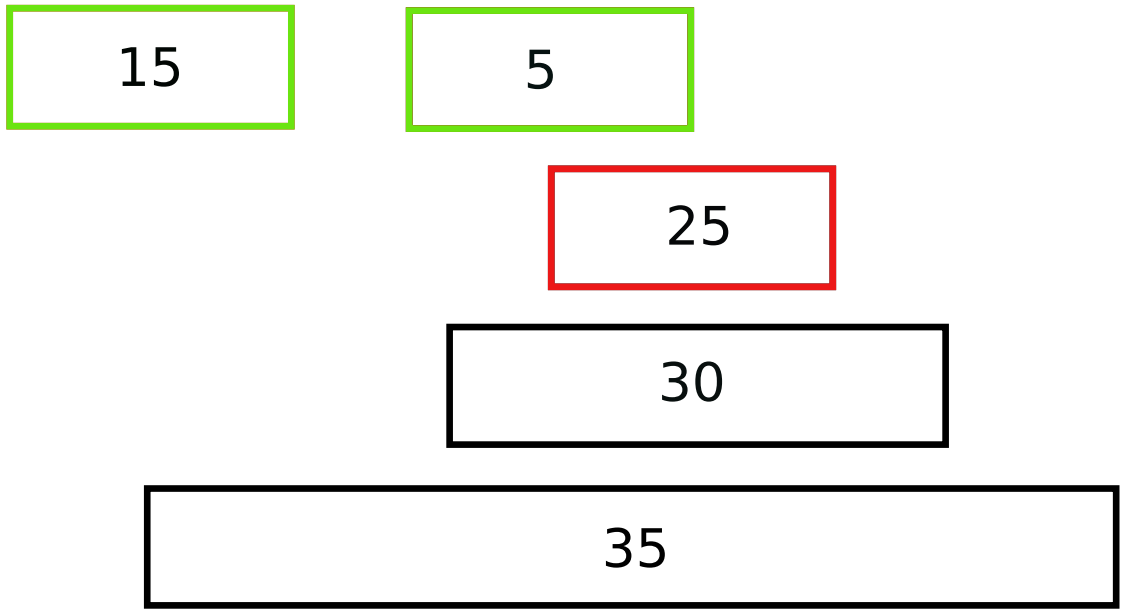
One might think that ordering so that $|f_1 - s_1| \leq |f_2 - s_2| \leq \ldots$ is more natural. (When an interval graph is viewed as a chordal graph, the finishing time ordering is a perfect elimination ordering. For one machine, interval scheduling is the max independent set MIS problem.)

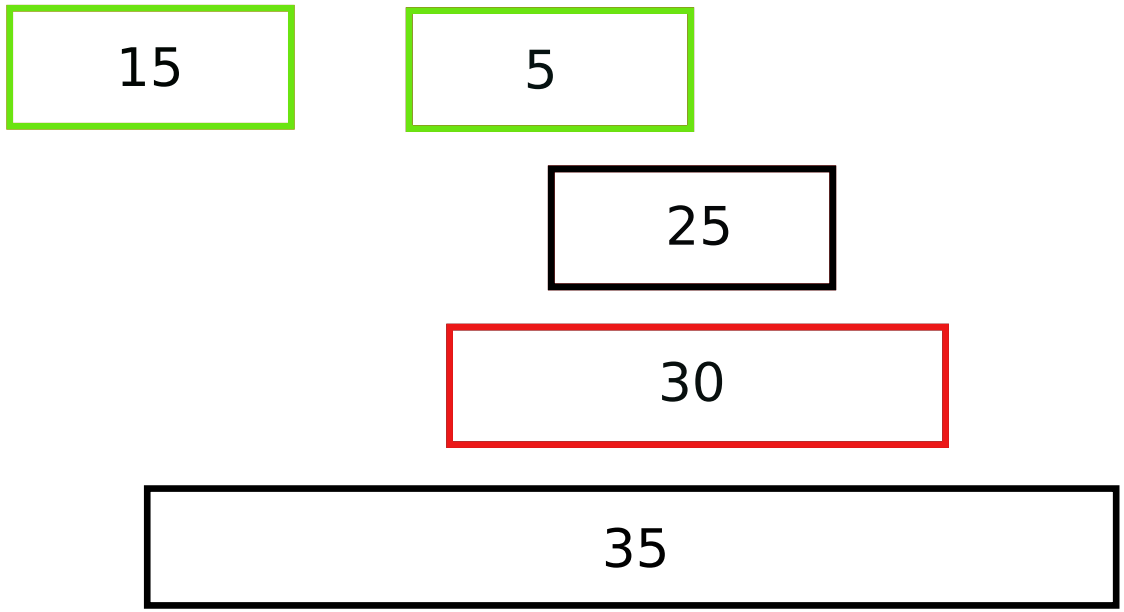
What fixed (or adaptive) orderings might work for *weighted* interval scheduling? Note: feasible subsets are not a matroid, greedoid, or k-independence system.

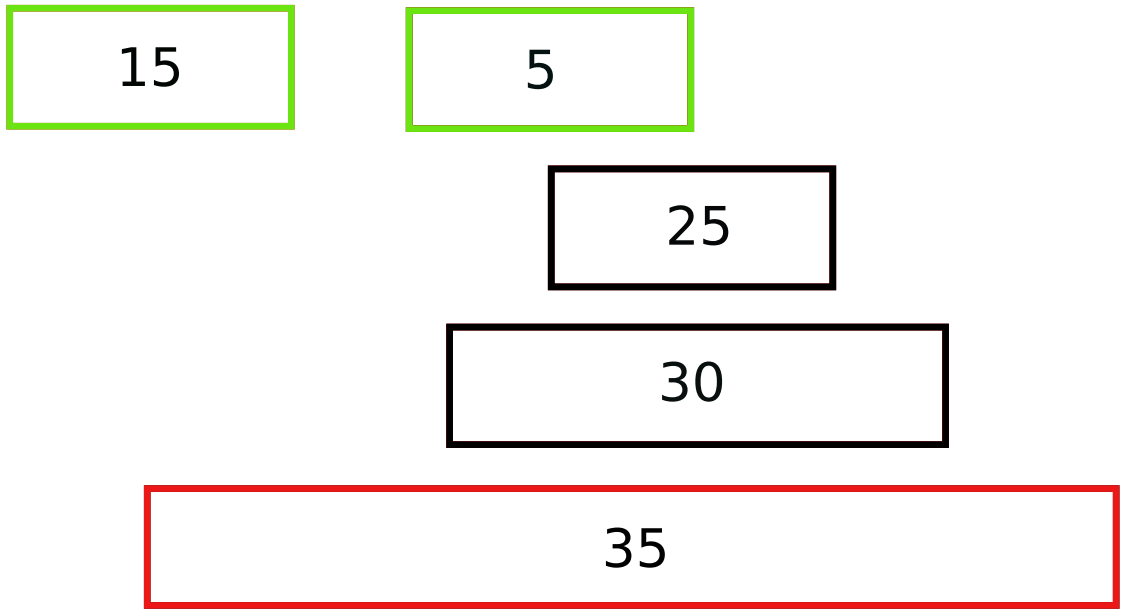
- The unweighted ordering: $f_1 \leq f_2 \leq \ldots f_n$. Unbounded approximation ratio.
- Order so that $w_1 \geq w_2 \geq \ldots w_n$. Unbounded approximation ratio.
- Order so that $|f_1 - s_1|/w_1 \leq |f_2 - s_2|/w_2 \leq \ldots |f_n - s_n|/w_n$. (Most natural greedy?) Unbounded approximation ratio.

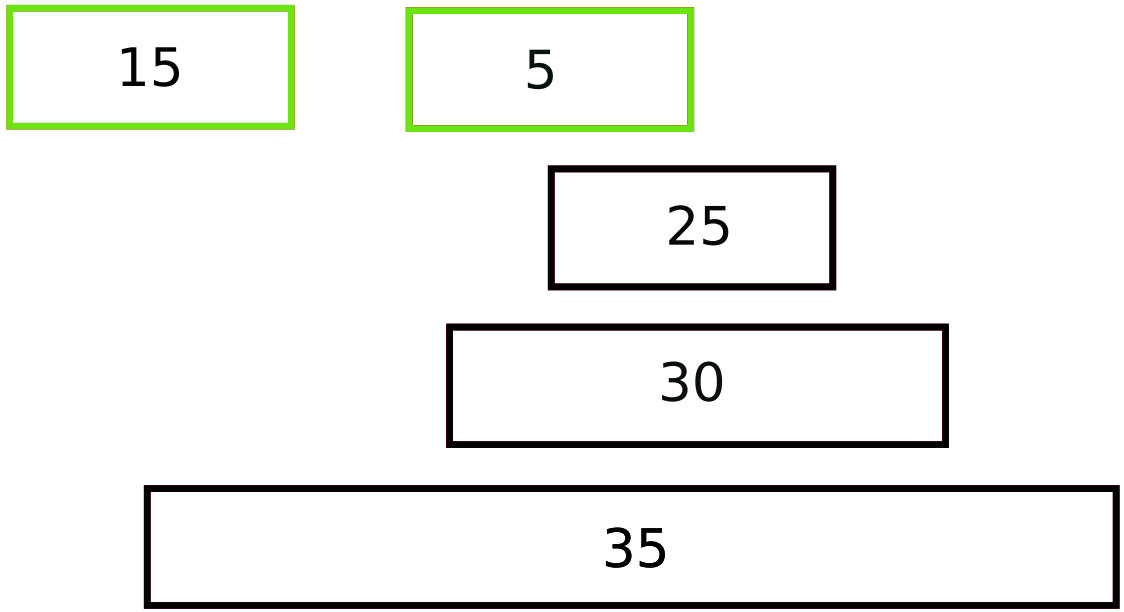

15
5
25
30
35


15
5
25
30
35


15
5
25
30
35


15
5
25
30
35


15
5
25
30
35


15
5
25
30
35

Priority algorithm limitations for interval scheduling

- There does not exist an optimal (adaptive) priority algorithm for the weighted interval scheduling problem even for one machine.

  In fact, the best possible approximation ratio obtainable for this problem will depend on $\Delta = \frac{\max_j(w_j/p_j)}{\min_j(w_j/p_j)}$ where $p_j = f_j - s_j$.

- For proportional profit (when $w_j = p_j$), LPT provides a 3-approximation fixed order greedy algorithm and hence a 3Δ approximation for arbitrary weights. For one machine proportional profit, the best priority approximation ratio is 3.

- The 3-inapproximation bound for proportional profit holds for fixed priority greedy algorithms for any number m of machines.
- For m even, there is an adaptive greedy priority algorithm for proportional profit with approximation ratio 2.
- A 1.56 inapproximation bound for 2 machine proportional profit priority.

Challenge for weighted interval scheduling WISP

There is an optimal dynamic programming algorithm that solves the m machine WISP in time $O(n^m)$ and a time $O(n^2 \log n)$ optimal algorithm [6] based on min-cost max-flow.

What is the best approximation factor possible by an $O(n \log n)$ time algorithm for arbitrary or proportional profit on $m \geq 2$ machines? Can we obtain a "highly efficient FPTAS"; i.e. an $(1+\epsilon)$ approximation in $poly(\frac{1}{\epsilon})n \log n$ steps?

Some other priority inapproximation bounds

- Davis and Impagliazzo [19] derived a number of inapproximations for weighted graph problems. For example, no *fixed order* priority algorithm can have a constant approximation for the single source shortest path problem in contrast to Dijkstra's (adaptive priority) algorithm. As another example, priority algorithms cannot $(2-\epsilon)$- approximate weighted vertex cover.
- Regev [47] proves a *fixed order* priority inapproximation ratio of $\Omega(\frac{\log n}{\log \log n})$ for the makespan problem in the restricted machines model. It is open if a non constant inapproximation bound holds for adaptive priority algorithms. The natural (online) greedy algorithm has approximation $\log n$.

- While there is a truthful greedy allocation mechanism for single minded CAs with $O(\sqrt{m})$ approximation of the social welfare, Brendan Lucier and I show that (even for two minded bidders) any incentive compatible priority allocation (i.e. that can be made truthful) cannot achieve an approximation ratio better than the trivial $\Omega(\min\{n, m\})$. The inapproximation holds for both "bidders" or "bids" as input items. Similarly, although there is a simple 2-approximation for submodular CAs, there cannot be a incentive compatible priority algorithm allocation with bounded approximation when objects are items.
  This perspective has led to a greedy mechanism for general CAs with price of anarchy $O(\sqrt{m})$.

Did we correctly capture *all* greedy algorithms? NO.

A few extensions:

- We can let the algorithm have access to some easily computed global information; eg, the number of input items, or the maximum and minimum length of a job. In the previous negative bound, the result still applies but allowing such information permits some non obvious orderings. One could also say that each input item (i.e. interval) also specifies the number or names of its intersecting intervals.
- Reverse greedy or worst out greedy (vs best in). The densest subgraph 2-approximation algorithm of and Kortsatz and Peleg [39], and Charikar [15] is a reverse greedy algorithm (iteratively removing the vertex of smallest degree) and taking the best of the induced subgraphs.

- We could maintain a small number of partial solutions that we continue to augment. This falls within the pBT model [3] where we show that any "simple DP" will require time and space $\Omega(n^m)$ for m machine weighted interval scheduling.
- We could allow some lookahead. Some forms of lookahead also fall within pBT model but others do not. In particular, ordering may not be IIA and this is more difficult to analyze.
- The local ratio (primal dual) algorithm of [9, 12] can be viewed as a 2 pass algorithm which (using a fixed priority ordering) pushes input items (intervals) onto a stack and then pops the stack so as to create a feasible solution.

Local ratio (simple primal dual) algorithm for weighted interval scheduling on m machines. [9]

Stack := $\emptyset$

For $i : 1..n$

    $w'_i := w_i$ % w'_i will be the current residual profit

End For

While $\mathcal{I} \neq \emptyset$

    Push I_j onto Stack where I_j has smallest finishing time for any interval in $\mathcal{I}$

    For all intervals I_k intersecting I_j

        $w'_k := w'_k - \frac{1}{m} w'_j$

        If $w'_k \leq 0$ then Remove I_k from $\mathcal{I}$

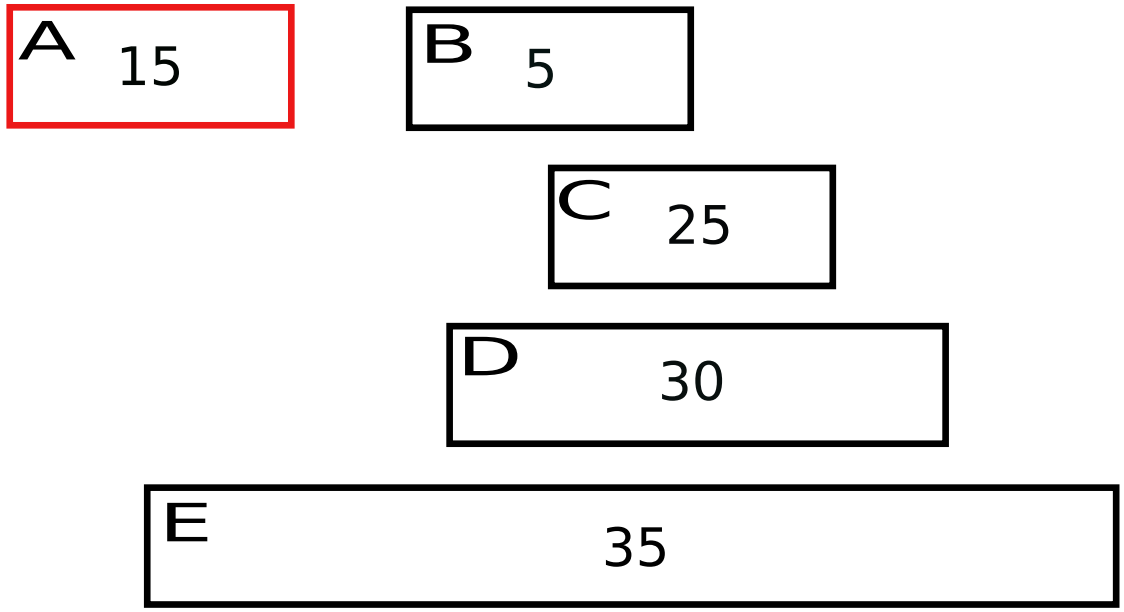
    End For

EndWhile

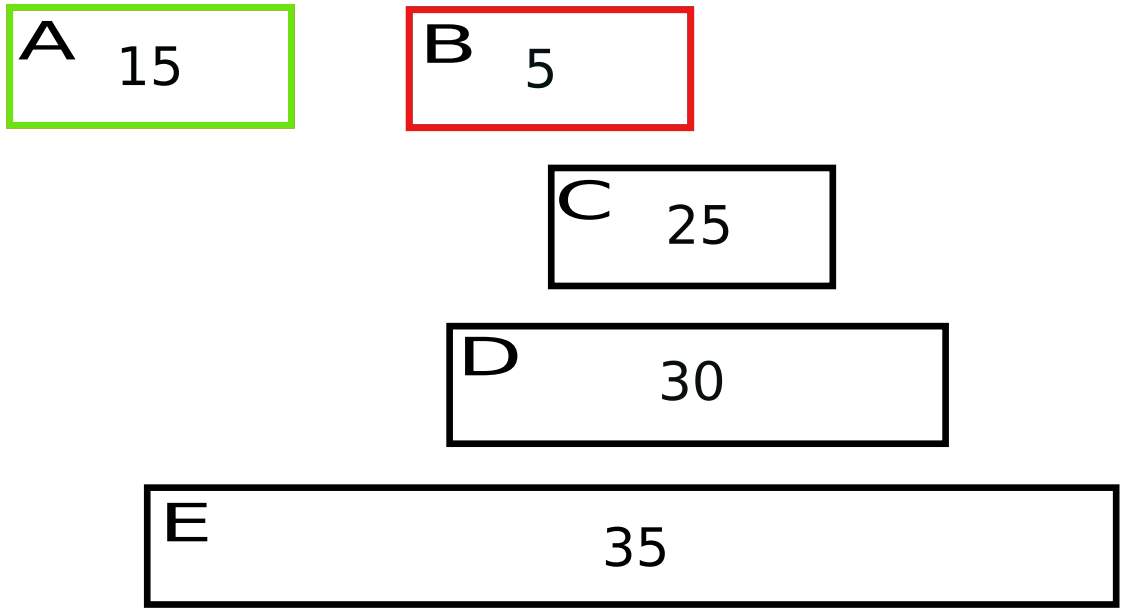
Continuation of local ratio algorithm.

$\mathcal{S} := \emptyset$;
While Stack $\neq \emptyset$
    Pop Stack and let I be interval popped
    If I can be feasibly scheduled on any machine
    (having already scheduled intervals in $\mathcal{S}$)
        Then place I on an available machine; $\mathcal{S} := \mathcal{S} \cup \{I\}$
        Else I is not scheduled.
    End IF
End While

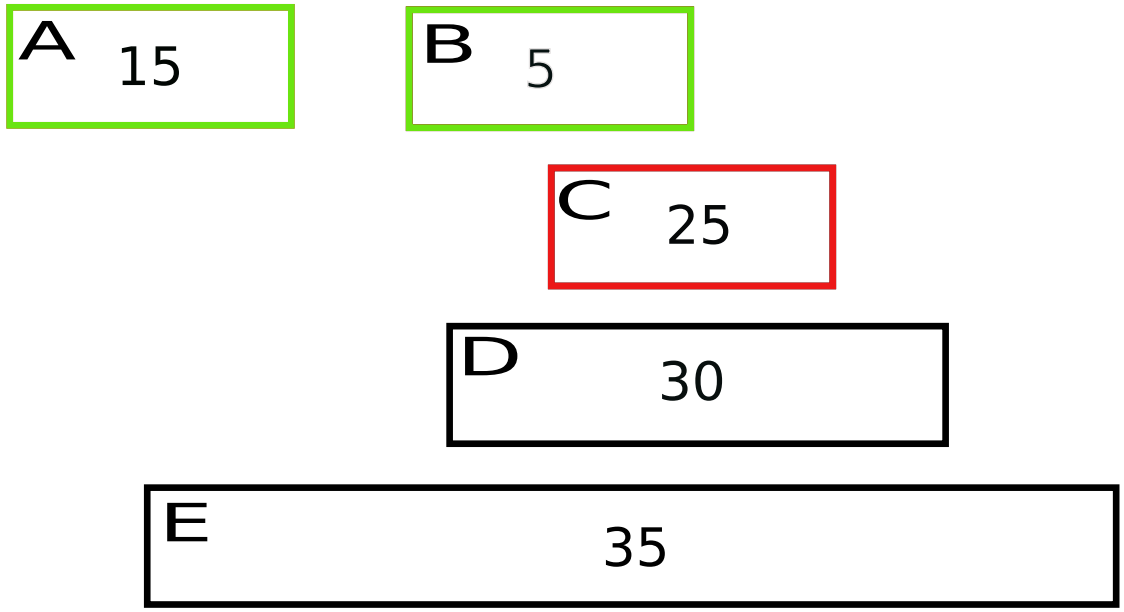
For $m \geq 1$, the approximation ratio is $2 - \frac{1}{m}$ and hence optimal for $m = 1$. [9]



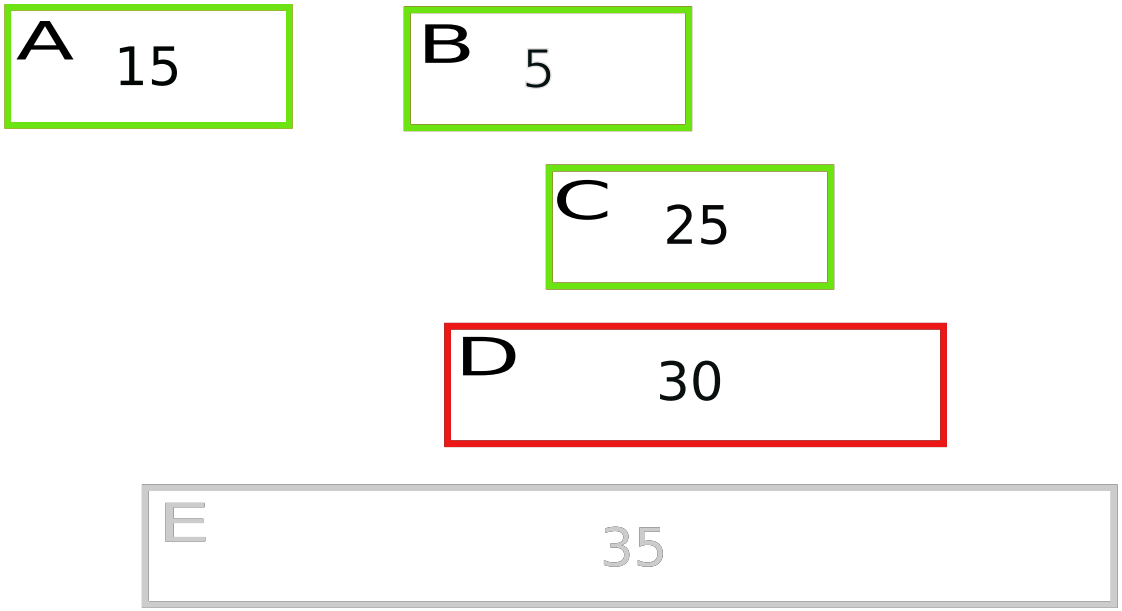
$w_A^{'} = 15$
$w_B^{'} = 5$
$w_C^{'} = 25$
$w_D^{'} = 30$
$w_E^{'} = 35 - 15 = 20$



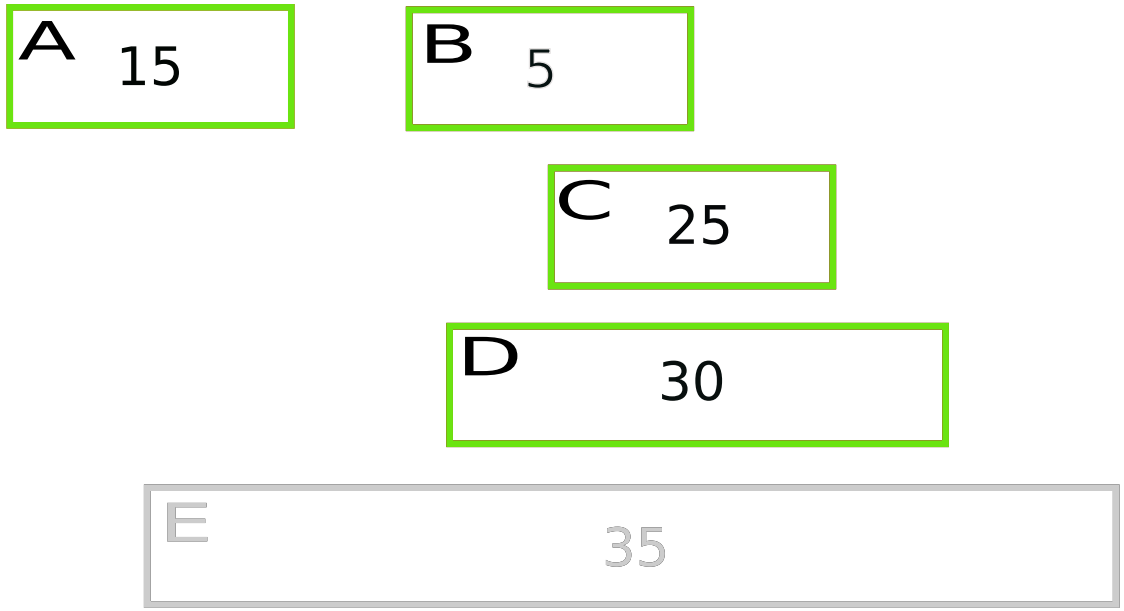
$w_A^{'} = 15$
$w_B^{'} = 5$
$w_C^{'} = 25 - 5 = 20$
$w_D^{'} = 30 - 5 = 25$
$w_E^{'} = 20 - 5 = 15$



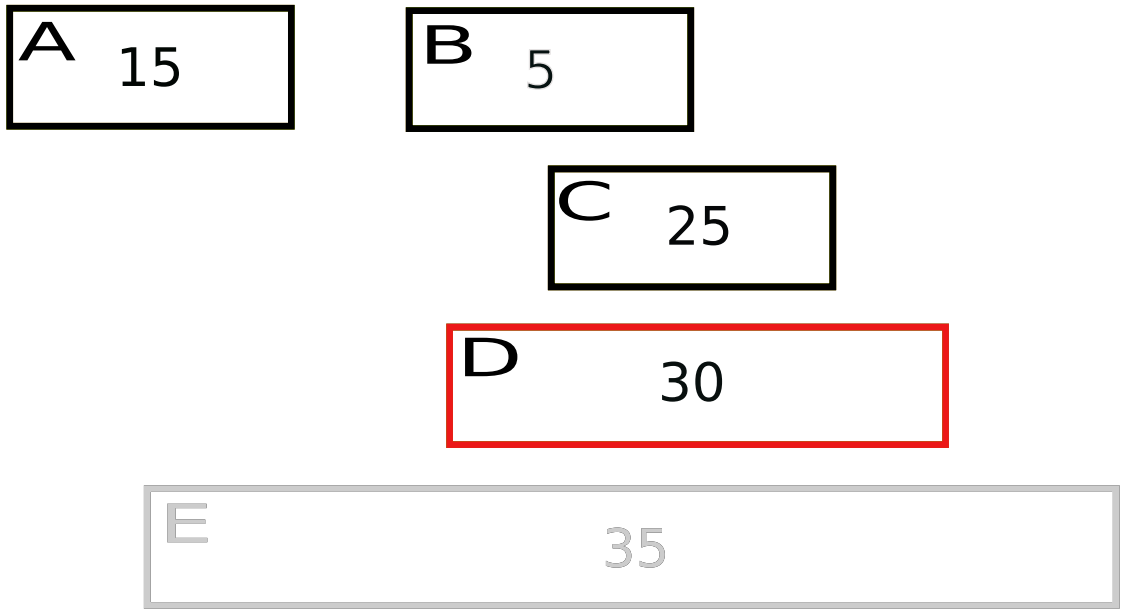
$w_A^{'} = 15$
$w_B^{'} = 5$
$w_C^{'} = 20$
$w_D^{'} = 25 - 20 = 5$
$w_E^{'} = 15 - 20 = -5$

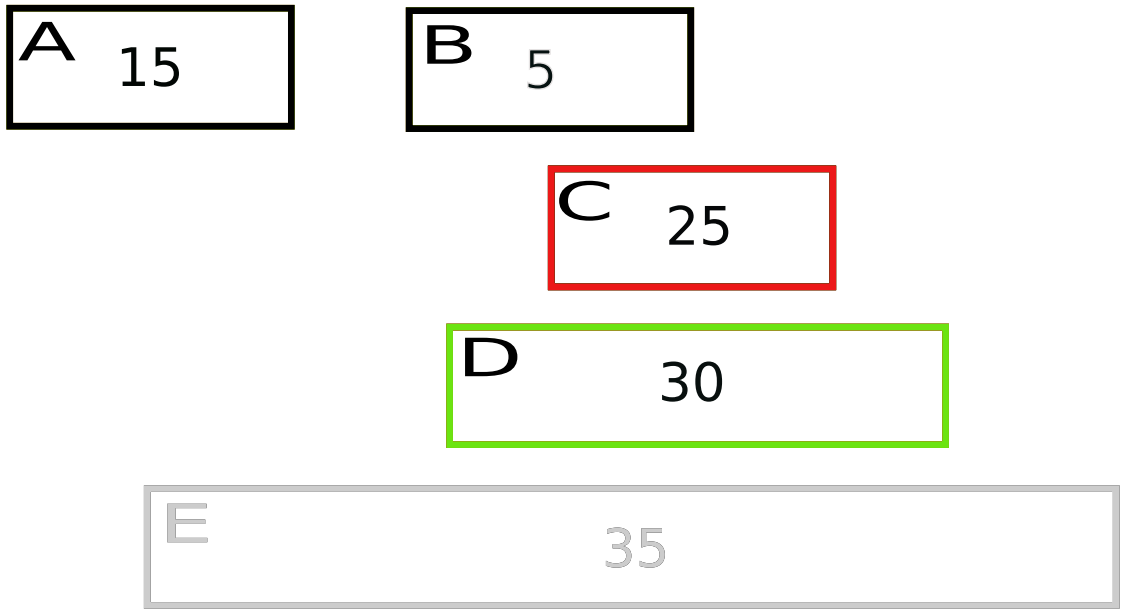


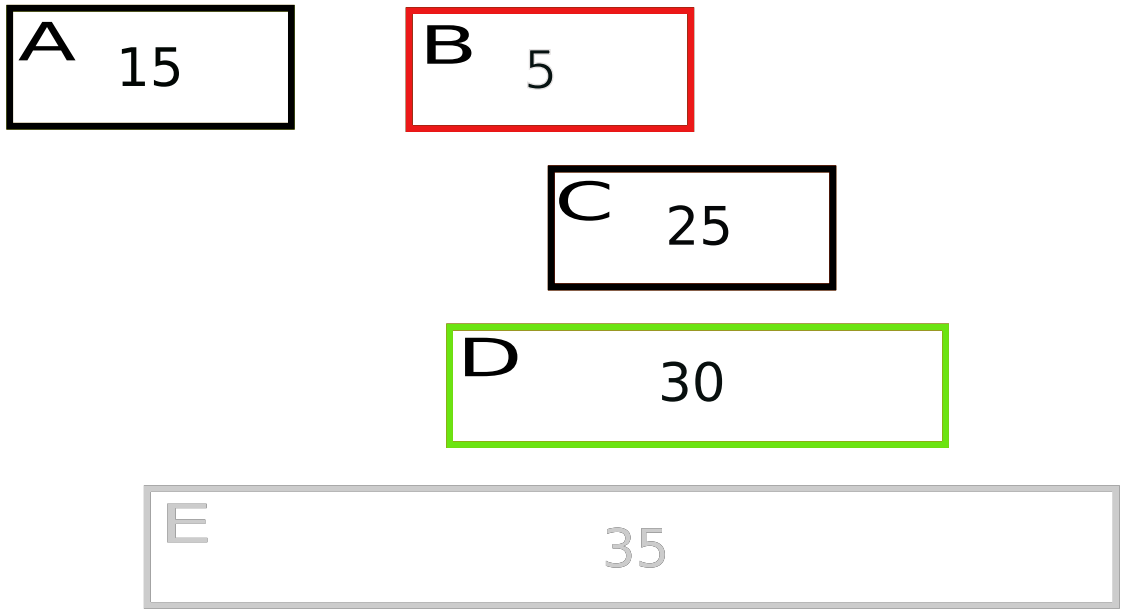
$w_A^{'} = 15$
$w_B^{'} = 5$
$w_C^{'} = 20$
$w_D^{'} = 5$

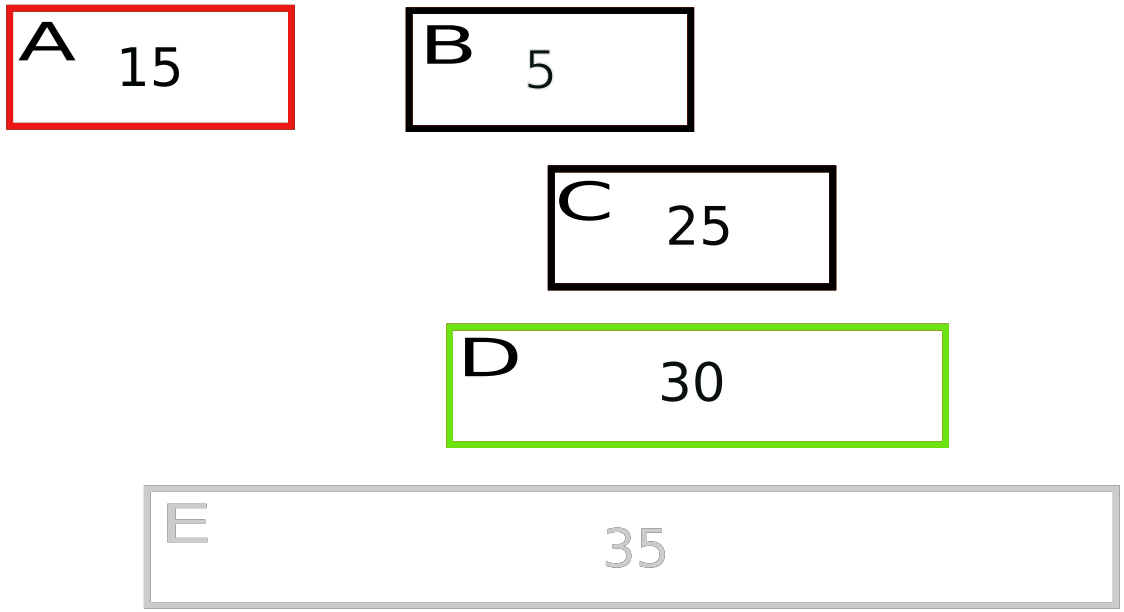


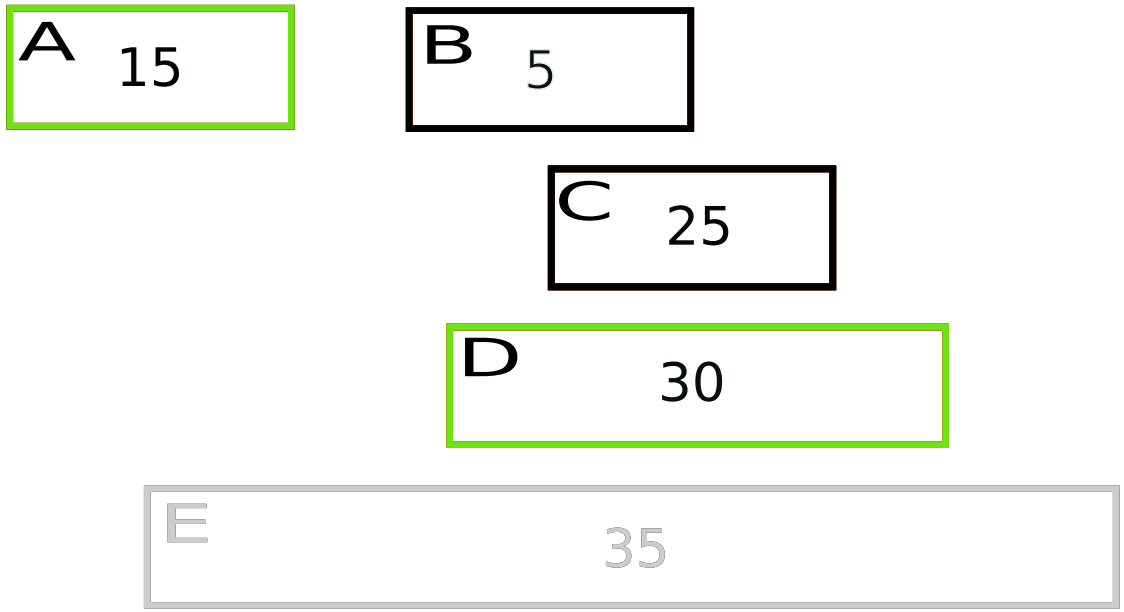
$w_A^{'} = 15$
$w_B^{'} = 5$
$w_C^{'} = 20$
$w_D^{'} = 5$


A
15
B
5
C
25
D
30
E
35


A
15
B
5
C
25
D
30
E
35


A
15
B
5
C
25
D
30
E
35


A
15
B
5
C
25
D
30
E
35


A
15
B
5
C
25
D
30
E
35

Is there an *optimal* local ratio algorithm for interval scheduling on more than one machine?

Borodin, Cashman and Magen [13] priority stack model for packing problems: A (fixed or adaptive) priority algorithm pushes (some) items onto a stack (possibly creating a non-feasible solution) and then (greedily) pops the stack so as to create a feasible solution.

No *fixed order* priority stack algorithm for weighted interval scheduling on $m \geq 2$ machines can be optimal [13]. Our current approximation lower bound for 2 machines is $\frac{6}{\sqrt{30}} \approx 1.077$ and as $m \rightarrow \infty$, our inapproximation bound approaches 1 (whereas the local ratio approximation ratio $(2 - \frac{1}{m})$ approaches 2 as $m \rightarrow \infty$). Open: an inapproximation for adaptive priority stack algorithms.

Generalizing this local ratio algorithm

Bar-Noy et al, and independently Berman and Das Gupta show that fixed ordering by finishing times can be used to approximate a number of more general interval scheduling problems, including the (weighted) JISP problem for which it achieves a 2-approximation. Viewing the one machine problems as graph MIS problems, Akcoglu et al [2] abstract the underlying graph property enabling these simple approximation algorithms (whereas MIS in general graphs is NP-hard to approximate within a factor $n^{1-\epsilon}$). Namely, they generalize the perfect elimination ordering of chordal graphs to a "k-sequentially independent" ordering of vertices $v_1, \ldots, v_n$ so that $N(v_i) \cap \{v_{i+1}, \ldots, v_n\}$ has at most k independent vertices. They show that using a k-sequentially independent ordering as a fixed (albeit not necessarily priority) ordering, the local ratio algorithm is a k-approximation algorithm for the MIS problem on such a "k-elimination" graph.

Yuli Ye and I have studied such k-elimination graphs showing that they include many common graph classes (e.g. planar graphs are 3-elimination graphs) for small k and that simple "greedy like" algorithms can efficiently approximate many different graph problems for this class generalizing many previous results. In particular, a local ratio algorithm $(k + 1 - \frac{1}{m})$-approximates the m-partite induced subgraph problem on k-elimination graphs. The priority stack model inapproximation suggests that it will not be possible to use this approach to optimally solve the m-partite induced subgraph problem for chordal graphs for which $k = 1$. More generally, we are considering sequential elimination graphs based on other local neighborhood properties. One can then consider the power of (non priority) local ratio algorithms for such graph classes when given a fixed elimination order.

More priority algorithm extensions

- Two pass and multiple pass algorithms

  An algorithm could make two (or more) passes over the algorithm, using the initial pass(es) to stream the input filtering out some input items and then using the final pass to make irrevocable decisions. Analyzing such 2 pass algorithms seems challenging but not impossible. (Fixed order stack algorithms are a special case.)

- The revocable priority model for packing problems is a (one pass) priority model for packing problems where only rejections are irrevocable. The model is the same as for the (irrevocable) priority model except now previously accepted items can be deleted. The algorithm always maintains a feasible solution. The revocable priority model has received little attention and it seems plausible that new algorithms can be developed in this framework.

Revocable priority algorithms are provably more powerful than priority.

Weighted Job Interval Scheduling Problem

The "$Greedy_\alpha$" of [10, 24] is a revocable priority algorithm for the $WJISP$ (and therefore interval scheduling) that achieves a constant approximation ratio. The parameter α determines how much better a new interval has to be in order to discard previously accepted intervals. Hence, the decision as to whether or not to accept the next interval is not strictly a "greedy decision".

Erlebach and Spieksma state that "these algorithms ...seem to be the simplest algorithms that achieve constant approximation ratios for $WJISP$". The are simpler in the sense that it seems easier to prove limitations.

Sort intervals so that $f_1 \leq f_2 \ldots \leq f_n$

$A := \emptyset$

For $i : 1..n$

    If I_i does not conflict with intervals in A

        then $A := A \cup \{I_i\}$

        else let $C_i \subseteq A$ be (the) minimum profit conflicting set;

            If $w(C_i) \leq \alpha \cdot w_i$ then $A := A - C_i + \{I_i\}$

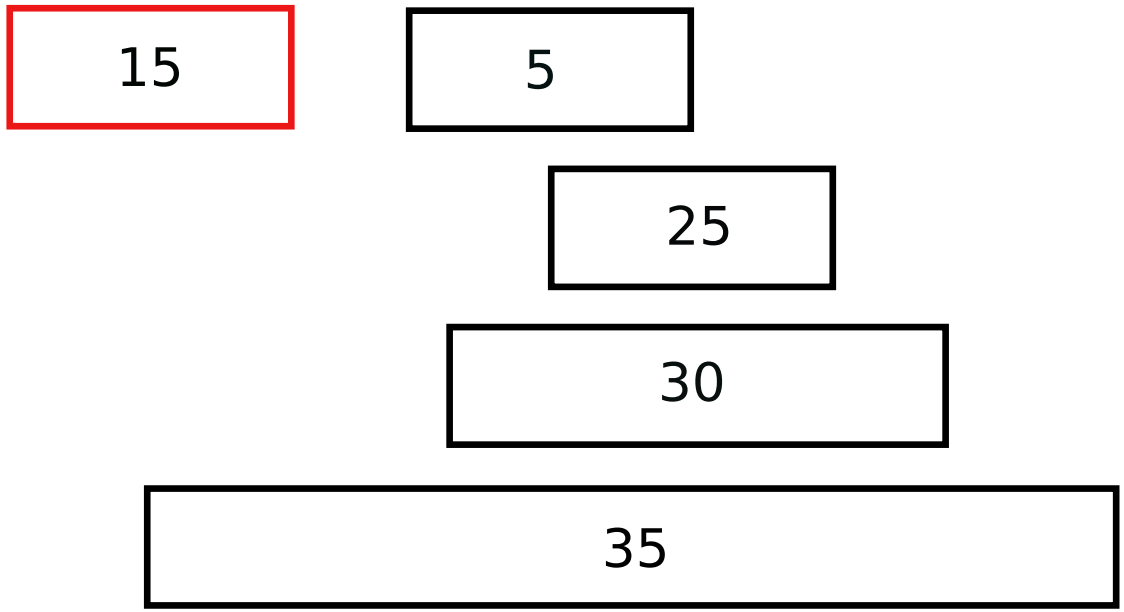
            End If

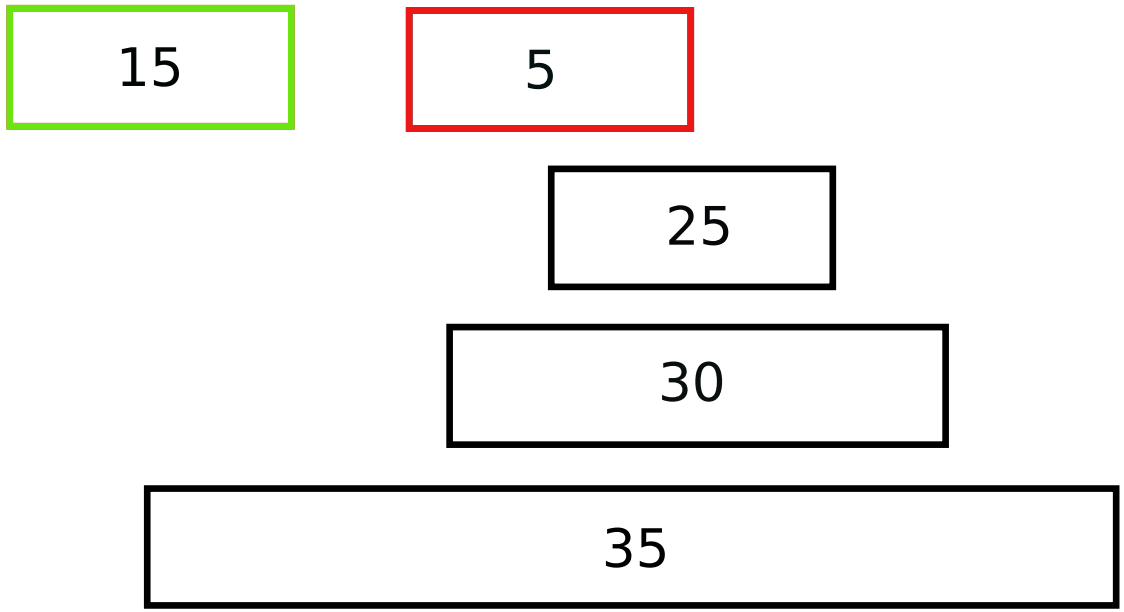
    End If

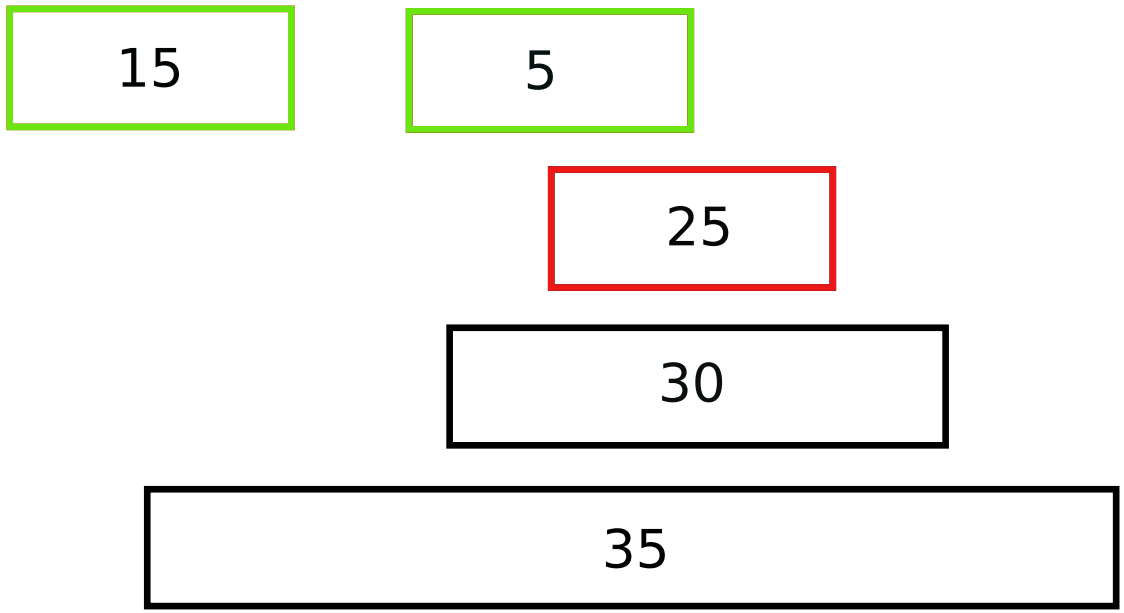
End For

Any $\alpha < 1$ yields an $O(1)$- approximation with $\alpha = 1/2$ providing a 4 (resp. 8)-approximation for interval scheduling (resp. $WJISP$).

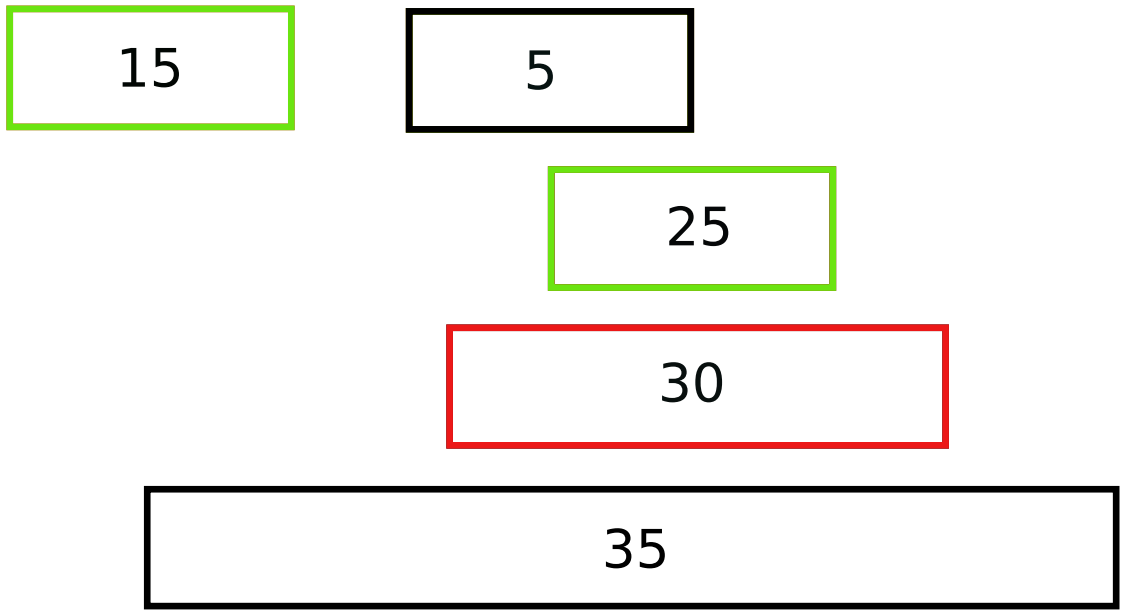
No revocable priority algorithm can achieve approximation ratio better than ≈ 1.17 for interval scheduling (Horn [32]) or better than $\frac{3}{2}$ for $JISP_3$ (Ye).


15
5
25
30
35

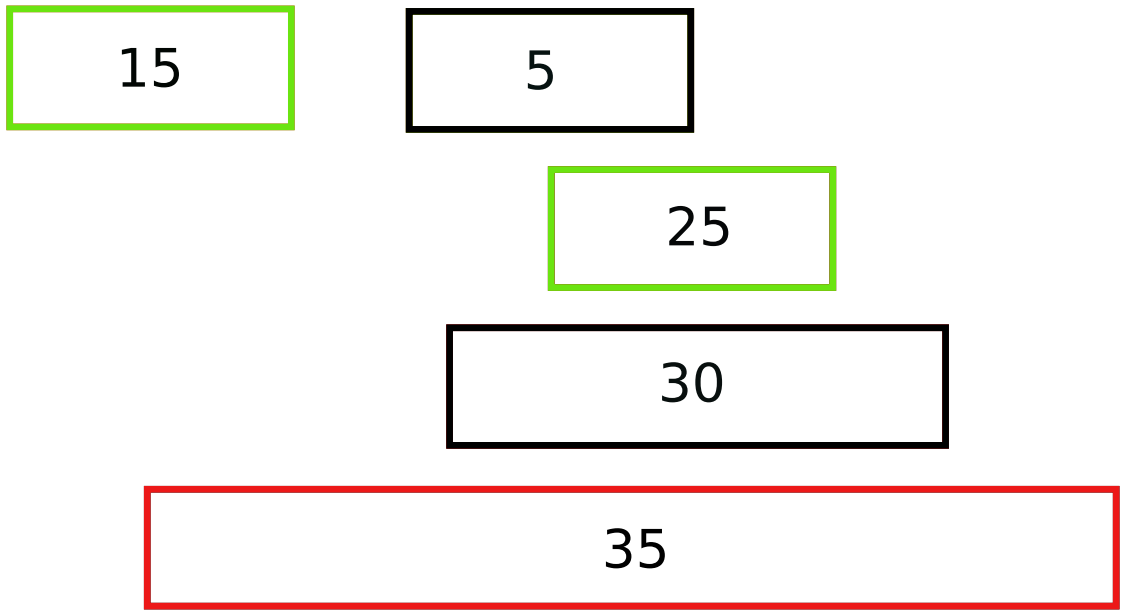

15
5
25
30
35



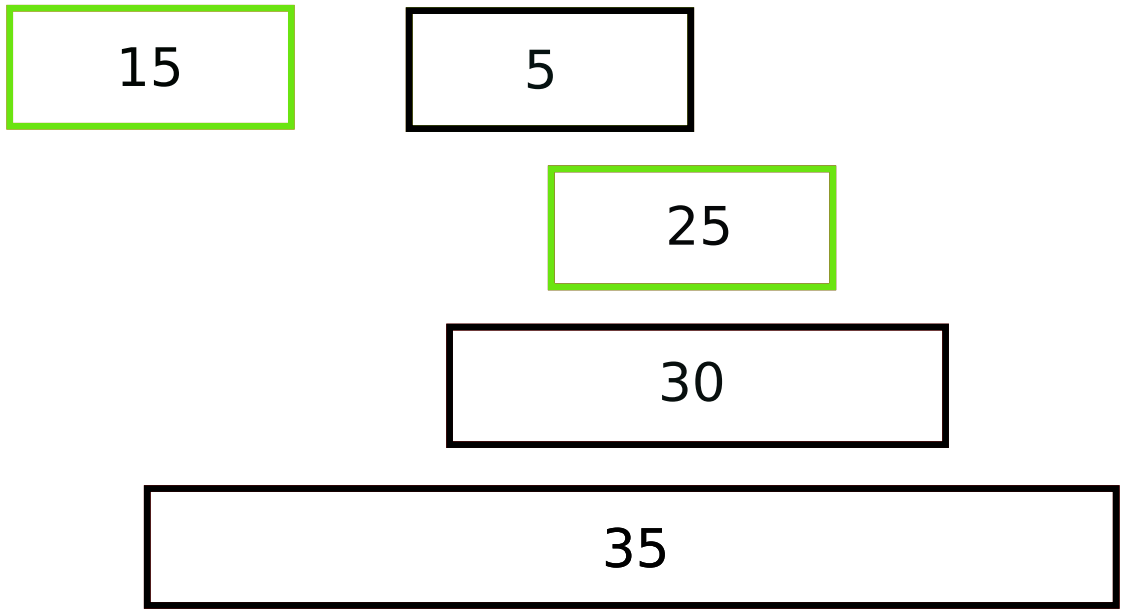
$5 \leq \frac{1}{2} \cdot 25$, so swap the second for the third interval.



$25 \not\leq \frac{1}{2} \cdot 30$, so current solution unchanged.



$15 + 25 \not\leq \frac{1}{2} \cdot 35$, so current solution unchanged.


15
5
25
30
35

Simple dynamic programming example.

Well known DP algorithm optimally solves the weighted interval scheduling problem for any fixed number of machines m. For $m = 1$, the DP is based on computing the array $OPT[i]$ = optimal profit obtainable using the first i intervals where again we assume $f_1 \leq f_2 \leq \ldots f_n$. $OPT[i]$ is computed using the recurrence:

$$OPT[i] = \max\{OPT[i-1], OPT[\pi(i)] + w_i\}$$

where $\pi(i) = \max\{j | f_j \leq s_i\}$. (Either the best schedule uses the i^{th} interval or it doesn't use it.)
Extending to $m \geq 2$, we use an m-dimensional array and the DP algorithm runs in time and space $\Theta(n^m)$.

Following Woeginger [48], simple DP intuitively means that the recursion is based on the number of items considered.

Must every DP for m machine weighted interval scheduling suffer this curse of dimensionality?

pBT: A Model for Simple DP and Backtracking

We [3] propose a model that captures some aspects of lookahead (in priority algorithms), parallel greedy algorithms, search methods (e.g. DPLL style) used in various SAT solvers, and Woeginger's [48] "simple" dynamic programming (DP) algorithms.

pBT model for simple backtracking and simple DP.

A pBT program is a leveled tree where:

- Nodes are labeled by a priority ordering rule determining an input item which will be accessed at this node.
- Based on the input item being considered (and the path leading to this node), the tree can branch so as to allow different irrevocable decisions (in previous example, accept or reject) about the accessed item; tree edges are labeled by the decision being made.
- Any path can be aborted (based on the items seen thus far).
- For an optimization problem the output is taken to be the value given by the best path in the pBT tree. For a search problem, some path creates an accepting solution if one exists.

Each path in a pBT algorithm is a priority algorithm. The ordering of input items can be fixed or adaptive. We define an *adaptive pBT algorithm* as one in which the ordering of the items is done adaptively but at each level the same input item is being considered. If the input item being considered depends on the path (and not just the items considered thus far) then the pBT algorithm is fully adaptive. We measure the complexity of a pBT program by its maximum width (i.e. a space measure) or its "depth first search size" (i.e. a time measure). Any NP problem can be solved optimally with exponential pBT trees. We consider the width or depth first search size required for optimality or for a (suitably approximate) solution.

Overview of pBT Results.

1. Interval scheduling
   - Curse of dimensionality width $\Omega(n^m)$ lower bound for optimal adaptive pBT for any fixed m.
   - The optimal DP can be implemented by a n^m width fixed ordering pBT.
   - A (weak) constant approximation lower bound (for proportional profit) for any bounded width fixed ordering pBT (even allowing revocable acceptances).
   - An adaptive width 2 pBT achieves a 2-approximation for one machine proportional profit in contrast to the priority (width 1) 3-approximation lower bound.

2. Subset-sum and Knapsack
   - $2^{\Omega(n)}$ width and depth first size lower bound for fully adaptive pBT using pBT preserving reduction. Also a direct proof for adaptive case allowing revocable acceptances.
   - Optimal DP algorithms and "FPTAS" algorithms are simple DP algorithms and can be realized in pBT model.
   - An $\Omega((\frac{1}{\epsilon})^{1/3.17})$ width lower bound for any adaptive pBT that provides a $(1+\epsilon)$-approximation vs $O(\frac{1}{\epsilon^2})$ upper bound.
3. CNF-SAT (pBT as model for DPLL style algorithms)
   - $\forall\epsilon\exists\delta$: any fixed order pBT requires width $2^{\delta n}$ to obtain $\frac{22}{21}-\epsilon$ approximation for MAX2SAT.
   - $2^{n/6}$ width lower bound for fixed order pBT for 2SAT.
   - $O(n)$ width adaptive pBT for 2SAT.
   - 3SAT requires $2^{\Omega(n)}$ depth first size (and width) in the fully adaptive model.

A more general framework for dynamic programming.

The pBT model does not capture optimal DP algorithms such as:

- Bellman-Ford algorithm for shortest paths (or longest path in DAG)
- "Non-serial" DP algorithms for binary search trees, the matrix chain problem, and RNA folding.

Recently, Buresh-Oppenheim, Davis, and Impagliazzo extend the tree pBT model to pBP DAG model which can capture the Bellman-Ford algorithm and is still amenable to analysis. Further progress recently by Jeff Edmonds.

Some preliminary interesting pBP results

- The fixed and adaptive priority pBP models can be simulated by (respectively) fixed and adaptive pBT algorithms.
- The Bellman-Ford algorithm can be realized by a $O(n^3)$ size strongly adaptive pBP using edges as input items.
- Some evidence that with edge input items, any pBT that solves the shortest path problem will require exponential width although there is a $O(n^2)$ width pBT when vertices are the input items.
- When edges are the inputs, any pBP algorithm for maximum matching in an unweighted bipartite graph will require exponential width providing some evidence that DP cannot efficiently solve bipartite graph matching.

Reflections and Many Open Questions

- How generally can one apply "information theoretic arguments" (as in the priority, pBT, pBP and stack models) to show limitations; specifically, to what extent can we understand the power of DP as defined by abstract recursive equations. Beware the Turing tarpit!
- Extensions of the priority framework. As stated before, 2-pass and multi-pass priority algorithms appear to be very hard (but not impossible) to analyze. Non IIA orderings.
- Randomized (priority, pBT, stack, etc.) algorithms. Angelopoulos's [4] extends a 4/3 facility location lower bound to randomized priority algorithms. Also the pBT lower bound for 3CNF implies some form of randomized lower bound.
- Viewing the pBT model as a backtracking model, how much information can be shared between paths in the pBT tree?

- Need to improve analysis so as to derive width/size vs approximation results for pBT (and pBP) approximation algorithms.
- Many (really almost all) specific problems (e.g. Max2SAT, Max Cut) where best priority approximation ratio is not understood. Proving results for (unweighted) graph problems when the input items are vertices seems difficult.
- Does randomization and/or non-greedy decisions help for (say) makespan on identical and non-identical machine models (as it does for online algorithms)?

Beyond priority based models

- The formalization of local search algorithms and related variants.
- Formalizing algorithms for problems that are not search and optimization problems. In particular, dynamic programming is used in many contexts. How to differentiate dynamic programming from divide and conquer.
- Appropriate versions of reductions and methods for comparing and composing simple algorithms. Can we have a theory of simple algorithms?

It is rather amazing how little we know about the power and limitations of conceptually simple algorithms.

Reprise: Holy Grail?

Will any of these frameworks lead (directly or indirectly) to new algorithms or at least to a better analysis or simplification of known algorithms? The competitive analysis of online algorithms has led to new algorithms and new insights; e.g. Bartal's hierarchically separated tree spaces. Or is it a sufficient justification to be able to convincingly rule out certain approaches? Worth considering why the Turing model is such an invaluable model.

Fast matrix multiplication, linear time string matching, linear time median are a few examples of algorithms that had origins in complexity theory. Complexity theory dramatically changed the fields of cryptography and optimization. Can a more systematic study of (simple) algorithmic paradigms have significant positive benefits?

References

[1] Dimitris Achlioptas and Gregory B. Sorkin. Optimal myopic algorithms for random 3-SAT. In *IEEE Symposium on Foundations of Computer Science*, pages 590–600, 2000.

[2] Karhan Akcoglu, James Aspnes, Bhaskar DasGupta, and Ming-Yang Kao. Opportunity cost algorithms for combinatorial auctions. *CoRR*, cs.CE/0010031, 2000.

[3] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a model for backtracking and dynamic programming. In *20th Annual IEEE Conference on Computational Complexity CCC05*, 2005.

[4] S. Angelopoulos. Randomized priority algorithms. In *Proceedings of the 1st Workshop on Approximation and Online Algorithms (WAOA)*, 2003.

[5] S. Angelopoulos and A. Borodin. On the power of priority algorithms for facility location and set cover. In *Proceedings of the 5th*

International Workshop on Approximation Algorithms for Combinatorial Optimization, volume 2462 of *Lecture Notes in Computer Science*, pages 26–39. Springer-Verlag, 2002.

[6] E. M. Arkin and E. L. Silverberg. Scheduling jobs with fixed start and end times. *Disc. Appl. Math*, 18:1–8, 1987.

[7] S. Arora, B. Bollobás, and L. Lovász. Proving integrality gaps without knowing the linear program. In *Proceedings of the 43rd Annual IEEE Conference on Foundations of Computer Science*, pages 313–322, 2002.

[8] S. Arora, B. Bollobás, L. Lovász, and I. Tourlakis. Proving integrality gaps without knowing the linear program. *Theory of Computing*, 2(2):19–51, 2006.

[9] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *JACM*, 48(5):1069–1090, 2001.

[10] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines under real-time scheduling. *SICOMP*, 31(2):331–352, 2001.

[11] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

[12] P. Berman and B. Das Gupta. Improvements in throughput maximization for real-time scheduling. In *STOC: ACM Symposium on Theory of Computing*, 2000.

[13] A. Borodin, D. Cashman, and A. Magen. Formalizing the local-ratio method. In *32nd International Colloquium on Automata, Languages and Programming ICALP05*, 2005.

[14] A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.

[15] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinato-*

rial Optimization (APPROX), Lecture Notes in Computer Science, Vol.1913, pages 84–95, 2000.

[16] V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[17] V. Chvátal. Hard knapsack problems. *Operations Research*, 28(6):1402–1441, 1980.

[18] K. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16:23–25, 1983.

[19] S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[20] A. Dechter and R. Dechter. On the greedy solution of ordering problems. *ORSA Journal on Computing*, 1(3):181–189, 1989.

[21] D.Gale. Optimal assignments inan ordered set. *PJournal of Combinatorial Theory*, 4:176–180, 1968.

[22] J. Edmonds. Submodular functions, matroids, and certain polyhedra, 1970.

[23] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[24] T. Erlebach and F.C.R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *Technical Report 152, Computer Engineering and Networks Laboratory, ETH*, October 2002.

[25] U. Faigle and W.M. Nawijn. Greedy k-decomposition of interval orders. In *Proceedings of the Second Twente Workshop in Graphs and Combinatorial Optimization*, pages 53–56, University of Twente, 1991.

[26] R. Gonen and D. Lehmann. Optimal solutions for multi-unit combinatorial auctions: branch anbd bound heuristics. In *Proc. of 2nd ACM Conf. on Electronic Commerce*, pages 13–20, 2000.

[27] R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[28] P. Helman. A common schema for dynamic programming and branch and bound algorithms. *Journal of the Association of Computing Machinery*, 36(1):97–128, 1989.

[29] P. Helman and A. Rosenthal. A comprehensive model of dynamic programming. *SIAM Journal on Algebraic and Discrete Methods*, 6:319–324, 1985.

[30] A. Hoffman. On simple linear programming problems. In *Proceedings of 7th Symposium in Pure Mathematics*, pages 317–327. American Math Society, 1963.

[31] A. Hoffman. On greedy algorithms that succeed, 1985.

[32] S.L. Horn. One-pass algorithms with revocable acceptances for job interval selection. *MSc Thesis, University of Toronto*, 2004.

[33] T.A. Jenkyns. The efficiency of the 'greedy' algorithm. In *Proc. of* 46^{th} *Foundations on Combinatorics*, pages 341–350, 1976.

[34] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[35] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1999.

[36] B. Korte and D. Hausmann. An analysis of the gredy heuristic for indepedence systems. *Annals of Discrete Math*, 2:65–74, 1978.

[37] B. Korte and L. Lovász. Mathematical structures underlying greedy algorithms. *Lecture Notes in Computer Science*, 177:205–209, 1981.

[38] B. Korte and L. Lovász. Greedoids and linear objective functions. *SIAM Journal Algebraic and Discrete Methods*, 5:229–238, 1984.

[39] Guy Kortsarz and David Peleg. Generating sparse 2-spanners. *J. Algorithms*, 17(2):222–236, 1994.

[40] U. Vazirani L. Orecchia, L. Schulman and N. Vishnoi. On partitioning graphs via single commodity flows. In *To appear in Proceedings of ACM Symposium on Theory of Computing (STOC)*, 2008.

[41] D. Lehmann, L.L. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49:577–602, 2002.

[42] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[43] B. Lucier and A. Borodin. Greedy mechanism design for combinatorial auctions: Truthfulness and the price of anarchy.

[44] M. Mahdian, J. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 229–242, 2002.

[45] G.L. Nemhauser M.L. Fisher and L.A. Wosley. An analysis of approximations for maximizing submodular set functions - ii. *Mathematical Programming Studies*, 8:73–87, 1978.

[46] R. Rado. A note on independence functios. *Proceedings of the London Mathematical Society*, 7:300–320, 1957.

[47] Oded Regev. Priority algorithms for makespan minimization in the subset model. *Information Processing Letters*, 84(3):153–157, Septmeber 2002.

[48] G. Woeginger. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing*, 12:57–75, 2000.

[49] Y. Ye and A. Borodin. Priority algorithms for the subset-sum problem. In *Proceedings of the 13th Annual Computing and Combinatorics Conference (COCOON), Lecture Notes in Computer Science, Vol.4598*, pages 504–514, 2007.