Why Simple Algorithms Can Be Complex: Toward a Systematic Study of Algorithms?

Allan Borodin

Department of Computer Science, University of Toronto

CRM-Fields-PIMS Lecture

presented (with my great appreciation) at

The Fields Institute, April 28, 2008

Based on years of pestering colleagues and papers with

Michael Alekhnovich [2]

Spyros Angelopoulos [5]

Josh Buresh-Oppenheim [2]

David Cashman [13]

Stephanie Horn [30]

Russell Impagliazzo [19]

Avner Magen [2], [13]

Morten Nielson [14]

Toniann Pitassi [2]

Charles Rackoff [14]

Yuli Ye [46]

A Theory for All Algorithms

In the context of discrete computation there is a well understood theory for the concept of an algorithm.
Namely, the Church-Turing computability definition is well accepted and we agree on what is and what is not computable.
For computation over the reals see, for example, Braverman [15] for a discussion of the 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.

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

Proving **upper** bounds on computation time

This is done by presenting algorithms

Rich, well-developed methodology:

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

Taught in every undergraduate CS curriculum.

Proving **lower** bounds on computation time

Methods:

Diagonalization

Reduction (reduce hard problem to your problem)

Counting (for Boolean circuit size)

Good success for very hard problems (super exponential complexity)

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 computation” with polynomial time computation by a Turing machine or appropriate RAM (or maybe quantum computers if that is your computational orientation) and quasi-linear time $O(n \log^k n)$ with “very efficient”.

Complexity theory has been very successful in many regards (e.g. (ZK)IP and PCP proofs, unique games hardness, derandomization, natural proofs and algebrization barriers, applications to cryptography, pseudo-randomness, coding theory, ...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 has also been very active with (for example) the development of SDP approximations, and new areas such as algorithmic game theory, mechanism design, and social networks.

A different perspective on the rich, well developed algorithmic methodology.

New modest title: An attempt at a more systematic study of some *"simple"*, *"combinatorial"* algorithms for combinatorial search and optimization problems.

And even more accurate, focus is on such problems where a feasible solution is (naturally) represented by decisions about input items. Furthermore the perspective (in terms of stated results) is that of worst case analysis. However, the definitions of the algorithmic models and their potential applicability do not depend on the worst case perspective. Furthermore, algorithmic concepts such as dynamic programming clearly have application beyond search and optimization problems.

Motivation

Conceptually simple algorithms are usually the first thing one tries when faced with (say) an optimization problem and sometimes these simple algorithms provide good (or even optimal) results or at least benchmarks for more sophisticated approaches.

For years, I have taught a standard course in the Design and Analysis of Algorithms, arguably the main undergraduate *theory course* in most CS departments. This is a theory course, where the basic objects of study, algorithmic techniques, are generally not defined! Yes we do prove theorems about particular algorithms for particular problems. Many courses and texts (Brassard and Bratley, Kleinberg and Tardos, Cormen et al, DasGupta et al) on this subject organize the material in terms of these undefined (albeit intuitively understood) conceptually simple algorithmic paradigms (or meta-algorithms).

Typical paradigms taught in an algorithms course

- The big three: greedy, divide and conquer, dynamic programming.
- Other common paradigms: graph searching, network flows, local search.
- Somewhat more advanced topics: LPs, IP/LP rounding, primal dual algorithms, backtracking, branch and bound, algebraic methods.
- More advanced topics: semi-definite programming, metric embeddings, multiplicative weights update.
- And orthogonal to all this: reductions and randomization.

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

- Is Dijkstra's shortest path algorithm a greedy algorithm or is it dynamic programming?
- Do we need dynamic programming for shortest paths when edge costs can be negative? Or is there a greedy algorithm?
- Can we efficiently compute maximum (bipartite) matching by dynamic programming?

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 speaking for many: Who cares what kind of an algorithm it is?

Bellman [12] 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 [a] 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 laboriuous trial and error, will yield suitable formulations of involved processes.

[a] I take this to mean that one cannot axiomatize or completely formalize an intuitive concept.

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

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

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

Graham [25] (list scheduling algorithms 1966), Chvátal [17] (branch and bound 1980), Helman and Rosenthal, Helman [27, 26] (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 [39] (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 [21]. Hoffman [29] 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. as used in inventory theory) such algorithms were called *myopic*. For my purpose, I think myopic is more general and suggestive than greedy but greedy algorithms (les algorithmes gloutons) is the popular terminology.

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

Beautiful development starting in the 1950's with the work of Rado [43], Gale [20] and Edmonds [21, 22], (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 maximal 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 which sorts edges by the non-decreasing weight.

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 [28] on the transportation problem, a necessary and sufficent condition (the existence of a Monge sequence) determines when the transportation problem can be solved greedily (using the fixed order of the Monge sequence). Significant body of research on the exploitation of Monge sequences and arrays to greedy and dynamic programming algorithms.

Maximizing a submodular function

Canonical greedy for the maximization of modular and submodular functions subject to a matroid (and more generally k-indepedence) constraint following Edmonds and starting with Jenkyns [31], Korte and Hausmann [36], and Fisher, Nemhauser and Wosley [42]. Recent interest due to application to combinatorial auctions.

```
$S := \emptyset$
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-indpendence set system $(E, \mathcal{I})$.

Greedy is an adjective

However, the term greedy algorithm has taken on a more general meaning and this is my starting point for thinking about a more systematic study of algorithms. Not counting brute force enumeration, greedy approximation algorithms are perhaps the simplest algorithms to design; yet it is not easy to define nor understand the ultimate power and limitations of such algorithms. The design and analysis of a greedy algorithm can be quite sophisticated. Surprisingly, relatively little attention so far as to the scope and power of what we call greedy algorithms.

Ignoring Socratic reservation, I will present a precise model, *priority algorithms* (Borodin, Nielsen, Rackoff [14]), that I believe captures most (but not all) known greedy algorithms. Extending the priority algorithm formulation, we also obtain models for some basic types of dynamic programming, backtracking, and primal dual algorithms.

The Holy Grail for an algorithmic model.

- Model should capture all (almost all, most, many, or at least some) known specific algorithms within this class.
- Model should be intuitive and reasonably "appealing".
- Model should be amenable to analysis. In particular, we should be able to provably establish (interesting) limitations on the power of an algorithmic model.
- The Holy Grail: Model should lead to new insights! For most people such models are perhaps only valuable if they lead to better algorithms! Complexity theory religiously advocates that studying limitations can lead to new algorithms. In this regard we need some more observed miracles. More ambitiously, one might even pray that a precise framework might permit some semi-automation for design and analysis.

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 $\{F_n : (U \times D)^n \rightarrow \Re \cup \{+\infty, -\infty\} | n \geq 1\}$; $I \subset U, |I| = n$ 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 (as in canonical greedy) 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. We equate priority algorithms with myopic or "greedy-like" and greedy as a special case.

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).

   What orderings are allowed? We allow any "locally defineable 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, the restriction to IIA orderings seems quite permissive. For example, we can use any ordering induced by sorting according to the values of *any* function f : {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 (like some previous restricted models) does not have any explicit complexity constraints. Hence one can establish limits on optimality and approximation bounds independent of the $\mathcal{P} \neq \mathcal{NP}$ conjecture. But even though we avoid complexity constraints, greedy and greedy-like (i.e. priority) algorithms tend to be very efficient. The same approach is used (for example) in the competitive anaylsis of online algorithms which can be viewed as a special case of priority algorithms where "nature" (or an adversary) is determining the ordering of the input items. Unlike online algorithms and some of the other previous studies of restricted models, the model definition is not dictated but rather has to be justified.

A few examples of algorithms that fit the model

- Graham's [25] online and LPT greedy approximation algorithms for makespan on identical machines. (The beginning of the study of worst case approximation algorithms.)
- The optimal greedy algorithm for ISP (interval selection) and the greedy 2-approximation algorithm for $JISP$ (job interval selection) and unweighted maximum throughput problems.
- The greedy H_n-approximation set cover algorithm [32, 16, 40].
- The myopic (random SAT threshold) search algorithms [1].
- Various 2-approximation algorithms for vertex cover.
- Huffman optimal prefix coding (with tree nodes as items).
- Kruskal's and Prim's optimal MST algorithms for MST.
- The current best polynomial time approximation (1.52) for the uncapacitated metric facility location problem [41].

Even a fixed order algorithm can be a challenge

This shouldn't be a surprise as we know that even online algorithms (where the algorithm has no control over the order in which items are considered) can be subtle.

Example of a non obvious ordering

From flow shop scheduling [33]:
Derive a greedy algorithm for the problem of scheduling (the order of contestants) in a swimming-bicycling biathlon held in a one lane swimming pool and a one lane velodrome. Given the expected swimming-biking times (s_i, b_i) for each contestant, the goal is to minimize the expected latest completion time.

Answer to this flow shop scheduling problem:

Start with those that swim faster than they bike in increasing order of swim times, and then order the people who bike faster than they swim in decreasing order of bike times. [33]

The “natural” 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 $(u, v) \in E$
        $dc(u) := dc(u) - 1$
        $wc(u) = wc(u) - \frac{w(v)}{dc(v}$
    End For
    $S := S \cup \{v\}$; V:= V - {v} - {u: dc(u) = 0}
End While
```

Complicating a simple "solved" problem (for illustration).

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, a fixed order priority algorithm.

Does the weighted problem have such a simple solution? Note: feasible subsets are not a matroid, greedoid, or k-independence system.

Greedy for unweighted m machine interval scheduling. [24]

```
Sort $\mathcal{I} = \{I_1, \ldots, I_n\}$ so that $f_1 \le f_2 \le \ldots f_n$
$S := \emptyset$
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 $t_j \le s_i$ 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 most obvious greedy algorithm?

The ordering in the previous greedy algorithm is reasonably natural but perhaps not the most obvious choice. One might think that ordering so that $|f_1 - s_1| \leq |f_2 - s_2| \leq \ldots |f_n - s_n|$ is more natural. Moreover, does it matter on which machine we schedule a job?

What fixed (or adaptive) orderings might work for *weighted* (one machine) interval scheduling?

- 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.

Two NP-hard and not so well understood extensions.

- The (W)JISP Job Interval Scheduling Problem where an input item is an interval $I_i = (s_i, f_i, w_i, j_i)$ where now j_i is the job to which I_i belongs. As in interval scheduling, intervals cannot intersect and additionally at most one interval per job can be scheduled. Even in the unweighted case, the problem is MAX-SNP hard (and hence unlikely to have a PTAS) and this hardness holds for the case of 2 intervals per job.

- The (W)TCSP Time Constrained Scheduling Problem where an input item is a job that has a release time, a deadline, and a processing time. Even in the unweighted case the problem remains strongly NP-hard, and weakly NP-hard when there is no release time.

The adaptive greedy algorithm for the unweighted one-machine unit profit *TCSP* (and JISP). [10]

```
$S := \emptyset$
$t := 0$    %$t$ is the completion time of the last scheduled job
While $\mathcal{I} \neq \emptyset$
    Let $i = argmin_j\{\min(t, r_j) + p_j\}$ %earliest completion time
    delete $I_i$ from $\mathcal{I}$
    If $t + p_i \leq d_i$ then
        schedule $I_i$ to start at time $t$
End While
```

Perhaps surprisingly, by applying the one machine algorithm to each machine on the remaining unscheduled intervals, [10] shows that the approximation ratio $\frac{(1+1/m)^m}{(1+1/m)^m-1}$ for the resulting priority algorithm *improves* with the number of machines.

Priority algorithms 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 weighted interval scheduling problem in time (and space) $O(n^m)$ and there is also a time $O(n^2 \log n)$ algorithm [6] (based on min cost-max flow) that optimally solves the problem. 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?

Did we correctly capture all greedy algorithms?

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).
- We could maintain some small number of partial solutions that we continue to augment. This falls within the pBT model [2] which we will soon discuss. In particular, any “simple DP” will require time and space $\Omega(n^m)$ for any fixed number of machines m [2].

- 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] is a 2 pass algorithm which (using a fixed ordering) streams the input items onto a stack and then pops the stack so as to create a feasible solution. This algorithm solves the m machine interval scheduling with approximation ratio $2-(1/m)$ and hence optimally for one machine.

  Such stack algorithms are formalized and studied in [13]

More general two pass algorithms

We could allow an algorithm to make two passes over the algorithm, using the first pass to stream the input filtering out some input items and then using the second pass to make irrevocable decisions.

Analyzing such 2 pass algorithms seems challenging but not impossible.

The revocable priority model for packing problems

There is a more general (one pass) priority model for packing problems where only rejections are irrevocable. The model is exactly the same as for the (irrevocable) priority model except now previously accepted items can (if desired) be deleted. The algorithm must maintain a feasible solution at all times. The (partial) power to possibly reverse a bad decision clearly adds a great deal of flexibility to the model.

Claim: The revocable priority model has not received much attention and it seems plausible that new algorithms can be developed in this framework. Moreover, the revocable model is more amenable to analysis than (say) the 2 pass model.

An example where revocable priority algorithms are provably more powerful than priority.

$WJISP$

The "$Greedy_\alpha$" of [10, 23] 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 "locally optimal decision".

Erlebach and Spieksma state that "these algorithms ...seem to be the simplest algorithms that achieve constant approximation ratios for $WJISP$".

```
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 [30]) or better $\frac{3}{2}$ for $JISP_3$. (Ye).

Simple Dynamic Programming and Backtracking as Priority Extensions

What is dynamic programming? From a blog ... ‘there are about as many definitions of “dynamic programming” as there are computer scientists’.
DasGupta, Papadimitriou and U. Vazirani text:

“... identifying *a collection of subproblems* and tackling them one by one, smallest first, using the answers to small problems to help figure out larger ones...”. “DP and LP, the worlds two most general algorithmic techniques”.

Definitions for DP appear in Helman [26] following Karp and Held [34] and Helman and Rosenthal [27]. “These papers did not have a lot of traction”.
Following Woeginger [45], simple DP intuitively means that the recursion is based on the number of items considered.

Simple dynamic programming example.

A well known DP algorithm solves the weighted interval scheduling problem optimally 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.) The same idea can be used for any number $m \geq 2$ of machines but now we (seem to) need an m-dimensional array and this DP algorithm will run in time $\Theta(n^m)$. Note: The algorithm is maintaining many partial solutions.

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

pBT: A Model for Simple DP and Backtracking

(Alekhnovich, Borodin, Buresh-Oppenheim, Impagliazzo, Magen, Pitassi [2])

Using the priority framework as a starting point, we [2] propose a model that allows us to also capture some aspects of lookahead (in priority algorithms), parallel greedy algorithms, search methods (e.g. DPLL style) used in various SAT solvers, and Woeginger's [45] class of "simple" dynamic programming (DP) algorithms. In particular, the DP algorithm for weighted interval scheduling is a simple DP (where the partial solutions are being constucted one item at a time). In the same way, the optimal pseudo polynomial time and FPTAS DP algorithms for the knapsack problem are simple DPs as is the edit distance DP algorithm.

The pBT tree model for backtracking and simple DP.

We continue to restrict attention to NP search problems where a certificate is desired (e.g. the set of true literals that satisfy a CNF formula) or to optimization problems whose objective function is determined by the feasible solutions. We consider a pBT program to be a leveled tree:

- 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 our two examples, accept or reject) about the item accessed at this node; each tree edge is 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 an NP-search problem some path must verify 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 we say that the pBT algorithm is fully adaptive. We measure the complexity of a pBT program by its maximum width (i.e. a space meaure) and (for the fully adaptive model) also 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 Results on pBT Width and Depth First Size.

1. Interval scheduling
   - Width lower bound of $\Omega(n^m)$ for optimal adaptive pBT for any fixed m. Thus the curse of dimensionality is necessary in this framework.
   - 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. $O(\frac{1}{\epsilon^2})$ width 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 2CNF-SAT.
   - 3SAT requires $2^{\Omega(n)}$ depth first size (and width) in the fully adaptive model.

Some comments about the 3SAT lower bound

The lower bound for 3SAT is based on a very restricted class of 3SAT formulas, namely the formulas induced by the equations $Ax = b$ $(mod2)$ where A is a non singular $n \times n$ {0,1} matrix representing a suitable bipartite expander graph, $x = (x_1, \ldots x_n)$ is the vector of {0,1} propositional variables and b is a {0,1} vector. Furthermore, there are at most some constant k 1's in any column (corresponding to variables) and at most 3 1's in any row (corresponding to equations). Such examples have been introduced in proof complexity [3]. Algebraically, it is easy to solve such a system. But pBT algorithms require $2^{\Omega(n)}$ width and depth first size, even when A is known in advance. That is, only the b vector is an "input" to the problem. Each non zero setting of the b vector defines a propostional formula having a unique solution. But for a random instance, a random path on that instance has non-negligible probability of leading to an "indispensible partial solution".

A more general framework for dynamic programming.

The pBT model does not capture DP algorithms such as (say) the Bellman-Ford algorithm for shortest paths for graphs with no negative weight cycles (or longest path in a DAG), nor does it capture "non-serial" optimal DP algorithms (say) for constructing a binary search tree, for the matrix chain problem, and for the Nussinov and other DP based RNA folding algorithms.

Recent work by Buresh-Oppenheim, Davis, and Impagliazzo extends the tree pBT model to a DAG model (called pBP for priority branching program) which can capture the Bellman-Ford algorithm and is still amenable to analysis (e.g. exponential width lower bound for finding a maximum matching in a bipartite graph when the input items are edges). A further extension is suggested (but not studied) that can model the non-serial DP algorithms above.

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 convincing evidence that with edge input items, any BT that solves this version of the shortest path problem will require exponential width although there is a $O(n^2)$ width pBT when vertices are the input items. That is, there is convincing evidence that simple DP algorithms are restrictive.
- When edges are the inputs, any pBP algorithm for maximum matching in an unweighted bipartite graph will require exponential width providing evidence that DP cannot efficiently solve bipartite graph matching.

The Stack Model for Simple Primal Dual

The one stage "dual ascent primal dual scheme" (as defined for covering problems in Williamson [44]) can be interpreted as a priority algorithm under certain assumptions on the input items corresponding to the IP variables.

Two stage primal dual: The more general (but still quite basic) two stage primal dual uses a "reverse delete stage" following an initial "priority stage". Primal dual algorithms have been informally shown by Bar-Yehuda and Rawitz [11] to be equivalent to local ratio algorithms (for both covering and packing problems). Such a two stage algorithm can optimally solve one-machine interval scheduling [9] and can also provide a 2-approximation for WJISP.

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

Note: This becomes the optimal greedy algorithm for the $m = 1$ unweighted case.

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

Borodin, Cashman and Magen [13] 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. (For interval scheduling problems any fixed order revocable priority algorithm can be simulated by a stack algorithm.)

No *fixed order* stack algorithm for weighted interval scheduling on $m \geq 2$ machines can be optimal [13]. Our current approximation lower bound for 2 machines is 14/13 and as $m \rightarrow \infty$, our inapproximation bound approaches 1 (whereas the previous local ratio approximation ratio $(2 - \frac{1}{m})$ approaches 2 as $m \rightarrow \infty$).

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 boud 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. In particular, what is the exact tradeoff between width and approximation for the weighted interval selection problem, for the edit distance problem, etc.. Some tradeoff results for fixed order pBT but almost nothing for adaptive and strongly adaptive order.
- In general, proving results for (unweighted) graph problems when the input items are vertices seems difficult. Note that edge items are independent of each other while this is not the case for vertex items. But 3CNF proof gives some hope.
- Many specific problems where best priority approximation ratio is not understood. For example, makespan on identical and non-identical machine models. Does randomization and/or non-greedy decisions help here as it does for online algorithms? Max2SAT, Max Cut, etc. etc.

- The formalization of local search algorithms, etc.
- 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? Or perhaps 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. Claim: the competitive analysis of online algorithms has led to new algorithms and new insights; e.g. Bartal's hierarchically separated tree spaces.

Fast matrix multiplication, linear time string matching, linear time median are a few examples of algorithms that had origins in complexity theory. And, of course, complexity theory dramatically changed the field of cryptography. I am hopeful that a more systematic study of (simple) algorithmic paradigms will have some (perhaps unexpected) 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] 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.

[3] M. Alekhnovich, E. Hirsch, and D. Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. In *Automata, Languages and Programming: 31st International Colloquium, ICALP04*, 2004.

[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. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. In *4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX*, pages 24–35, 2001.

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

[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. Braverman. On the complexity of real functions. In *Proc. of 46^{th} Annual IEEE Conference of Foundastions of Computer Science (FOCS)*, pages 155–164, 2005.

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

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

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

[23] 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.

[24] 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.

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

[26] 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.

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

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

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

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

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

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

[33] S.M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logist. Quart.*, pages 61–68, 1954.

[34] R.M. Karp and M. Held. Finite state processes and dynamic programming. *SIAM J. Applied Mathematics*, 15:693–718, 1967.

[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] 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.

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

[41] 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.

[42] 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.

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

[44] D. P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming, Series B*, 91(3):447–478, 2002.

[45] 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.

[46] 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.