

# Evaluating Algorithmic Design Paradigms

Sashka Davis\*

University of California San Diego  
sdavis@cs.ucsd.edu

## Abstract

We present formal models for some of the known algorithmic design paradigms. Ideally, a formal model of a paradigm should: (1). Include all or nearly all of the known algorithms intuitively classified as examples of the technique; (2). Capture the intrinsic characteristics of the paradigm; (3). Be useful in lower bounding the limits of the technique through proofs of negative results for all algorithms in the formal model.

Building on [1] we give submodels for greedy algorithms and dynamic programming.

## 1 Introduction

In an algorithm design class, we are taught the basic algorithm paradigms such as divide-and-conquer, greedy algorithms, backtracking and dynamic programming. The paradigm is taught by an intuitive example together with a number of counter examples. Intuitive formulations, while easy to understand, do not allow us to answer the following natural questions. Suppose we have an optimization problem that we want to solve.

1. *What algorithmic design paradigm can help?* For example: Do we need dynamic programming to solve single source shortest path in graphs with negative weights, or can we solve the problem using some greedy strategy? Do we need flow algorithms to find a maximal matching in bipartite graphs or can we use a simpler and more efficient dynamic programming approach?

For example, greedy algorithms are typically simple and have efficient implementations. Hence, we would like to know whether we can find a greedy algorithm that solves a given optimization problem. Suppose that, after much thought and effort we were unable to find a greedy solution for our problem. Then we would like to be able to make a strong statement that no greedy algorithm exists that solves the problem exactly. To do so, we need a formal model of greedy algorithms. Once we certify that our problem is hard for all greedy algorithms in the model, then we try the next “more powerful” technique, e.g., a backtracking approach, or dynamic programming.

2. *Is a given algorithm optimal?* If we were lucky and

found an algorithm for our problem using some algorithmic paradigm, then the next natural question to ask is whether our solution is the most efficient solution within this class of algorithms. We could give a precise answer to such a question if we are able to prove lower bounds on resources for all algorithms that fit a given class (Note that the space of such algorithms is infinite). After formalizing the algorithm class, if we prove a lower bound matching the resources used by our algorithm then we have a certificate that our algorithm is optimal within the paradigm and cannot be improved without radical change. Suppose we exhaust all known algorithmic techniques and still have not been able to solve our problem, then we face the next question.

3. *How good an approximation scheme can we get?* If we cannot solve our problem exactly using the formalized approaches then we settle for approximation. The question here is what algorithmic technique delivers the best approximation. And how do we know that a given heuristic is really the best? We can certify a given approximation scheme as optimal, similarly as in the case of exact algorithms, if we prove a matching lower bound on the approximation ratio achievable by any other algorithm within a given class.

Another aspect of this line of work is a general *characterization of the power of the different approaches to optimization*. Intuitively greedy algorithms are efficient but weak, and whatever problems we can solve using greedy algorithms we can solve using say dynamic programming algorithms. But we would like to back this intuition with strong formal reasoning, which is impossible as long as our definitions are intuitive and our lower bounds hold for a single algorithm and not for an infinite class of algorithms.

To answer such questions, we need formal models that capture the intuitive notion of a given paradigm, along with a technique for proving lower bounds on the resources used by all algorithms that fit the given model. Not only can this formal approach be used to answer the questions above but it can also help us first to better understand the intrinsic structure or hardness of our problem, by knowing what algorithmic paradigms can or cannot solve it. We also learn the strength and weaknesses of the known algorithmic design techniques, which we measure by the kinds of problems they can solve exactly or approximately,

---

\*Research partially supported by NSF Award CCR-0515332, but views expressed are not endorsed by the NSF.

and the types of problems that are hard for the technique. Third, by understanding the problems and the paradigms better, we improve our skills as algorithm designers.

**Outline of the paper:** In Sect. 2 we briefly describe the current formal models and some of the known results; In Sect. 3 we demonstrate the approach by discussing the PRIORITY algorithms framework. We will present the general technique for proving lower bounds for all PRIORITY algorithms, and will use the technique to prove lower bound on the approximation ratio achieved by all such algorithms for the Maximum Independent Set problem.

## 2 Models of Algorithmic Design Paradigms

Three formal models of algorithmic paradigms have been developed so far. [1] (See also [2]) formalized the intuitive notion of greedy algorithms, [3] for backtracking and dynamic programming, and [4] presents a stronger model for dynamic programming algorithms.

**PRIORITY algorithms** - *a formal model of greedy-like algorithms.*

[1] built a formal model of greedy algorithms for scheduling problems, which they called PRIORITY algorithms. [2] generalized the PRIORITY models to any problem domain and instantiated it for many graph optimization problems, e.g., shortest paths, vertex cover, spanning trees, independent set, and minimum steiner trees. We will delay a definition of PRIORITY algorithms until Sect.3 but will illustrate the framework by an example. Consider the Vertex Cover problem. Within the framework of PRIORITY algorithms an instance of the problems is given by a set of vertices of the graph. Each vertex is described by its name and a list of the names of its neighbors. At each iteration a PRIORITY algorithm chooses an ordering of the remaining vertices of the graph. It then considers the first vertex in the order and commits to a decision to add the vertex to the cover or reject it. The decision whether to accept or reject a vertex is irrevocable and can only depend on the current and previously considered vertices but not on future unseen vertices. The algorithm halts when decisions for all vertices are made.

Note that PRIORITY algorithms resemble on-line algorithms. The difference between the two is how the ordering of the vertices of the graph is chosen. In the case of on-line algorithms the ordering is adversarial, while PRIORITY algorithms have the power to choose the ordering. The technique for analyzing PRIORITY algorithms in [1] is borrowed from competitive analysis of on-line algorithms. [2] generalized the lower bound technique of [1] by abstracting away the domain specific details and defined it as a combinatorial zero-

sum game between two players, an Adversary and a Solver. A strategy for the Adversary in the game establishes a lower bound on the resources needed by *any* PRIORITY algorithm, while existence of a PRIORITY algorithm can be used by the Solver player as a strategy in the game. PRIORITY algorithms were shown to be weaker than dynamic programming algorithms ([1, 2]), and their power has been characterized by proving lower bounds for a variety of problems. We would like to point out that PRIORITY algorithms are a very general model of greedy algorithms which captures the majority of the intuitively named greedy algorithms in the literature, such as: Kruskal's and Prim's algorithms for minimum spanning trees, Dijkstra's single source shortest path algorithm, the known greedy 2-approximation of weighted vertex cover problem, the greedy approximations for set cover problem, facility location, job and interval scheduling problems. However, there are some algorithms termed greedy that do not seem to fit the PRIORITY model, for example the best known approximation for maximum independent set in bounded degree graphs does not seem to fit the model.

**prioritized Branching Trees** - *a formal model for backtracking and dynamic programming algorithms.*

[3] defined models for backtracking and simple dynamic programming algorithms, called prioritized Branching Trees (pBT). A pBP algorithm like backtracking and dynamic programming algorithms, maintains multiple solutions to subproblems and each computation step extends an already existing solution. [3] characterized the power of this large class of algorithms by proving lower bounds on the size of the computation tree built by any pBT algorithm for a variety of hard problems, e.g. SAT and Knapsack. Although very powerful, the pBT model of [3] has a shortfall that a classical algorithm like Bellman-Ford algorithm (for shortest paths in graphs with negative weights but no negative weight cycles) couldn't be seen to fit the pBT model. This motivated us to design the next model, to capture more complex dynamic programming algorithms.

**prioritized Branching Programs** - *a stronger formal model of dynamic programming algorithms.*

Our current work, [4], extends the pBT model of [3] and defines a stronger model of dynamic programming algorithms, which we call prioritized Branching Programs (pBP). The pBP model simulates the pBT model but in addition, formalizes the notion of *memoization*, which we believe is essential to dynamic programming algorithms. Thus, more of the canonical dynamic programming algorithms fit the pBP model. Again, the lower bound technique is defined as a zero-sum game between an Adversary and a Solver. However pBP algorithms are powerful and the definition of the game is technically involved. We have shown a separation between flow algorithms and a subclass of pBP algorithms. Our current work aims to give precise characterization of

the pBP model by showing that pBP algorithms are exponentially more “efficient” than the pBT algorithms of [3], and evaluating the performance of such algorithms on hard problems.

### 3 PRIORITY Algorithms

While we do not have space in this article to give technical description of the more complex models, here we present the model for greedy algorithms and illustrate some of the proof techniques used to reason about this model.

What are the defining characteristics of greedy algorithms? Let us look at a classical greedy algorithm and take it apart. Consider Kruskal’s algorithm for building a minimum cost spanning tree. The algorithm **sorts** all the edges of the graph according to their weight. It **considers each edge in this order** once and adds it to its solution, as long as the current partial solution remains a forest.

Next we will discuss the vital characteristics of greedy algorithms modeled by the PRIORITY algorithms. We will refer to Kruskal’s algorithm as an example<sup>1</sup>. A PRIORITY algorithm:

1. **Views the instance as a set of “data items”** In the case of Kruskal’s algorithm, the data items are the edges. Each edge is encoded by the names of the two vertices and its weight.
2. **Views the output as a set of “choices” (decisions) to be made, one per “data item”.** For the MST problem, the decisions are to either *accept* the edge and add it to the spanning tree or to *reject* it.
3. **Defines a “criterion” for best choices, which orders data items.** We refer to this criterion as an ordering function. For Kruskal algorithm the ordering function is simply the weight of each edge. The algorithm orders all data items of the instance only once, prior to making any decisions<sup>2</sup>.
4. In the order defined by this criterion, makes and **commits itself to the choices for the data items.**
5. **Never reverses a choice once made** (decisions are irrevocable).
6. In making the choice for the current data item, **only considers the current and previous data items, not later data items.**

To give a PRIORITY model for a problem we need to define the type of a data item and specify the decision options. Here is a PRIORITY model for the Minimum Spanning Tree problem: The type of a data item (edge of the graph) is  $\Gamma = (u, v, w)$ , where  $u, v$  are names of vertices of the graph, and  $w \in \mathbb{R}$  is the weight. The set of decisions is  $\Sigma = \{\text{accepted, rejected}\}$ . A formal spec-

<sup>1</sup>We refer the reader to [1, 2] for a formal presentation of the PRIORITY model and its submodels.

<sup>2</sup>We need to point out that Dijkstra’s algorithm for single source shortest paths in graphs, Prim’s algorithm for MST, and many others fit the more powerful class of PRIORITY algorithms, called ADAPTIVE PRIORITY. See [1, 2] for details.

ification of PRIORITY algorithms with its submodels is lengthy and we instead will cast the Kruskal’s algorithm as a PRIORITY algorithm

Kruskal’s algorithm in the PRIORITY framework:

1. The ordering function  $\pi : \Gamma \rightarrow \mathbb{R}$ , is set to be  $\pi(u, v, w) = w$ .
2. Initialize an empty solution:  $MST \leftarrow \emptyset$ .
3. Order  $E(G)$  according to  $\pi$ .
4. **while** (there is an edge not yet considered):  
**repeat:**  
Let  $e = (u, v, w)$  be the next edge according to  $\pi$ .  
If  $MST \cup \{e\}$  is a forest then accept  $e$ , else reject.

Now that we have a model (more precisely an example of what a model looks like) we want to be able to prove lower bounds for all PRIORITY algorithms. The lower bound is defined in terms of a combinatorial zero sum game between a Solver and an Adversary<sup>3</sup>.

Given a combinatorial problem  $\Pi$  with objective function  $\mu$ . Let  $\Gamma, \Sigma$ , be a PRIORITY model for  $\Pi$ , where  $\Gamma$  be the type of a data item (the set of all valid data items) and  $\Sigma$  be the set of options for each data item. At a high level, the Adversary must obtain a family of instances of  $\Pi$  so that for any PRIORITY algorithm there is some instance for which the algorithm fails to produce an optimal solution.

Initially the Adversary picks a finite set of instances  $T$  and a sets  $\Gamma_1 \subseteq \Gamma$  to be the set of data items that make up instances in  $T$ . The algorithm initializes an empty partial instance and partial solution  $PI_1, PS_1$ , respectively. From there on the game proceeds in rounds. At the beginning of round  $t$  we have remaining set of data items  $\Gamma_t$ , and current partial solution and instance  $PS_t, PI_t$ , respectively. The Solver selects a data item  $\gamma_t \in \Gamma_t$  and commits to a decision  $\sigma_t \in \Sigma$  for it.  $PI_t, PS_t$  are updated accordingly  $PI_t \leftarrow PI_t \cup \{\gamma_t\}$  and  $PS_t \leftarrow PS_t \cup \{(\gamma_t, \sigma_t)\}$ .  $\gamma_t$  is removed from  $\Gamma_t$  but the Adversary observes the move of the Solver and can further restrict the remaining set of data items  $\Gamma_{t+1}$  for the next round. Either a new round begins or the Adversary could choose to end the game at the end of any round. Otherwise the game continues until  $\Gamma_t = \emptyset$ . Let  $PI, PS$  be the partial instance and solutions maintained by the Solver at the end of the game. The Solver wins the game if  $PI$  is not a valid instance of  $\Pi$ , otherwise the Adversary chooses a solution  $OPT$  for  $PI$  and the Solver is awarded the ratio  $\rho = \frac{\mu(PI, PS)}{\mu(PI, OPT)}$ .

**Lemma 1 ([2])** *There is a winning strategy for the Solver in the above game if and only if there is a PRIORITY algorithm that achieves an approximation ratio of  $\rho$  on every instance of  $\Pi$  in  $T$ .*

#### 3.1 Maximum Independent Set (MIS)

Consider the MIS problem. Our goal is to show that MIS is hard for PRIORITY algorithms. A node model

<sup>3</sup>Precise formal definition of the game can be found in [2], Sect. 2.

for graph problems defines each data item to be the name of a node, the list of the name of its neighbors.

**Theorem 2** No *PRIORITY* algorithm in the node model can achieve an approximation ratio better than  $\frac{3}{2}$  for the *MIS* problem, even for graphs of degree at most 3.

**Proof:** In the node model the set of data items are the vertices of the graph with their names and adjacency lists. Naturally, the set of decision options is  $\Sigma = \{\textit{accepted}, \textit{rejected}\}$ . The Adversary sets  $T$  to be the graphs shown in Figure 1, and all isomorphic copies of these graphs. Therefore  $\Gamma_1$  is all possible tuples of node name and adjacency lists of size 2, and 3. Both  $G_1$  and  $G_2$  have 6 vertices, with degrees 2 and 3, and cannot be distinguished by the Solver a priori. The Solver orders all possible data items. Based on her first choice and decision made, the Adversary plays the following strategy:

1. The first data item chosen by the Solver is a vertex of **degree 3**.

**(1.a)** If the Solver decides to accept it, then the Adversary presents an isomorphic copy of graph  $G_2$ , where the node chosen by the Solver is  $C$ . The possible solutions for the solver are  $\{B, C\}$  or  $\{C, E\}$ , while the Adversary selects  $S_{Adv} = \{A, D, E\}$ . The approximation ratio awarded is  $\rho = \frac{3}{2}$ .

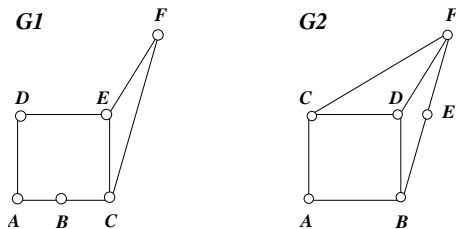


Figure 1: Nemesis graphs chosen by the Adversary

**(1.b)** If the Solver decides to reject the node, then the Adversary presents an isomorphic copy of  $G_2$ , such that the node chosen by the Solver is  $D$ . The possible solutions for the Solver are  $\{C, D\}$ ,  $\{A, E\}$ , or  $\{A, F\}$ , while the Adversary selects  $\{A, D, E\}$ , and the approximation ratio is  $\rho = \frac{3}{2}$ .

2. The first data item chosen by the Solver is a vertex of **degree 2**.

**(2.a)** If the Solver decides to take it, then the Adversary presents an isomorphic copy of  $G_1$ , in Figure 1, in which the data item chosen by the Solver is  $A$ . Any solution for the Solver has size at most 2, and the possibilities are  $\{A, C\}$ ,  $\{A, E\}$ ,  $\{A, F\}$ , while the Adversary chooses  $\{B, D, F\}$ . The approximation ratio awarded to the Solver is  $\rho = \frac{3}{2}$ .

**(2.b)** If the Solver decided to reject the the node of degree 2, then the Adversary presents an isomorphic copy of graph  $G_2$  in Figure 1, in which the node chosen by the Solver is  $A$ . The possibilities for the Solver are  $\{B, C\}, \{B, F\}, \{C, E\}$ , or  $\{D, E\}$  of size 2, while the Adversary chooses  $\{A, D, E\}$ . The approximation ratio awarded to the Solver is  $\rho = \frac{3}{2}$ .

## 4 Future Work

Our future goal is to build another model which combines the dynamic programming approach and divide and conquer strategy. We hope that this formalization would deliver even stronger and more expressive formal model than the current pBP model.

We believe that this line of research will deepen our understanding of the basic algorithmic design paradigms by presenting a rigorous statement of the strengths and weaknesses of each technique. It would help the algorithm designers by presenting a new approach to certifying algorithmic solutions as optimal or combinatorial problems hard for many techniques. We hope that such exploration could also help the development of new algorithmic techniques by deeply understanding the weakness of the current approaches to optimization.

**Acknowledgment:** The author would like to thank Jeanne Ferrante and Russell Impagliazzo.

## References

- [1] Borodin, A., Nielsen, M.N., Rackoff, C.: (Incremental) Priority Algorithms. *Algorithmica* **37**(4) (2003) 295–326
- [2] Davis, S., Impagliazzo, R.: Models of Greedy Algorithms for Graph Problems. In: *SODA*. (2004) 381–390
- [3] Alekhovich, M., Borodin, A., Buresh-Oppenheim, J., Impagliazzo, R., Magen, A., Pitassi, T.: Toward a Model for Backtracking and Dynamic Programming. In: *IEEE Conference on Computational Complexity*. (2005) 308–322
- [4] Buresh-Oppenheim, J., Davis, S., Impagliazzo, R.: A Stronger Model of Dynammic Programming Algorithms. *Manuscript* (2006)
- [5] Vazirani, V.V.: *Approximation Algorithms*. Springer (2001)
- [6] Angelopoulos, S., Borodin, A.: The Power of Priority Algorithms for Facility Location and Set Cover. *Algorithmica* **40**(4) (2004) 271–291