

# Empirical Performance Models

B.A. Rachunok

School of Industrial Engineering  
Purdue University

September 10, 2016



# History

- ▶ Early work by Cheeseman *et al.* (1991) → Varied parameters and looked at results
- ▶ Fink (1998) → Used regression to predict which of three algorithms would work best
- ▶ Leyton-Brown *et al.* (2003) → Predicted runtimes for several solvers.
- ▶ I read the big papers by Leyton-Brown *et al.* and their work will be the focus of this presentation

# History

- ▶ Early work by Cheeseman *et al.* (1991) → Varied parameters and looked at results
- ▶ Fink (1998) → Used regression to predict which of three algorithms would work best
- ▶ Leyton-Brown *et al.* (2003) → Predicted runtimes for several solvers.
- ▶ I read the big papers by Leyton-Brown *et al.* and their work will be the focus of this presentation

# History

- ▶ Early work by Cheeseman *et al.* (1991) → Varied parameters and looked at results
- ▶ Fink (1998) → Used regression to predict which of three algorithms would work best
- ▶ Leyton-Brown *et al.* (2003) → Predicted runtimes for several solvers.
- ▶ I read the big papers by Leyton-Brown *et al.* and their work will be the focus of this presentation

# History

- ▶ Early work by Cheeseman *et al.* (1991) → Varied parameters and looked at results
- ▶ Fink (1998) → Used regression to predict which of three algorithms would work best
- ▶ Leyton-Brown *et al.* (2003) → Predicted runtimes for several solvers.
- ▶ I read the big papers by Leyton-Brown *et al.* and their work will be the focus of this presentation

# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

Consider

the problem of finding a polynomial  $p(x)$  of degree  $d$  with integer coefficients such that  $p(x)$  is divisible by  $n$  for all integers  $x$  in the range  $0 \leq x < n$ .  
This is a problem that is  $\text{NP-hard}$  in general.  
The problem is  $\text{NP-hard}$  even if  $n$  is a prime.  
The problem is  $\text{NP-hard}$  even if  $n$  is a power of a prime.  
The problem is  $\text{NP-hard}$  even if  $n$  is a product of two primes.

# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

Consider

Let  $\mathcal{A}$  be a set of algorithms. Let  $\mathcal{I}$  be a set of instances. Let  $T_{\mathcal{A}, \mathcal{I}}$  be the running time of  $\mathcal{A}$  on  $\mathcal{I}$ . Let  $\mathcal{C}$  be a complexity class. Let  $\mathcal{I}_{\mathcal{C}}$  be the set of instances in  $\mathcal{I}$  that are in  $\mathcal{C}$ . Let  $T_{\mathcal{A}, \mathcal{I}_{\mathcal{C}}}$  be the running time of  $\mathcal{A}$  on  $\mathcal{I}_{\mathcal{C}}$ . Let  $\mathcal{I}_{\mathcal{C}^c}$  be the set of instances in  $\mathcal{I}$  that are not in  $\mathcal{C}$ . Let  $T_{\mathcal{A}, \mathcal{I}_{\mathcal{C}^c}}$  be the running time of  $\mathcal{A}$  on  $\mathcal{I}_{\mathcal{C}^c}$ . Let  $\mathcal{I}_{\mathcal{C}^c}^{\text{hard}}$  be the set of instances in  $\mathcal{I}_{\mathcal{C}^c}$  that are hard for  $\mathcal{A}$ . Let  $T_{\mathcal{A}, \mathcal{I}_{\mathcal{C}^c}^{\text{hard}}}$  be the running time of  $\mathcal{A}$  on  $\mathcal{I}_{\mathcal{C}^c}^{\text{hard}}$ . Let  $\mathcal{I}_{\mathcal{C}^c}^{\text{easy}}$  be the set of instances in  $\mathcal{I}_{\mathcal{C}^c}$  that are easy for  $\mathcal{A}$ . Let  $T_{\mathcal{A}, \mathcal{I}_{\mathcal{C}^c}^{\text{easy}}}$  be the running time of  $\mathcal{A}$  on  $\mathcal{I}_{\mathcal{C}^c}^{\text{easy}}$ . Let  $\mathcal{I}_{\mathcal{C}^c}^{\text{easy}}$  be the set of instances in  $\mathcal{I}_{\mathcal{C}^c}$  that are easy for  $\mathcal{A}$ . Let  $T_{\mathcal{A}, \mathcal{I}_{\mathcal{C}^c}^{\text{easy}}}$  be the running time of  $\mathcal{A}$  on  $\mathcal{I}_{\mathcal{C}^c}^{\text{easy}}$ .

# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

## Consider

- ▶ TSP is  $\mathcal{O}(n!)$  (if solved via brute force)
- ▶ However I can solve instances with 5000 points on my gross old laptop?
- ▶ Imagine an algorithm with  $\mathcal{O}(n^{4e10000})$  performance?
- ▶ ... still polynomial



# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

## Consider

- ▶ TSP is  $\mathcal{O}(n!)$  (if solved via brute force)
- ▶ However I can solve instances with 5000 points on my gross old laptop?
- ▶ Imagine an algorithm with  $\mathcal{O}(n^{4e10000})$  performance?
- ▶ ... still polynomial

# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

## Consider

- ▶ TSP is  $\mathcal{O}(n!)$  (if solved via brute force)
- ▶ However I can solve instances with 5000 points on my gross old laptop?
- ▶ Imagine an algorithm with  $\mathcal{O}(n^{4e10000})$  performance?
- ▶ ... still polynomial

# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

## Consider

- ▶ TSP is  $\mathcal{O}(n!)$  (if solved via brute force)
- ▶ However I can solve instances with 5000 points on my gross old laptop?
- ▶ Imagine an algorithm with  $\mathcal{O}(n^{4e10000})$  performance?
- ▶ ... still polynomial

# EPH Motivation

- ▶ We already have complexity theory to analyze algorithm performance.
- ▶ Why this too?

## Consider

- ▶ TSP is  $\mathcal{O}(n!)$  (if solved via brute force)
- ▶ However I can solve instances with 5000 points on my gross old laptop?
- ▶ Imagine an algorithm with  $\mathcal{O}(n^{4e10000})$  performance?
- ▶ ... still polynomial

## Motivation Pt 2

- ▶ Knowing how a particular instance of a problem will work gives us some options
- ▶ Potentially select the "best" predicted solver for a given instance (more on this)
- ▶ Provide some insights into how solvers handle instances (eg 4.3 clauses-to-literal ratio)

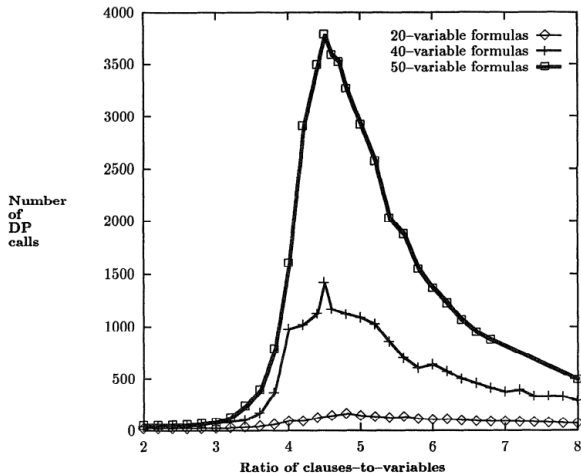
## Motivation Pt 2

- ▶ Knowing how a particular instance of a problem will work gives us some options
- ▶ Potentially select the "best" predicted solver for a given instance (more on this)
- ▶ Provide some insights into how solvers handle instances (eg 4.3 clauses-to-literal ratio)

## Motivation Pt 2

- ▶ Knowing how a particular instance of a problem will work gives us some options
- ▶ Potentially select the "best" predicted solver for a given instance (more on this)
- ▶ Provide some insights into how solvers handle instances (eg 4.3 clauses-to-literal ratio)

# Clause-to-Literal Ratio



Graph taken from Mitchell, Selman, Levesque 1992



To the board

# Types of Models

- ▶ Ridge Regression
- ▶ Neural Networks
- ▶ Gaussian Process Regression
- ▶ Regression Trees
- ▶ INSERT WHATEVER I USED HERE

# Types of Models

- ▶ Ridge Regression
- ▶ Neural Networks
- ▶ Gaussian Process Regression
- ▶ Regression Trees
- ▶ INSERT WHATEVER I USED HERE

# Types of Models

- ▶ Ridge Regression
- ▶ Neural Networks
- ▶ Gaussian Process Regression
- ▶ Regression Trees
- ▶ INSERT WHATEVER I USED HERE

# Types of Models

- ▶ Ridge Regression
- ▶ Neural Networks
- ▶ Gaussian Process Regression
- ▶ Regression Trees
- ▶ INSERT WHATEVER I USED HERE

# Types of Models

- ▶ Ridge Regression
- ▶ Neural Networks
- ▶ Gaussian Process Regression
- ▶ Regression Trees
- ▶ INSERT WHATEVER I USED HERE

# Types of Features for MIP

- ▶ Number of constraints and variables
- ▶ Variable types
- ▶  $\leq$ ,  $\geq$ , or  $=$  for the RHS
- ▶ Mean of objective function coefficients

# Types of Features for MIP

- ▶ Number of constraints and variables
- ▶ Variable types
- ▶  $\leq$ ,  $\geq$ , or  $=$  for the RHS
- ▶ Mean of objective function coefficients



# Types of Features for MIP

- ▶ Number of constraints and variables
- ▶ Variable types
- ▶  $\leq$ ,  $\geq$ , or  $=$  for the RHS
- ▶ Mean of objective function coefficients

# Types of Features for MIP

- ▶ Number of constraints and variables
- ▶ Variable types
- ▶  $\leq$ ,  $\geq$ , or  $=$  for the RHS
- ▶ Mean of objective function coefficients

# Types of Features for TSP

- ▶ Number of nodes
- ▶ Cluster distance
- ▶ Area spanned by nodes
- ▶ Centroid of points
- ▶ NEAREST NEIGHBOR path length

# Types of Features for TSP

- ▶ Number of nodes
- ▶ Cluster distance
- ▶ Area spanned by nodes
- ▶ Centroid of points
- ▶ NEAREST NEIGHBOR path length

# Types of Features for TSP

- ▶ Number of nodes
- ▶ Cluster distance
- ▶ Area spanned by nodes
- ▶ Centroid of points
- ▶ NEAREST NEIGHBOR path length

# Types of Features for TSP

- ▶ Number of nodes
- ▶ Cluster distance
- ▶ Area spanned by nodes
- ▶ Centroid of points
- ▶ NEAREST NEIGHBOR path length

# Types of Features for TSP

- ▶ Number of nodes
- ▶ Cluster distance
- ▶ Area spanned by nodes
- ▶ Centroid of points
- ▶ NEAREST NEIGHBOR path length

# Example Time

- ▶ now to the example



# Who cares?

- ▶ Why do we care about this at all?
- ▶ Consider *two* prediction models for *two* solvers
- ▶ Consider *three* prediction models for *three* solvers
- ▶ ...
- ▶ Consider *n* prediction models for *n* solvers

# Who cares?

- ▶ Why do we care about this at all?
- ▶ Consider *two* prediction models for *two* solvers
- ▶ Consider *three* prediction models for *three* solvers
- ▶ ...
- ▶ Consider *n* prediction models for *n* solvers

# Who cares?

- ▶ Why do we care about this at all?
- ▶ Consider *two* prediction models for *two* solvers
- ▶ Consider *three* prediction models for *three* solvers
- ▶ ...
- ▶ Consider *n* prediction models for *n* solvers

# Who cares?

- ▶ Why do we care about this at all?
- ▶ Consider *two* prediction models for *two* solvers
- ▶ Consider *three* prediction models for *three* solvers
- ▶ ...
- ▶ Consider  $n$  prediction models for  $n$  solvers

# Who cares?

- ▶ Why do we care about this at all?
- ▶ Consider *two* prediction models for *two* solvers
- ▶ Consider *three* prediction models for *three* solvers
- ▶ ...
- ▶ Consider *n* prediction models for *n* solvers

# Who cares cont'd ...

- ▶ If we can *cheaply* compute features and make predictions
- ▶ ... then we can solve our instance with the best algorithm from a *portfolio*
- ▶ eg SATZILLA

# Who cares cont'd ...

- ▶ If we can *cheaply* compute features and make predictions
- ▶ ... then we can solve our instance with the best algorithm from a *portfolio*
- ▶ *eg* SATZILLA

# Who cares cont'd . . .

- ▶ If we can *cheaply* compute features and make predictions
- ▶ . . . then we can solve our instance with the best algorithm from a *portfolio*
- ▶ eg SATZILLA