

# **ParamILS:** An Automatic Algorithm Configuration Framework

Hutter, Hoos, Leyton-Brown, Stützle

JAIR'09

# Reading

Comprehensive journal paper on a seminal work

Check website for specific sections to read 🙄

Feel free to read more at your own interest 😊

# Integer programming and SAT

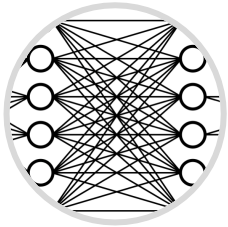
## Integer program (IP)

$$\begin{aligned} \max \quad & \mathbf{c} \cdot \mathbf{z} \\ \text{s.t.} \quad & A\mathbf{z} \leq \mathbf{b} \\ & \mathbf{z} \in \mathbb{Z}^n \end{aligned}$$

## SAT

$$\begin{aligned} & (x_1 \vee x_4) \wedge (x_1 \vee \bar{x}_3 \vee \bar{x}_8) \\ & \wedge (x_1 \vee x_8 \vee x_{12}) \wedge (x_2 \vee x_{11}) \end{aligned}$$

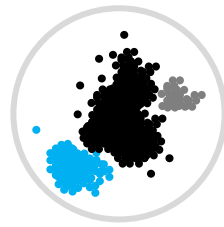
**Tons** of applications:



Robust ML



MAP estimation



Clustering



Routing



Scheduling

# Algorithm configuration

Solvers come with tons of tunable parameters

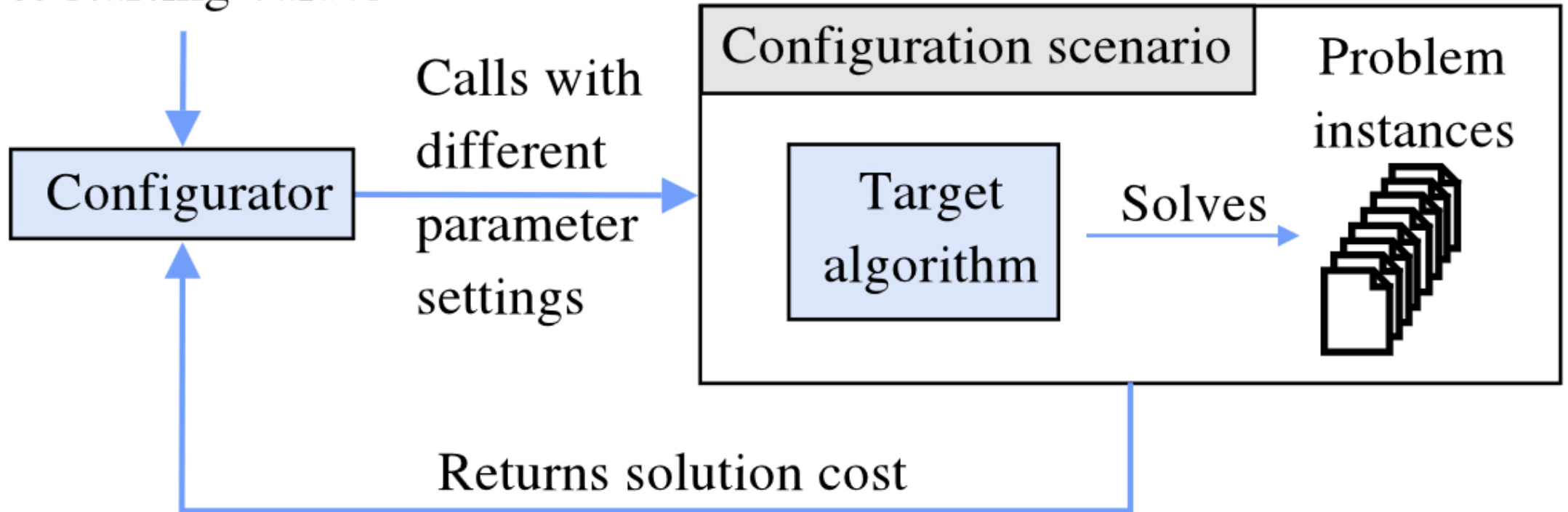
Tuning by hand is notoriously **slow, tedious**, and **error-prone**

Can we **automatically** optimize parameters?

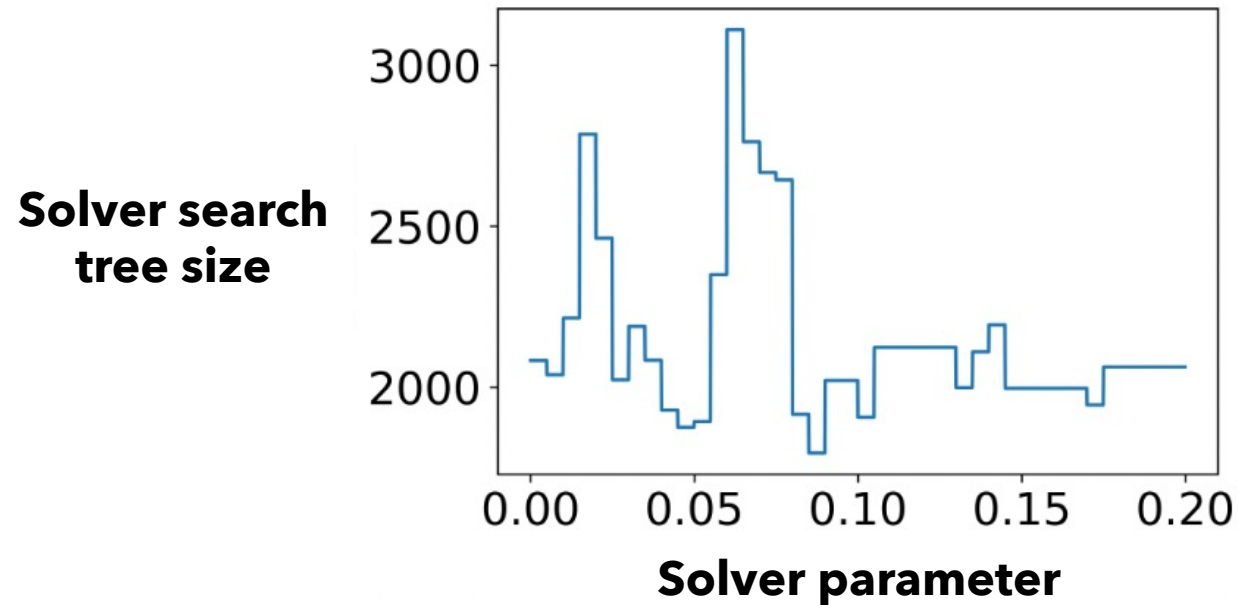
CPX_PARAM_NODEFILEIND 100	CPX_PARAM_TRELIM 160	CPX_PARAM_RANDOMSEED 130	CPXPARAM_MIP_Pool_RelGap 148	CPX_PARAM_FLOWCOVERS 70	CPX_PARAM_BRDIR 39
CPX_PARAM_NODELIM 101	CPX_PARAM_TUNINGDETILIM 160	CPX_PARAM_REDUCE 131	CPXPARAM_MIP_Pool_Replace 151	CPX_PARAM_FLOWPATHS 71	CPX_PARAM_BTTOL 40
CPX_PARAM_NODESEL 102	CPX_PARAM_TUNINGDISPLAY 162	CPX_PARAM_REINV 131	CPXPARAM_MIP_Strategy_Branch 39	CPX_PARAM_FPHEUR 72	CPX_PARAM_CALCQCPDUALS 41
CPX_PARAM_NUMERICALEMPHASIS 102	CPX_PARAM_TUNINGMEASURE 163	CPX_PARAM_RELAXPREIND 132	CPXPARAM_MIP_Strategy_MIQCPStrat 93	CPX_PARAM_FRACCAND 73	CPX_PARAM_CLIQUES 42
CPX_PARAM_NZREADLIM 103	CPX_PARAM_TUNINGREPEAT 164	CPX_PARAM_RELOBJDIF 133	CPXPARAM_MIP_Strategy_StartAlgorithm 139	CPX_PARAM_FRACCUTS 73	CPX_PARAM_CLOCKTYPE 43
CPX_PARAM_OBJDIF 104	CPX_PARAM_TUNINGTILIM 165	CPX_PARAM_REPAIRTRIES 133	CPXPARAM_MIP_Strategy_VariableSelect 166	CPX_PARAM_FRACPASS 74	CPX_PARAM_CLONELOG 43
CPX_PARAM_OBJLLIM 105	CPX_PARAM_VARSEL 166	CPX_PARAM_REPEATPRESOLVE 134	CPXPARAM_MIP_SubMIP_NodeLimit 155	CPX_PARAM_GUBCOVERS 75	CPX_PARAM_COEREDIND 44
CPX_PARAM_OBJULIM 105	CPX_PARAM_WORKDIR 167	CPX_PARAM_RINSHEUR 135	CPXPARAM_OptimalityTarget 106	CPX_PARAM_HEURFREQ 76	CPX_PARAM_COLREADLIM 45
CPX_PARAM_PARALLELMODE 108	CPX_PARAM_WORKMEM 168	CPX_PARAM_RLT 136	CPXPARAM_Output_WriteLevel 169	CPX_PARAM_IMPLBD 76	CPX_PARAM_CONFLICTDISPLAY 46
CPX_PARAM_PERIND 110	CPX_PARAM_WRITELEVEL 169	CPX_PARAM_ROWREADLIM 141	CPXPARAM_Preprocessing_Aggregator 19	CPX_PARAM_INTSOLFILEPREFIX 78	CPX_PARAM_COVERS 47
CPX_PARAM_PERLIM 111	CPX_PARAM_ZEROHALFCUTS 170	CPX_PARAM_SCAIND 142	CPXPARAM_Preprocessing_Fill 19	CPX_PARAM_INTSOLLIM 79	CPX_PARAM_CPUMASK 48
CPX_PARAM_POLISHAFTERDETTIME 111	CPXPARAM_Benders_Strategy 30	CPX_PARAM_SCRIND 143	CPXPARAM_Preprocessing_Linear 120	CPX_PARAM_ITLIM 80	CPX_PARAM_CRAIN 50
CPX_PARAM_POLISHAFTEREPAGAP 112	CPXPARAM_Benders_Tolerances_feasibilitycut 35	CPX_PARAM_SIFTALG 143	CPXPARAM_Preprocessing_Reduce 131	CPX_PARAM_LANDPCUTS 82	CPX_PARAM_CUTLO 51
CPX_PARAM_POLISHAFTEREPGAP 113	CPXPARAM_Benders_Tolerances_optimalitycut 36	CPX_PARAM_SIFTDISPLAY 144	CPXPARAM_Preprocessing_Symmetry 156	CPX_PARAM_LBHEUR 81	CPX_PARAM_CUTPASS 52
CPX_PARAM_POLISHAFTERINTSOL 114	CPXPARAM_Conflict_Algorithm 46	CPX_PARAM_SIFTITILIM 145	CPXPARAM_Read_DataCheck 54	CPX_PARAM_LPMETHOD 136	CPX_PARAM_CUTSFACTOR 52
CPX_PARAM_POLISHAFTERNODE 115	CPXPARAM_CPUmask 48	CPX_PARAM_SIMDISPLAY 145	CPXPARAM_Read_Scale 142	CPX_PARAM_MFCUTS 82	CPX_PARAM_CUTUP 53
CPX_PARAM_POLISHAFTERTIME 116	CPXPARAM_DistMIP_Rampup_Duration 128	CPX_PARAM_SINGLIM 146	CPXPARAM_ScreenOutput 143	CPX_PARAM_MEMORYEMPHASIS 83	CPXPARAM_DATACHECK 54
CPX_PARAM_POLISHTIME (deprecated) 116	CPXPARAM_LPMethod 136	CPX_PARAM_SOLNPOOLGAP 146	CPXPARAM_Sifting_Algorithm 143	CPX_PARAM_MIPCBREDLP 84	CPX_PARAM_DEPIND 55
CPX_PARAM_POPULATELIM 117	CPXPARAM_MIP_Cuts_BQP 38	CPX_PARAM_SOLNPOOLCAPACITY 147	CPXPARAM_Sifting_Display 144	CPX_PARAM_MIPDISPLAY 85	CPX_PARAM_DETTILIM 56
CPX_PARAM_PPRIIND 118	CPXPARAM_MIP_Cuts_LocalImplied 77	CPX_PARAM_SOLNPOOLGAP 148	CPXPARAM_Sifting_Iterations 145	CPX_PARAM_MIPEMPHASIS 87	CPX_PARAM_DISJCUTS 57
CPX_PARAM_PREDUAL 119	CPXPARAM_MIP_Cuts_RLT 136	CPX_PARAM_SOLNPOOLINTENSITY 149	CPXPARAM_Simplex_Display 145	CPX_PARAM_MIPINTERVAL 88	CPX_PARAM_DIVETYPE 58
CPX_PARAM_PREIND 120	CPXPARAM_MIP_Cuts_ZeroHalfCut 170	CPX_PARAM_SOLNPOOLREPLACE 151	CPXPARAM_Simplex_Limits_Singularity 146	CPX_PARAM_MIPKAPPASTATS 89	CPX_PARAM_DPRIIND 59
CPX_PARAM_PRLINEAR 120	CPXPARAM_MIP_Limits_CutsFactor 52	CPX_PARAM_SOLNPOOLREPLACE 151	CPXPARAM_SolutionType 152	CPX_PARAM_MIPORDIND 90	CPX_PARAM_EACHCUTLIM 60
CPX_PARAM_PREPASS 121	CPXPARAM_MIP_Limits_RampupDetTimeLimit 127	CPX_PARAM_SOLUTIONTARGET (deprecated: see CPXPARAM_OptimalityTarget 106)	CPXPARAM_Threads 157	CPX_PARAM_MIPORDTYPE 91	CPX_PARAM_EPAGAP 61
CPX_PARAM_PRESLVND 122	CPXPARAM_MIP_Limits_RampupTimeLimit 128	CPX_PARAM_SOLUTIONTYPE 152	CPXPARAM_TimeLimit 159	CPX_PARAM_MIPSEARCH 92	CPX_PARAM_EPGAP 61
CPX_PARAM_PRICELIM 123	CPXPARAM_MIP_Limits_Solutions 79	CPX_PARAM_STARTALG 139	CPXPARAM_Tune_DefTimeLimit 160	CPX_PARAM_MIQCPSTRAT 93	CPX_PARAM_EPINT 62
CPX_PARAM_PROBE 123	CPXPARAM_MIP_Limits_StrongCand 154	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Display 162	CPX_PARAM_MIRCUTS 94	CPX_PARAM_EPMRK 64
CPX_PARAM_PROBEDETTIME 124	CPXPARAM_MIP_Limits_StrongIt 154	CPX_PARAM_STRONGCANDLIM 154	CPXPARAM_Tune_Measure 163	CPX_PARAM_MPSLONGNUM 94	CPX_PARAM_EPOPT 65
CPX_PARAM_PROBETIME 124	CPXPARAM_MIP_Limits_TreeMemory 160	CPX_PARAM_STRONGITLIM 154	CPXPARAM_Tune_Repeat 164	CPX_PARAM_NETDISPLAY 95	CPX_PARAM_EPPER 65
CPX_PARAM_QPMAKEPSDIND 125	CPXPARAM_MIP_OrderType 91	CPX_PARAM_SUBBALG 99	CPXPARAM_Tune_TimeLimit 165	CPX_PARAM_NETEPOPT 96	CPX_PARAM_NETEPRHS 66
CPX_PARAM_QPMETHOD 138	CPXPARAM_MIP_Pool_AbsGap 146	CPX_PARAM_SUBMIPNODELIMIT 155	CPXPARAM_WorkDir 167	CPX_PARAM_NETEPRHS 96	CPX_PARAM_EPRHS 67
CPX_PARAM_QPNZREADLIM 126	CPXPARAM_MIP_Pool_Capacity 147	CPX_PARAM_SYMMETRY 156	CPXPARAM_WorkMem 168	CPX_PARAM_NETFIND 97	CPX_PARAM_FEASOPTMODE 68
	CPXPARAM_MIP_Pool_Intensity 149	CPX_PARAM_THREADS 157	CraInd 50	CPX_PARAM_NETITLIM 98	CPX_PARAM_FILEENCODING 69
		CPX_PARAM_TILIM 159		CPX_PARAM_NETPRIIND 98	

# Algorithm configuration pipeline

Parameter domains  
& starting values



# Key challenge



- Solver performance is **extremely volatile**
- Gradients are often **uninformative**

# Outline

1. Introduction
- 2. Setup**
3. ParamLS
4. Experiments
5. Spectrum auctions

# Setup: Parameterized algorithm

Algorithm  $\mathcal{A}$  with  $k$  parameters

$i^{\text{th}}$  parameter setting from a set  $\Theta_i$

Assume  $|\Theta_i|$  is finite (e.g., by discretizing continuous parameters)

$\Theta \subseteq \Theta_1 \times \cdots \times \Theta_k$  is the set of all feasible configurations

In experiments,  $|\Theta|$  as large as  $1.38 \cdot 10^{37}$

$\mathcal{A}(\boldsymbol{\theta})$  is the algorithm with parameters  $\boldsymbol{\theta} \in \Theta$



# Setup: Modeling the application domain

Set of problem instances  $\Pi$

- E.g.,  $\pi \in \Pi$  is a routing integer program

Application-specific distribution  $\mathcal{D}$  over problem instances, e.g.:

- Distribution over Bay Area routing problems
- Uniform distribution over benchmark dataset



# Setup: Measuring performance

$o(\boldsymbol{\theta}, \pi, \kappa)$ : runtime of  $\mathcal{A}(\boldsymbol{\theta})$  on instance  $\pi$  with runtime cap  $\kappa$

$\kappa_{\max}$ : maximum runtime after which any run will be terminated

$c(\boldsymbol{\theta})$ : overall cost of running algorithm with parameters  $\boldsymbol{\theta}$

- E.g., expected runtime  $c(\boldsymbol{\theta}) = \mathbb{E}_{\pi \sim \mathcal{D}}[o(\boldsymbol{\theta}, \pi, \kappa_{\max})]$
- If  $\mathcal{D}$  is the uniform distribution over a benchmark set  $\Pi$ , equivalent to

$$c(\boldsymbol{\theta}) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} o(\boldsymbol{\theta}, \pi, \kappa_{\max})$$

# Algorithm configuration goal

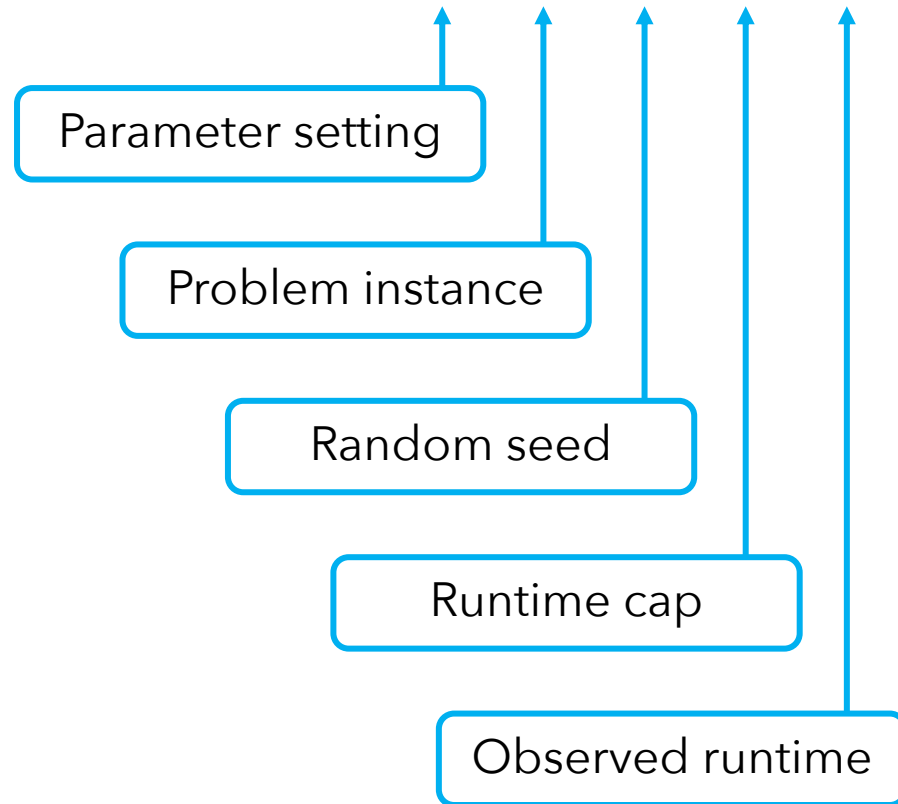
**Goal:** Find parameter setting  $\theta$  with low cost  $c(\theta)$

## **Challenges:**

- Distribution  $\mathcal{D}$  may be unknown
- Do not know analytical form of  $c$
- $c$  may be nonconvex, non-Lipschitz, ...

# Empirical cost

Sequence of runs  $\mathbf{R} = ((\boldsymbol{\theta}_1, \pi_1, s_1, \kappa_1, o_1), \dots, (\boldsymbol{\theta}_n, \pi_n, s_n, \kappa_n, o_n))$



# Empirical cost

Sequence of runs  $\mathbf{R} = ((\boldsymbol{\theta}_1, \pi_1, s_1, \kappa_1, o_1), \dots, (\boldsymbol{\theta}_n, \pi_n, s_n, \kappa_n, o_n))$

Empirical cost  $\hat{c}(\boldsymbol{\theta}, \mathbf{R}) = \text{average}(\{o_i: \boldsymbol{\theta} = \boldsymbol{\theta}_i\})$

Could be replaced by another statistic, e.g., median

**Goal:** Find  $\boldsymbol{\theta}$  with low empirical (training) cost  $\hat{c}(\boldsymbol{\theta}, \mathbf{R})$

- Ideally, this should lead to low **actual** cost  $c(\boldsymbol{\theta})$
- Later this quarter: statistical guarantees for bounding  $|c(\boldsymbol{\theta}) - \hat{c}(\boldsymbol{\theta}, \mathbf{R})|$

# Key questions

1. Which parameter configurations  $\Theta' \subseteq \Theta$  should we evaluate?
2. Which instances  $\Pi_{\theta'} \subseteq \Pi$  should we use to evaluate  $\theta' \in \Theta'$ ?
3. Which cutoff times  $\kappa_i$  should we use for each run?

# Outline

1. Introduction
2. Setup
- 3. ParamLS**
4. Experiments
5. Spectrum auctions

Param**ILS** ← Iterated local search

## Param**ILS**:

A seminal general-purpose algorithm configuration procedure

- Begins with a **default** parameter configuration
- Performs **local search** in configuration space
  - Changes the setting of **one parameter at a time**
  - Keeps those changes resulting in performance improvements
- After finding a **local minimum**:  
**Randomly changes** some parameters in order to escape



# BasicLS

**Step 1:** Randomly search for good initial configuration

- i.  $\theta_0$ : initial configuration
- ii. For  $i = 1, \dots, r$ :
  - a.  $\theta \leftarrow \text{random}(\Theta)$
  - b. if BETTER( $\theta, \theta_0$ ),  $\theta_0 \leftarrow \theta$

Essentially, run  $\mathcal{A}(\theta)$  on some random instances and compare  $\hat{c}(\theta, \mathbf{R})$  and  $\hat{c}(\theta_0, \mathbf{R})$

# BasicLS

**Step 2:** Search for better configuration in neighborhood of  $\theta_0$

$$\theta_{\text{ils}} \leftarrow \text{ITERATIVEFIRSTIMPROVEMENT}(\theta_0)$$

ITERATIVEFIRSTIMPROVEMENT( $\theta$ ):

Repeat:

- i.  $\theta' \leftarrow \theta$
- ii. Find best parameter setting  $\theta''$  in the neighborhood of  $\theta'$  according to  $\hat{c}$

$\theta''$  differs from  $\theta'$  in one component

# BasicLS

**Step 2:** Search for better configuration in neighborhood of  $\theta_0$

$\theta_{\text{ils}} \leftarrow \text{ITERATIVEFIRSTIMPROVEMENT}(\theta_0)$

ITERATIVEFIRSTIMPROVEMENT( $\theta$ ):

Repeat:

- i.  $\theta' \leftarrow \theta$
- ii. Find best parameter setting  $\theta''$  in the neighborhood of  $\theta'$  according to  $\hat{c}$
- iii. Set  $\theta = \theta''$

Until  $\theta' = \theta$   $\Rightarrow$  found a local minimum

# BasicLS

**Step 3** (repeat as many times as you can):

1.  $\theta \leftarrow \theta_{\text{ils}}$
2. for  $s$  rounds do random exploration:  
 $\theta \leftarrow$  random  $\theta'$  in the neighborhood of  $\theta$
3.  $\theta \leftarrow \text{ITERATIVEFIRSTIMPROVEMENT}(\theta)$
4. if  $\text{BETTER}(\theta, \theta_{\text{ils}})$ , set  $\theta_{\text{ils}} \leftarrow \theta$
5. With some small probability, restart:  $\theta \leftarrow \text{random}(\Theta)$

Return best  $\theta$  the algorithm ever found according to  $\hat{c}$

# Adaptive capping

Solving IPs can take forever...

- We want to give up as early as we can
- But still correctly evaluate  $\text{BETTER}(\theta_1, \theta_2)$

Without a good way to **cap** runs, will waste time on bad  $\theta$ 's

# Adaptive capping

## Illustrative example:

- $\theta_1$  takes 10 seconds total to solve 100 instances

$$\hat{c}(\theta_1, \mathbf{R}) = \frac{1}{100} \cdot 10 = 0.1$$

- $\theta_2$  takes at least 11 seconds to solve the first instance

$$\hat{c}(\theta_2, \mathbf{R}) \geq \frac{1}{100} \cdot (11 + \underbrace{0 + 0 + \dots + 0}_{99 \text{ zeros}}) = 0.11$$

- Can stop evaluating  $\theta_2$  11 seconds into the first run

# Simple adaptive capping

BETTER( $\theta_1, \theta_2$ ):

1. Evaluate  $\theta_1$  on  $N$  random instances to compute  $\hat{c}(\theta_1, \mathbf{R})$
2. Evaluate  $\theta_2$  on  $N$  random instances to compute  $\hat{c}(\theta_2, \mathbf{R})$

But **give up** after  $N \cdot \hat{c}(\theta_1, \mathbf{R})$  seconds and return  $\theta_1$

3. If didn't give up, return

$$\begin{cases} \theta_1 & \text{if } \hat{c}(\theta_1, \mathbf{R}) < \hat{c}(\theta_2, \mathbf{R}) \\ \theta_2 & \text{else} \end{cases}$$

Many ways **cap more aggressively** and improve this further

# Outline

1. Introduction
2. Setup
3. ParamLS
- 4. Experiments**
5. Spectrum auctions



# Experiments: example

Average runtime

Scenario	SimpleLS(100)	BasicILS(100)		<i>p</i> -value
	Performance	Performance	Avg. # ILS iterations	
SAPS-SWGCP	$0.5 \pm 0.39$	<b><math>0.38 \pm 0.19</math></b>	2.6	$9.8 \cdot 10^{-4}$
SAPS-QCP	$3.60 \pm 1.39$	<b><math>3.19 \pm 1.19</math></b>	5.6	$4.4 \cdot 10^{-4}$
SPEAR-QCP	$0.4 \pm 0.39$	<b><math>0.36 \pm 0.39</math></b>	1.64	<b>0.008</b>

- SAPS and SPEAR: SAT solvers
- SWGCP: graph coloring problems
- QCP: quasi-group completion problem
- SimpleLS: local search w/o randomization
  - Should get stuck in local minima

# Outline

1. Introduction
2. Setup
3. ParamLS
4. Experiments
- 5. Spectrum auctions**

# Spectrum auctions

Later work by the same UBC lab (and others):

- In 2016-17, FCC held an auction to repurpose radio spectrum
  - Broadcast television → wireless internet
  - In total, the auction yielded \$19.8 billion



Kevin-Leyton Brown  
UBC



Ilya Segal  
Stanford

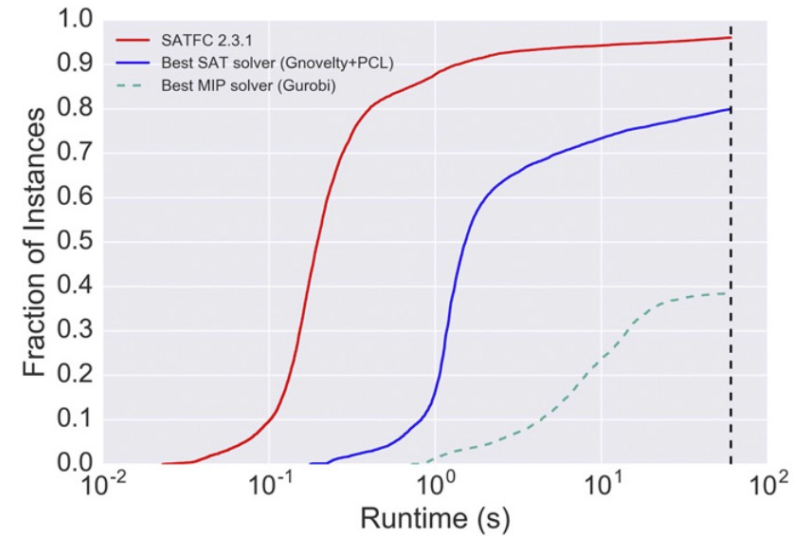
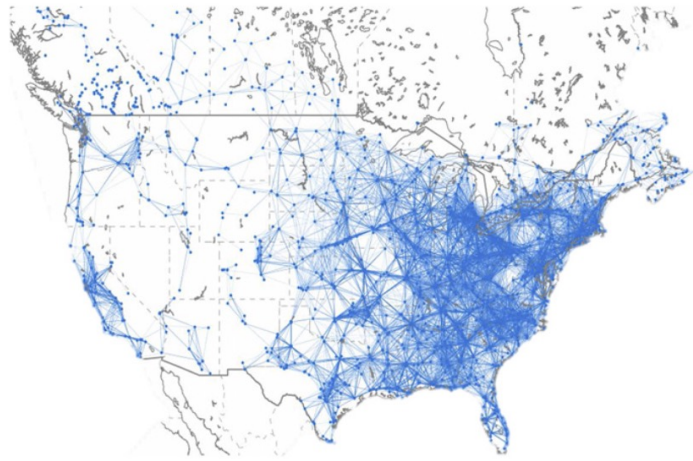


Paul Milgrom  
Stanford

And many others!

# Spectrum auctions

- The auction involves solving huge graph coloring problems



- SATFC uses algorithm configuration + portfolio selection
- Simulations indicate SATFC saved the government billions

# Overview

**ParamLS:** a seminal general-purpose configuration procedure

Combines **local search** with **random exploration**

Speedups for **IP** and **SAT** solvers

Inspired later breakthroughs for **spectrum auctions**