Automatic Algorithm Configuration based on Local Search

Christoph Baldow

January 20th, 2010
Content

**Introduction**
- Motivation
- Main-tasks

**General**
- Assumption
- Goal

**BasicILS**
- General
- Algorithm

**Evaluation(better)**
- Phases and Structures
- Problems of fixed-size training-set just informal

**FocusedILS**
- FocusedILS

**References**

Christoph Baldow
paramILS
Introduction

Motivation

- hard task to find good/best adjustment for effective algorithms
  - parameters: categorical choices, numerical parameters
- tuning of parameters in practice is often done manually by applying rules of thumb or crude heuristics
idea: using a more general approach to tune these algorithms

possibility to tune algorithms in many different properties, with arbitrary number of parameters

versatile tuning-algorithm to tune many different problem-classes

goal: try to outperform always the standard-configuration/random-configuration and find ”good” solution
ILS - iterated local search for algorithm configuration problem

- works for both, randomised and deterministic algorithms
- can be applied regardless of the tuning scenario or optimisation objective

Study the effects of over-confidence and over-tuning

- occur, when an algorithm is tuned based on a finite number of training instances
- already shown by Birattari (2004), find statistical arguments and experimental results

eILS - extended iterated local search

- extended algorithm for configuration problem in order to avoid over-tuning and over-confidence
It must be possible to assign a **scalar cost** to any single run on an **instance**, in the case of a randomised algorithm using a **seed**.

This **scalar cost** could for example be the run-time, approximation error or the improvement achieved over an instance-specific reference cost.
Objective in algorithm configuration is to minimise a statistic of cost distribution.

This is a stochastic optimisation problem, because cost distributions are typically unknown and we have to compute a limited number of samples to approximate their statistics.

⇝ Proof of Birattari (2004)
parameter-tuning for an algorithm can be easy or hard

- if only a few parameters, with only a few possible values exist, one could try every combination, which is also known as full fractional design.
- exponential growth with number of parameters and possible values of parameter.

- if too many parameters exist, typically one starts with an arbitrary configuration and change one parameter at a time until no one gets no improvement.
**Raw Filtering**
- random search
- find better configuration

**Check Termination Criterion**

**Restart**
- new configuration w.r.t. probability

**Perturbation**
- random neighbourhood - relations (s)
- escape local optima

**Acceptance Criterion**
- better-relation
- bet. old a. new configuration

**Local Search**
- Iterative First Improvement
- randomised order
parameters which are only relevant, when some "higher-level" parameters take certain values, we call **conditionals**

- conditionals are handled in `paramILS` by excluding the neighbourhoods from such a parameter configuration
Phases and Structures

- two phases, which are treated differently
  - learning-phase is the algorithm to find a better configuration
  - testing-phase is the phase to evaluate the new configuration
- two different sets of instances are needed: training-set and testing-set
Problems of fixed-size training-set just informal

- **over-confidence**
  - take the configuration with the "best" value on the training-set
  - imagine our Domain have huge instances and our training-set have just one instance
  - result: a good, probably the best $\theta$ for this instance but a worse solution for the whole Domain $\mathcal{D}$
  - it follows that we typically could not take the best configuration for the training-set

- **over-tuning**
  - is equivalent to *overfitting* in machine learning
  - too intensive learning of learning-set implies the learning of the failures
  - poor predictive performance
Example: Overconfidence

Performance

Instances

Configuration1
Configuration2
Example: Overtuning

\[ \epsilon \]

Time per configuration

Cpu-Time
modify **better-function** in order to overcome **over-confidence** and **over-tuning**

**conf}_1 \text{ dominates } conf}_2 \text{ iff } N(conf}_1) \geq N(conf}_2) \text{ and the performance of } conf}_1 \text{ using the first } N(conf}_2) \text{ samples is better than that of } conf}_2
better($conf_1, conf_2$) acquiring one sample for the configuration with smaller $N(conf_i)$ for $i \in 1, 2$, in case of ties one for each

whenever better($conf_1, conf_2$) returns true, we boost total number of samples with ”bonus samples”
Frank Hutter, Holger. H. Hoos and Thomas Stützle
Automatic Algorithm Configuration based on Local Search

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown and Thomas Stützle
An Automatic Algorithm Configuration Framework