NP-completeness, NFL and Statistical Testing

Thomas Stidsen

thst@man.dtu.dk

DTU-Management

Technical University of Denmark

Outline

- NP completeness:
  - What are the hard problems ?
  - ... and how hard are they ?
- No-Free-Lunch theorem:
  - Are there any best search algorithms ?
- Statistical Testing:
  - How to maximize performance ...

(Time) Complexity Theory

(Time) Complexity Theory is:

- How many “simple” operations needs to be performed by an algorithm to “solve” a task ?
- This time is measured in big O notation: E.g. sorting: O(N log(N)) where N is the number of entries which needs to be sorted.

NOTICE: We are talking about *asymptotic* behavior which may only be relevant for large problems.

Types of tasks

In general we differentiate between 3 types of tasks to perform:

1. The impossible (the Halting Theorem): Not many ...
2. The hard (time consuming): Many important ....
3. The easy problems: Many important ... (but already solved !)

For now we will ignore the impossible problems, which we usually do not encounter and concentrate on the difference between the easy and the hard problems.

Turing Machines (TM)

I do not want to go into (too many) details on Turing Machines:

- They are theoretical models for (almost) all kinds of computation today ...
- Can be divided into two types:
  - The deterministic TM
  - The non-deterministic TM

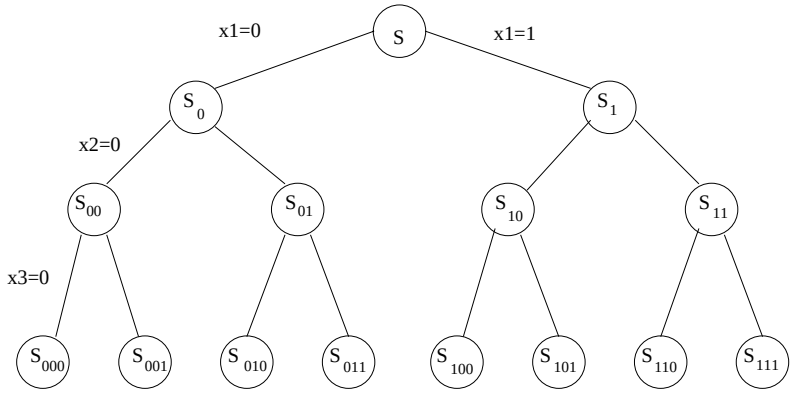
Deterministic TM

- Somewhat corresponding to computers today
- Can read and execute programs

Non-deterministic TM

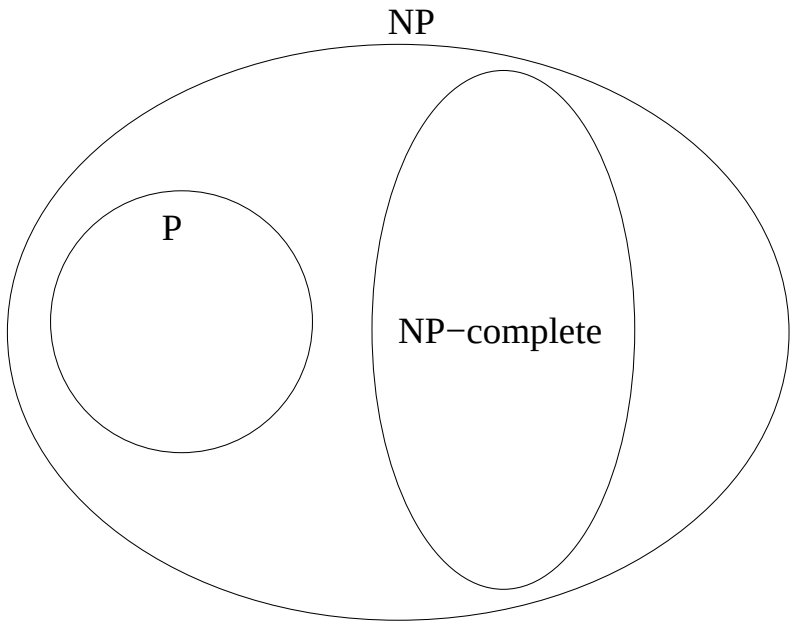
A very special type of machine !

- The machine is able to make lucky decisions !
- Can branch in polynomial time through a branch-and-bound tree ...



The computable problems

First we will only consider decision problems (yes/no problems):



NP-complete

A special and hard set of decision problems:

- It should belong the set NP
- It should be at least as hard as any other NP problem

Most often a problem is proven to be hard by proving that it belongs to the NP-complete class of problems.

Proving NP-completeness

Given a new decision problem Q

- Prove that $Q \in NP$, i.e. given a solution there should be a polynomial algorithm to check if the answer is yes or no.
- Make a (polynomial) transformation algorithm to convert some known problem to Q

Decision problems

An important detail: Complexity Theory deals with **decision problems** with a yes/no result. It is however easy to convert an optimization problem to a decision problem: Introduce a "question": Is the result less/more than x ?

We call optimization problems which have NP-complete decision problem counterparts for: **NP-hard** optimization problems.

$P \neq NP$????

The classes, P, NP and NP complete were formulated in the 70'ies, but has not yet been proved !!! One of the most important unsolved mathematical problems. Most (all !) researchers assume that $P \neq NP$.

What is the point showing NP-completeness ?

- It feels stupid to used advanced metaheuristics/decomposition algorithms, if other people can solve the problem fast to optimality ...
- If a problem is NP-complete is **EXTREMELY UNLIKELY** to be a simple problem to solve ...
- ... because then thousands of other peoples problems can be solved to optimality ...
- Optimization problems can be changed into decision problems by asking if solutions of a value less or greater than a certain number exists.

The No Free Lunch Theorem for Search

Assume to have two heuristic optimisation algorithms a_1 and a_2. Further assume to have a function f to optimize and a maximum of m evaluations encountered during the search. The vector $\overrightarrow{c}$ is a histogram over all the evaluations during the search. The NFL theorem says:

$$\sum_f P(\overrightarrow{c}|f, m, a_1) = \sum_f P(\overrightarrow{c}|f, m, a_2)$$

The No Free Lunch Theorem for Search

Originates from an article of David H. Wolpert and William G. Macready, both at the Santa Fe Institute, from 1995

`http://www.santafe.edu/~wgm/papers.html`

- In the beginning it was **extremely controversial**.
- It was discussed for more than 2 years on the GA mailing list ...

It is both very elegant mathematically and very surprising !

But is it useful ?

The fundamental problem with NFL is: Which problems does it actually consider ?

- **All** ?! But what do we mean by all ? This might include many problems we will never encounter ...
- The majority of problems we encounter has some “continuity”.

The lesson to be learned by NFL: All heuristic search algorithms favors certain problems where they perform well, make sure that your algorithms are good for the particular problems.

Statistical Testing

How can we scientifically examine the performance of two different meta-heuristics for a specific problem ? This is hard because:

- The meta-heuristics are parameter dependent
- The meta-heuristics are stochastic algorithms
- The performance is problem-sample dependent

This part of the lecture is about statistical testing and the exercise today deals with this subject. Probably the most important exercise in the course.

Parameter Tuning I

Now comes the time when you will hate that there are so many parameters in the meta-heuristics. We **cannot** evaluate the performance of an algorithm without properly tuning the parameters.

Parameter Tuning Cookbook

1. Decide how long time the algorithm should run
2. Make a list of the parameters
3. Make a list of test values for each parameter
4. Select a small set of diverse data-samples
5. Select a sample number S (small 3-10)
6. Run the meta-heuristic for each parameter setting a S times for each data-sample.
7. Calculate the average performance for each dataset, for each parameter setting.
8. Calculate the average performance for each parameter setting.

Algorithm Running Time

Because time is usually given by external requirements, and there always is a time-efficiency trade off, we will always use the time available. Hence, for this assignment you should select the time you want to allow your algorithm to run.

List of parameters

Make a list of **all** parameters (remember the hidden parameters). For Simulated Annealing this could be:

- Initial temperature
- Temperature decrease rate α

Notice we do not need a “finish temperature” that is given by the.

Make a list of test values

For each parameter select “intelligently” some values, ex:

- Initial temperature: $\{10, 50, 100\}$
- Temperature decrease rate: $\{0.85, 0.9, 0.95, 0.98, 0.99\}$

Select a small set of diverse data-samples

Select a few of the data-samples for parameter tuning:

1. Covering all sizes
2. .. and all problem characteristics.

This set of data samples we will call the parameter tuning data samples.

Select a sample number

Your algorithms are **stochastic** so we have to calculate average performance, for each parameter combination, for each parameter tuning data sample. We do that by running our algorithm a number of times and calculate average performance.

Calculate Average

Calculate Average performance and standard deviation for each parameter setting for each parameter tuning data sample. Select as parameters the best parameters for the algorithm based on the results on the whole parameter tuning data set.

Test the algorithms

Now, given the best performing parameters, compare the two algorithms performance on the remaining data samples.

Comparing Algorithms

Only properly tuned algorithms running the same time can be compared.