v0.05: Seth Gilbert
v0.5: Steven Halim

CS4234
Optimiz(s)ation Algorithms

L10 – Meta-heuristics & Its Performance

Many parts of the material is based on slides provided with the book 'Stochastic Local Search: Foundations and Applications' by Holger H. Hoos and Thomas Stützle (Morgan Kaufmann, 2004) – see http://www.sls-book.net for further information.

Outline

- Prologue: Randomization
- Meta-heuristics
  1. Simulated Annealing (SA) – simpler one
  2. Tabu Search (TS) – simpler one
  3. Iterated Local Search (ILS) – hybrid SLS
  4. Evolutionary Algorithms (EA) – population based
- Performance and Tuning of Meta-heuristics

Review & Limitation of Hill Climbing

Review of **Hill Climbing** a.k.a. **Steepest Descent** a.k.a. **Iterative Improvement (II)**:

- Start from a candidate solution, locally optimize it based on some neighbourhood relation, stop after reaching a Local Optima w.r.t that neighbourhood
- Problems:
  - Too easy to get stuck in a Local Optima (LO)
  - Terminates too fast even if we have more time resource
- Restarting from random candidate solution (the easiest LO escape mechanism) may not be a good way for all cases as all search history is 'lost'

Randomized Iterative Improvement

Key idea: In each search step, with a fixed probability, perform an uninformed random walk step instead of an iterative improvement step

Randomised Iterative Improvement (RII):

determine initial candidate solution s
While termination condition is not satisfied:
  With probability wp:
    choose a neighbour s' of s uniformly at random
  Otherwise:
    choose a neighbour s' of s such that $g(s') < g(s)$ or,
      if no such s' exists, choose s' such that $g(s')$ is minimal
  $s := s'$

Notes about RII

- With such simplistic RII, we do not need to terminate search when a Local Optima is encountered
  - Instead: Bound number of search steps or CPU time from beginning of search or after last improvement
- Probabilistic/stochastic mechanism permits arbitrary long sequences of random walk steps
  - Therefore: When run sufficiently long (e.g. infinitely :O), RII is guaranteed :O to find (optimal) solution to any problem instance with arbitrarily high probability
- A variant of RII has successfully been applied to a certain COP, but generally, RII is very often outperformed by more complex SLS methods

Probabilistic Iterative Improvement

Key idea of PII: Accept worsening steps with probability that depends on respective deterioration in evaluation function value: bigger deterioration ~ smaller probability

Implementation:

- Create function **p(g, s)** that determines probability distribution over neighbors of **s** based on their values under evaluation function **g** and let **step(s)(s') := p(g, s)(s')**

Note:

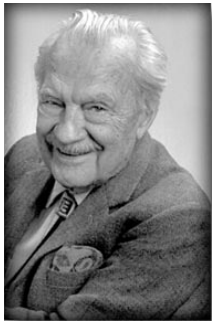
- Behavior of PII crucially depends on choice of **p**
  - Design and Tuning Problem that we will discuss soon
- II and RII are special cases of PII (think about it)

PII for TSP (1)

- **Search space**: Set of all TSP tours
- **Solution set**: TSP tours with the minimum length
- **Neighborhood relation**: 2-exchange neighborhood
- **Initialization**: Pick a random TSP tour
- **Step function**: Next slide
- **Termination**: When exceeding Run Time Limit

PII for TSP (2)

Nicholas Metropolis



- **Step function**: Implemented as 2-stage process:
  1. Select neighbor **s' ∈ N(s)** uniformly at random
  2. Accept as new search position with probability:

  p(s, s', T) := 1, if f(s') ≤ f(s) or // i.e. improve, always take

  := exp((f(s)−f(s')) / T), otherwise, throw coin

  // note that **f(s)-f(s')** is negative

  This is called "Metropolis condition", where temperature parameter **T** controls likelihood of accepting worsening steps

PS: Every occurrence of **bold red text** indicates a tune-able parameter that will somewhat influence the SLS performance

Meta-heuristics

Definition: **Generic** technique or approach or algorithm that is used to guide or control an **underlying** problem-specific heuristic method (e.g. a basic Local Search/Hill Climbing algorithm) *in order to improve its performance and/or robustness*

The next few algorithms that we will discuss fall into this category: **Simulated Annealing** (SA, early 1980), **Tabu Search** (TS, mid 1980), **Iterated Local Search** (ILS, 1990ies), **Evolutionary/Genetic/Memetic Algorithm** (EA/GA/MA, early 1990ies), etc

Simulated Annealing (SA)

Key idea: We vary the temperature parameter **T** (the probability of accepting worsening moves) in Probabilistic Iterative Improvement (PII) according to **annealing schedule** (a.k.a. **cooling schedule**)

Inspired by physical annealing process:

- Candidate solutions ~= states of physical system
- Evaluation function ~= thermodynamic energy
- Globally optimal solutions ~= ground states
- Parameter **T** ~= physical temperature

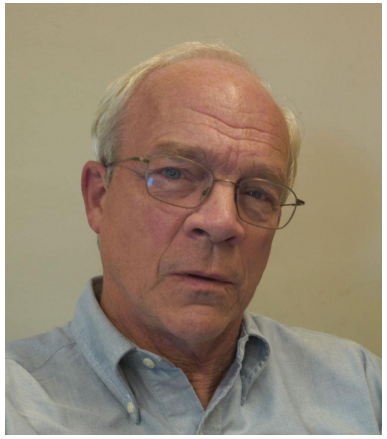
Note: In physical process (e.g., annealing of metals), perfect ground states are achieved by **very (or extremely) slow** lowering of temperature

SA Pseudocode

Simulated Annealing (SA):

determine initial candidate solution s
set initial temperature T according to *annealing schedule*
While termination condition is not satisfied:
  probabilistically choose a neighbour s' of s
    using *proposal mechanism*
  If s' satisfies probabilistic *acceptance criterion* (depending on T):
    $s := s'$
  update T according to *annealing schedule*

Popularized by Scott Kirkpatrick
in early 1980ies



Notes about SA

- 2-stage step function based on
  - **Proposal mechanism** (often uniform random choice from **N(s)**)
  - **Acceptance criterion** (often Metropolis condition, see previous slides)
- **Annealing schedule** is a function that maps run-time **t** onto temperature **T(t)**:
  - Initial temperature T_0
    - May depend on properties of given problem instance
  - Temperature update scheme
    - e.g., geometric cooling: **T := α*T**
  - Number of search steps to be performed at each temperature
    - Often multiple of neighborhood size
- **Termination predicate** is often based on acceptance ratio, i.e., ratio of proposed vs accepted steps

SA for TSP

Extension of previous PII algorithm for TSP, with:

- Proposal mechanism: Uniform random choice from 2-exchange neighborhood
- **Acceptance criterion**: Metropolis condition (always accept improving steps, accept worsening steps with probability **exp[(f(s)-f(s'))/T]**, as in previous few slides)
- **Annealing schedule**: Geometric cooling **T := 0.95 * T** with **n * (n-1)** steps at each temperature (n = number of vertices in given graph), $\mathbf{T_0}$ chosen such that **0.97** of proposed steps are accepted
- **Termination**: After **5** successive temperature values, no improvement in solution quality & acceptance ratio < **2%**

Implementation Details

- Neighborhood pruning (e.g., candidate lists for TSP)
- Greedy initialization (e.g., by using NNH for the TSP)
- Low temperature starts (to prevent good initial candidate solutions from being too easily destroyed by worsening steps)
- Look-up tables for acceptance probabilities: Instead of computing exponential function **exp(Δ/T)** for each step with **Δ := f(s)-f(s')** (expensive!), use precomputed table for range of argument values **Δ/T**

Results of SA

'Convergence' result: Under certain conditions (**extremely** slow cooling), any **sufficiently long** trajectory of SA is guaranteed to end in an optimal solution… [Geman and Geman, 1984; Hajek, 1998]

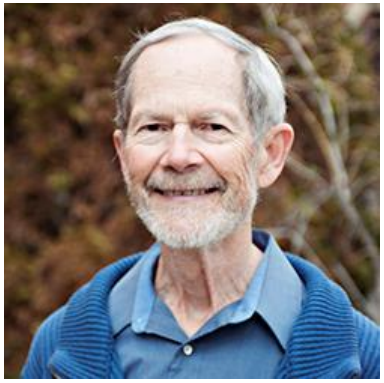
But… important notes:

- Practical relevance of SA for combinatorial problem solving is very limited (impractical nature of necessary conditions and it has too many parameters that have to be configured properly… ☹)
- In combinatorial problem solving, ending in optimal solution is typically unimportant, but finding reasonably good solution quickly is
- PS: Not tried on https://open.kattis.com/problems/tsp yet, any taker?
  - Note that we ONLY have 2s for that problem so we can't wait for 'too long'…

Tabu Search (TS)

Key idea: Use aspects of search history (memory **M**) to escape from local minima

The name Tabu is really from word 'Taboo'



- Inventor: Fred Glover, around mid 1980ies, the one who coined the term 'Meta-heuristics'

Simple Tabu Search:

- Associate tabu attributes with candidate solutions or usually the solution components
- Forbid steps to search positions (or the reuse of solution components) recently visited (used) by underlying iterative best improvement procedure based on tabu attributes

TS Pseudocode

Tabu Search (TS):

determine initial candidate solution s
While *termination criterion* is not satisfied:
    determine set N' of non-tabu neighbours of s
    choose a best improving candidate solution s' in N'
    update tabu attributes based on s'
    $s := s'$

Notes about TS

- Non-tabu search positions in **N(s)** are called **admissible neighbors** of **s**
- After a search step, the current search position *or usually, the solution components* just added/removed from it are declared **tabu** for a fixed number of subsequent search steps (parameter **tabu tenure**)
- Often, an additional **aspiration criterion** is used: this specifies conditions under which tabu status may be *overridden* (e.g., if considered step leads to improvement in incumbent solution)

Note about Tabu Tenure

Performance of Tabu Search depends crucially on the setting of one important parameter **tabu tenure TT**:

- TT too low ⇒ search stagnates
  - due to inability to escape from local minima (back to square one…)
- TT too high ⇒ search becomes ineffective
  - due to overly restricted search path (admissible neighborhood is too small)

Advanced TS methods:

- Robust Tabu Search [Taillard, 1991]:
  - Repeatedly (but **randomly**) choose **TT** from a given interval [lo..hi]
  - Also: Force specific steps that have not been made for a long time
- Reactive Tabu Search [Battiti and Tecchiolli, 1994]:
  - Dynamically adjust **TT** during search
  - Also: Use escape mechanism to overcome stagnation (**randomization**)

More TS-related Strategies

Further improvements can be achieved by using *intermediate-term* or *long-term memory* to achieve additional *intensification* or *diversification*

Examples:

- Occasionally backtrack to *elite candidate solutions*, i.e., high-quality search positions encountered earlier in the search and clear all associated tabu attributes
- Freeze certain solution components and keep them fixed for long(er) periods of the search
- Occasionally force rarely used solution components to be introduced into current candidate solution
- Extend evaluation function to capture frequency of use of candidate solutions or solution components

Result of TS

Tabu Search-related algorithms are state of the art for solving several Combinatorial Optimization Problems, including Steven's version for LABS Problem [Halim et al., 2008] – hopefully still state of the art?

- Yes, Tabu Search is Steven's old favorite SLS algorithm

Crucial factors for successful TS applications:

- Choice of neighborhood relation (standard)
- Super important: Efficient evaluation of candidate solutions (caching and incremental updating mechanisms)
- Specific for TS: Setup of Tabu Tenure, what to be set as Tabu (and also the setting of Aspiration Criteria)

Hybrid SLS Methods

Combination of 'simpler' SLS methods (discussed earlier) often, *but not always*, yields substantial performance improvements

Simple examples:

- Commonly used restart mechanisms can be seen as hybridizations with Uninformed Random Picking
- Iterative Improvement (II/Hill Climbing) + Uninformed Random Walk = Randomized Iterative Improvement (RII)

Iterated Local Search (ILS)

Key Idea: Use two types of SLS steps:

- **Subsidiary local search** steps for reaching local optima as efficiently as possible (intensification)
- **Perturbation steps** for effectively escaping from local optima (diversification)

Also: Use **acceptance criterion** to control diversification vs intensification behavior

ILS Pseudocode

Iterated Local Search (ILS):

determine initial candidate solution s
perform *subsidiary local search* on s
While termination criterion is not satisfied:
    $r := s$
    perform *perturbation* on s
    perform *subsidiary local search* on s
    based on *acceptance criterion*,
        keep s or revert to $s := r$

Notes about ILS

- **Subsidiary local search** results in a Local Optima
- ILS trajectories can be seen as walks in the space of Local Optima of the given evaluation function
  - We will visualize this in the next lecture…
- **Perturbation mechanism** and **acceptance criterion** may use aspects of search history (i.e., access some form of limited memory **M**)
- In a high-performance ILS algorithm, **subsidiary local search**, **perturbation mechanism**, and **acceptance criterion** need to complement each other well

Subsidiary local search

- More effective **subsidiary local search** procedures (usually) lead to better ILS performance
  - Example: 2-opt vs 3-opt vs LK for TSP
  - At the expense of longer run time (per iteration)
    - Problematic if we only have limited run time
- Often, **subsidiary local search** used is the simple Iterative Improvement algorithm (i.e. Hill Climbing), but more sophisticated (and slower per iteration) SLS methods can be used (e.g., Tabu Search)

Perturbation Mechanism (1)

- Needs to be chosen such that its effect cannot be easily undone by subsequent local search phase
  - Often achieved by search steps in larger neighborhood
  - Example: local search = 2-exchange, perturbation = 4-exchange steps in ILS for TSP
- The **perturbation mechanism** may consist of one or more perturbation steps

Perturbation Mechanism (2)

- Weak perturbation ⇒ short subsequent **subsidiary local search** phase but incurs risk of revisiting current Local Optima (back to square one)
- Strong perturbation (the strongest being total random restart) ⇒ more effective escape from Local Optima but may have similar drawbacks as random restart
- Advanced ILS algorithms may change nature and/or **perturbation strength** *adaptively* during search

Acceptance Criteria

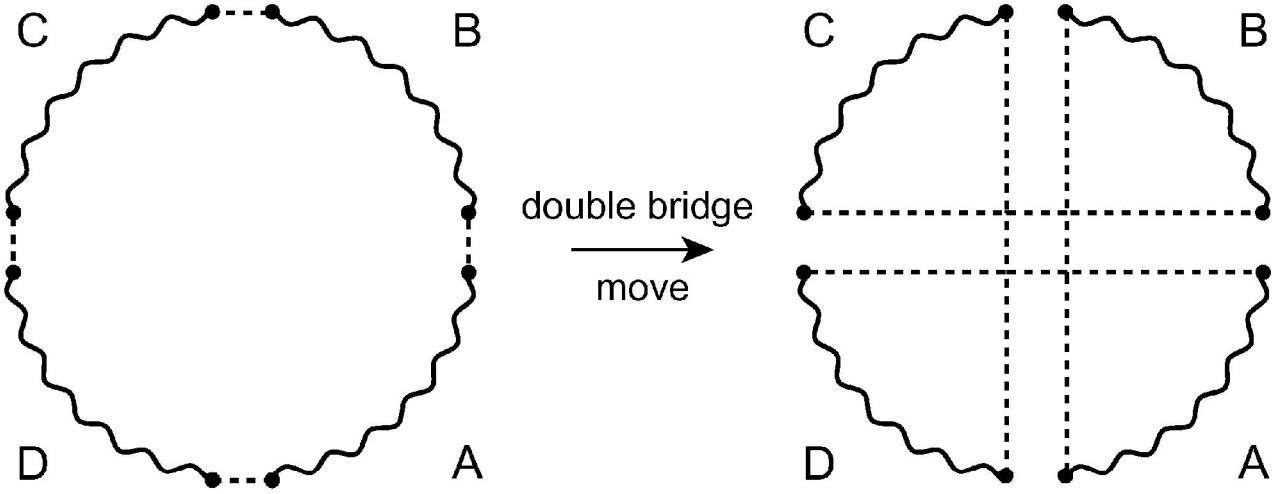
- Always accept **the better of the two** candidate solutions ⇒ ILS performs Iterative Improvement in the space of LO reached by subsidiary local search
- Always accept **the more recent of the two** candidate solutions ⇒ ILS performs random walk in the space of LO reached by subsidiary local search
- Intermediate: Select between the two candidate solutions based on the **Metropolis criterion**
- Advanced **acceptance criteria** take into account search history, e.g., by occasionally reverting to incumbent solution

ILS for TSP (1)

- Given: TSP instance **G** of size **N** points
- Search space: (**N**-1)! TSP tours in **G**
- **Perturbation mechanism**: Use 4-exchange neighborhood (details in the next slide)
- **Subsidiary local search**: 2-exchange Hill Climbing as discussed in previous Lecture 9
- **Acceptance criterion**: Always return the better of the two given candidate tours

ILS for TSP (2)

- **Perturbation mechanism**: 'double-bridge move' = particular 4-exchange step:



- Note:
  - Cannot be directly reversed by a sequence of 2-exchange steps
  - Empirically shown to be effective independent of instance size

Results of ILS

- ILS algorithms are typically rather easy to implement (especially if existing implementation of subsidiary simple SLS algorithms are already done)
  - But not so easy to get the pieces right, as ILS involves (much more) design decisions and parameters than the simpler SLS algorithms
- ILS algorithms achieve state-of-the-art performance on several Combinatorial Optimization Problems, including the TSP :O…
  - Yes, ILS is Steven's second favorite SLS algorithm…

Population-based SLS Methods

SLS methods/algorithms (or Meta-heuristics) discussed so far manipulate one candidate solution of given problem instance in each search step

Straightforward extension: Use population (i.e., a set) of candidate solutions instead

Note:

- The use of populations provides a generic way to achieve search diversification
- Population-based SLS methods fit into the general definition from earlier Lecture 9 by treating sets of candidate solutions as search positions

Evolutionary Algorithm (EA)

Also known as Genetic Algorithm (GA)

Key idea: Iteratively apply genetic operators **mutation**, **recombination**, **selection** to a population of candidate solutions

Inspired by simple model of biological evolution:

- Mutation introduces random variation in the genetic material of individuals
- Recombination of genetic material during sexual reproduction produces offspring that combines features inherited from both parents
- Differences in evolutionary fitness lead selection of genetic traits ('survival of the fittest')
- It 'works' in nature :O…

EA Pseudocode

Evolutionary Algorithm (EA):

determine initial population *sp*

While *termination criterion* is not satisfied:

- generate set *spr* of new candidate solutions by *recombination*
- generate set *spm* of new candidate solutions from *spr* and *sp* by *mutation*
- *select* new population *sp* from candidate solutions in *sp*, *spr*, and *spm*

Memetic Algorithm (MA)

"Small" Problem: Pure Evolutionary Algorithms (EA) often lack capability of sufficient search intensification

- Very annoying to see a member of population 'near' local optima (of a simple neighborhood) but pure EA cannot find it easily

Solution: Apply **subsidiary local search** (similar idea as ILS) after **initialization**, **mutation**, and **recombination**

Such variants of Evolutionary Algorithms are called **Memetic Algorithms (MA)** or Genetic Local Search

- Popularized by Pablo Moscato in late 1980/early 1990

MA Pseudocode

Memetic Algorithm (MA):

determine initial population *sp*

perform *subsidiary local search* on *sp*

While *termination criterion* is not satisfied:

- generate set *spr* of new candidate solutions by *recombination*
- perform *subsidiary local search* on *spr*
- generate set *spm* of new candidate solutions from *spr* and *sp* by *mutation*
- perform *subsidiary local search* on *spm*
- *select* new population *sp* from candidate solutions in *sp*, *spr*, and *spm*

Initialization and Recombination

- **Initialisation**
  - Often: Independent, uninformed random picking from given search space, e.g. set of random TSP tours
    - But can also use multiple runs of construction heuristic, e.g. variations of Greedy Nearest Neighbors for TSP
- **Recombination**
  - Typically repeatedly selects a set of parents from current population and generates offspring candidate solutions from these by means of recombination operator
  - Recombination operators are generally based on linear representation of candidate solutions and piece together offspring from fragments of parents, lots of options for TSP
    - Question: Can delta evaluations be applied here?

Mutation

- Mutation
  - Goal: Introduce relatively small perturbations in candidate solutions in current population + offspring obtained from **recombination**
  - Typically, perturbations are applied stochastically and independently to each candidate solution; amount of perturbation is controlled by **mutation rate**
  - Can also use subsidiary selection function to determine subset of candidate solutions to which mutation is applied
  - In the past, the role of mutation (as compared to recombination) in high-performance evolutionary algorithms has been often underestimated [Back, 1996]

Selection (I)

- **Selection**
  - Determines population for next cycle (generation) of the algorithm by selecting individual candidate solutions from current population + new candidate solutions obtained from recombination, mutation (+ subsidiary local search)
  - Goal: Obtain population of high-quality solutions while maintaining population diversity
  - Selection is based on evaluation function (fitness) of candidate solutions such that better candidate solutions have a higher chance of 'surviving' the selection process

Selection (II)

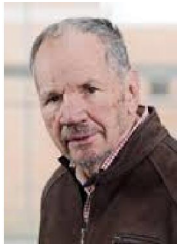
- **Selection** (continued)
  - Many **selection schemes** involve probabilistic choices, e.g., roulette wheel selection, where the probability of selecting any candidate solution **s** is proportional to its fitness value, **g(s)**
  - It is often beneficial to use **elitist selection strategies**, which ensure that the best candidate solutions are always selected (that is, keep the alpha male in the population…)

Subsidiary Local Search

- Often useful and necessary for obtaining high-quality candidate solutions
  - We will see one later in T09: MA for LABS Problem…
- Typically consists of selecting some or all individuals in the given population and applying an iterative improvement procedure (or another more sophisticated SLS algorithm) to each element of this set independently
  - Again, at the cost of runtime…

Other Meta-heuristics

- Simpler ones:
  - Greedy Randomized Adaptive Search Procedure (GRASP)
  - Variable Neighborhood Search
    - Hansen et al, late 1990ies



- Population-based:
  - Ant Colony Optimization (ACO)
    - Dorigo et al
    - Have good result for TSP :O

  - Particle Swarm Optimization (PSO)
  - But read https://en.wikipedia.org/wiki/List_of_metaphor-based_metaheuristics#Criticism

Parameter/Design Settings

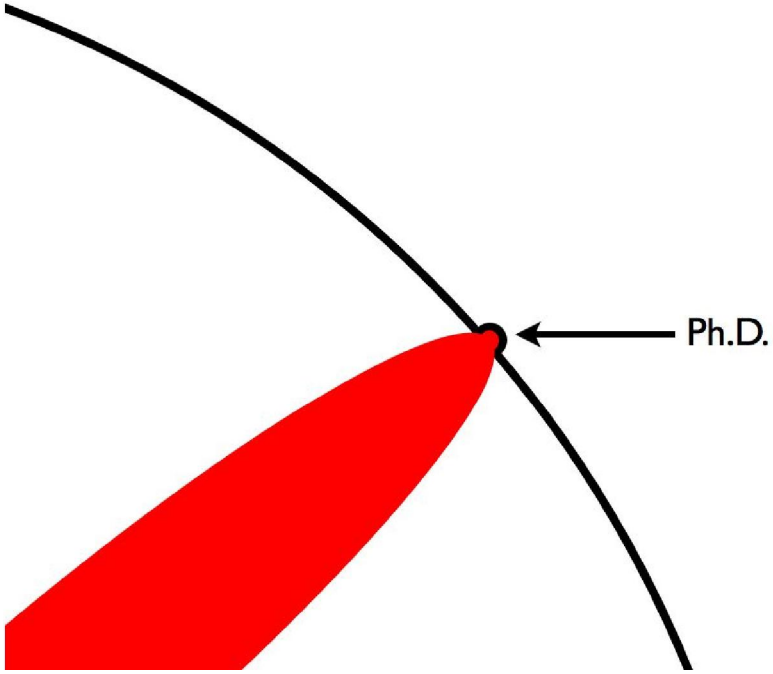
- Throughout the lecture, you must have encountered many **bold red text** that indicates customized components and/or parameter values
  - Setting them correctly (or wrongly) influences the SLS algorithm performance
  - You cannot just pick a textbook (or a paper) settings and apply them blindly towards your new (NP-)hard COP…

What should you do?

- You need to **customize** them towards your needs
  - Do you have limited runtime or do you aim for the best possible result (runtime is not really a problem)?
    - e.g. for Mini Project part 1… you don't have that much time (2s)
  - Do you chase peak performance (repeat the experiments many time) or SLS with low variability (i.e. more robust)?
    - e.g. for Mini Project part 2 ($66 \leq N \leq 100$)…
      you are chasing for peak performance and report only that peak…
  - Do you have to solve ALL instances of (NP-)hard COP or you are given a set of specific instances (still proven to be NP-hard, so there is no polynomial time solution yet…)?
    - You can analyze that set of instances, who know they have exploitable properties?

SLS Design and Tuning Problem

- The previous slide was the motivation of my 5 years of research (2004-2009) that gave me my PhD...



http://matt.might.net/articles/phd-school-in-pictures/

Which Meta-heuristic to use?

- My (current) answer: It depends…
- Usually the answer is based on what one has been exposed to (and now familiar with…)
- Usually one will only switch or experiment with other Meta-heuristics when one have tried his/her favorite (for quite some time) but it cannot get his/her desired objective…

Summary

- I have introduced a few well known Meta-heuristics (or advanced SLS algorithms): Simulated Annealing (SA), Tabu Search (TS), Iterated Local Search (ILS), and Evolutionary/Genetic/Memetic Algorithm (EA/GA/MA)
- Each of them have their own strengths and weaknesses and there is no clear winner on what to use for a given, new, non textbook (NP-)hard COP
  - Some Meta-heuristics are easier to tune than the others (next lecture)
  - Some are inherently faster (to get good solutions) than the others
  - Usually the choice of certain Meta-heuristic is down to the expertise and familiarity of the problem solver :O…
- Up next: Summary of 5 years of Steven's PhD research
  - SLS Design and Tuning Problem (and Steven's thesis on how to deal with it)