Chapter 00: Introduction

(Version of 14th January 2024)

Pierre Flener

Department of Information Technology
Computing Science Division
Uppsala University
Sweden

Course 1DL481:
Algorithms and Data Structures 3 (AD3)

Outline

Outline

1. **The End of Course AD2**

2. **Combinatorial Optimisation**
   - Constraint Problems
   - Solving Technologies
   - Modelling
   - Solving

3. **Course AD3**
   - Contents
   - New Concepts
   - Learning Outcomes
   - Organisation

In a decision problem we seek a 'yes' / 'no' answer to an existence question. An instance of a problem is given by its input data.

Example (Travelling salesperson: Decision TSP)

Given a budget b and a map with n cities, **is** there a route visiting each city exactly once, returning to the starting city, and costing at most b **?**

A decision problem R is:

- in NP if a witness to a 'yes' instance is checkable in time polynomial in the instance size: checking is in P;
- NP-complete if in NP and there is a reduction from each problem Q in NP, polytime converting any instance of Q into a same-answer instance of R.

It is believed that NP-complete problems are intractable (or: hard), requiring non-polynomial time to solve **exactly**.

Example

TSP is NP-complete as a witness is checkable in $\mathcal{O}(n)$ time and the NP-complete Hamiltonian-Cycle problem reduces to it.

In a satisfaction problem we seek a witness for a ‘yes’ answer.

Example (Satisfaction TSP)

Given a budget b and a map with n cities, **find** a route visiting each city exactly once, returning to the starting city, and costing at most b.

In an optimisation problem we seek an optimal witness, according to some objective function, for a ‘yes’ answer.

Example (Optimisation TSP)

Given a map with n cities, **find** a **cheapest** route visiting each city exactly once and returning to the starting city.

In addition to decision problems that are at least as hard as every NP problem (as every NP problem reduces to them), satisfaction and optimisation problems with NP-complete decision versions are often also said to be NP-hard: they are unlikely to be easier than their decision versions.

What Now?

Several courses at Uppsala University teach techniques for addressing NP-hard optimisation and satisfaction problems:

1TD184	Continuous Optimisation	(period 2)
1DL451	Modelling for Combinatorial Optimisation	(period 1)
1DL442	Combinatorial Optimisation and Constraint Programming	(period 1+2)
1DL481	Algorithms and Data Structures 3	(period 3)

NP-hardness is not where the fun ends, but where it begins!

Example (Optimisation TSP over n cities)

A brute-force algorithm evaluates all $n!$ candidate routes:

- A computer of today evaluates 10^6 routes / second:

n	time
11	40 seconds
14	1 day
18	203 years
20	77k years

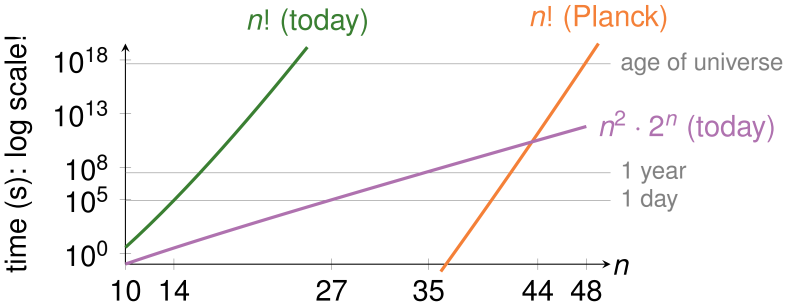
- Planck time is the shortest useful interval: $\approx 5.4 \cdot 10^{-44}$ seconds; a Planck computer would evaluate $1.8 \cdot 10^{43}$ routes / second:

n	time
37	0.7 seconds
41	20 days
48	$1.5 \cdot$ age of universe

The dynamic program by Bellman-Held-Karp "only" takes $\mathcal{O}(n^2 \cdot 2^n)$ time: a computer of today takes a day for $n = 27$, a year for $n = 35$, the age of the universe for $n = 67$, and beats the $\mathcal{O}(n!)$ algo on Planck computer for $n \geq 44$.

Intelligent Search upon NP-Hardness

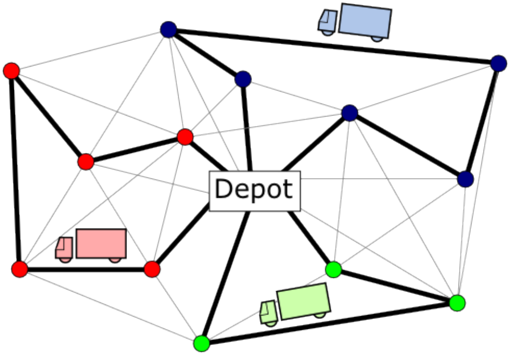
Do not give up but try to stay ahead of the curve:
there is an instance size until which an **exact** algorithm is fast enough!



Concorde TSP Solver beats Bellman-Held-Karp **exact** algorithm: it uses **local search** & **approximation** algos, but sometimes proves exactness of its optima. The largest instance solved exactly, in 136 CPU years in 2006, has $n = 85900$.

Outline

Optimisation



Optimisation is a science of service:
to scientists, to engineers, to artists, and to society.

Outline

Example (Agricultural experiment design)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley							
corn							
millet							
oats							
rye							
spelt							
wheat							

Constraints to be **satisfied**:

1. Equal growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, balance 1.

Example (Agricultural experiment design)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

Constraints to be **satisfied**:

1. Equal growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, balance 1.

UPPSALA UNIVERSITET

Example (Doctor rostering)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Doctor A							
Doctor B							
Doctor C							
Doctor D							
Doctor E							

Constraints to be **satisfied**:

1. #on-call doctors / day $= 1$
2. #operating doctors / weekday ≤ 2
3. #operating doctors / week ≥ 7
4. #appointed doctors / week ≥ 4
5. day off after operation day
6. ...

Objective function to be **minimised**: Cost: ...

Example (Doctor rostering)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Doctor A	call	none	oper	none	oper	none	none
Doctor B	appt	call	none	oper	none	none	call
Doctor C	oper	none	call	appt	appt	call	none
Doctor D	appt	oper	none	call	oper	none	none
Doctor E	oper	none	oper	none	call	none	none

Constraints to be **satisfied**:

1. #on-call doctors / day $= 1$
2. #operating doctors / weekday ≤ 2
3. #operating doctors / week ≥ 7
4. #appointed doctors / week ≥ 4
5. day off after operation day
6. ...

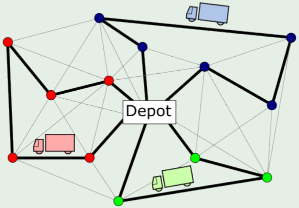
Objective function to be **minimised**: Cost: ...

Example (Vehicle routing: parcel delivery)

Given a depot with parcels for clients and a vehicle fleet,
find which vehicle visits which client when.

Constraints to be **satisfied**:

1. All parcels are delivered on time.
2. No vehicle is overloaded.
3. Driver regulations are respected.
4. ...

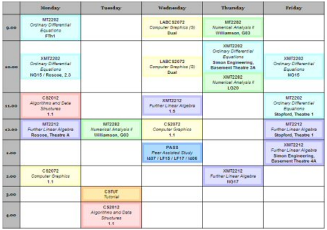


Objective function to be **minimised**:

- Cost: the total fuel consumption and driver salary.

Application Areas

School timetabling



	Monday	Tuesday	Wednesday	Thursday	Friday
9.00	MT2202 *Ordinary Differential Equations* FTh1		LABCS2072 *Computer Graphics (G)* Dual	MT2282 *Numerical Analysis II* Williamson, G03	
10.00	XMT2202 *Ordinary Differential Equations* NG15 / Roscoe, 2.3		LABCS2072 *Computer Graphics (G)* Dual	XMT2202 *Ordinary Differential Equations* Simon Engineering, Basement Theatre 3A; XMT2282 *Numerical Analysis II* LG20	XMT2202 *Ordinary Differential Equations* NG15
11.00	CS2012 *Algorithms and Data Structures* 1.1		XMT2212 *Further Linear Algebra* 1.5		MT2202 *Ordinary Differential Equations* Stopford, Theatre 1
12.00	MT2212 *Further Linear Algebra* Roscoe, Theatre A	MT2282 *Numerical Analysis II* Williamson, G03	CS2072 *Computer Graphics* 1.1		MT2212 *Further Linear Algebra* Stopford, Theatre 1
1.00			PASS *Peer Assisted Study* I407 / LF15 / LF17 / I406		XMT2212 *Further Linear Algebra* Simon Engineering, Basement Theatre 4A
2.00	CS2072 *Computer Graphics* 1.1			XMT2212 *Further Linear Algebra* NG17	
3.00		CSTUT *Tutorial*			
4.00		CS2012 *Algorithms and Data Structures* 1.1			

Sports tournament design

svensk handboll

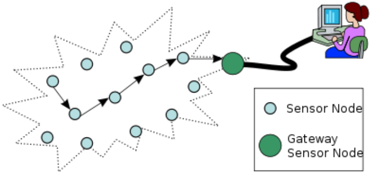
Security: SQL injection?

Container packing

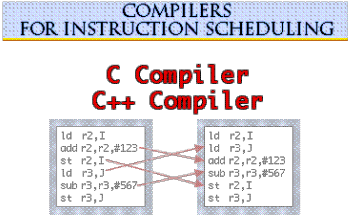
Applications in Programming and Testing

Robot programming

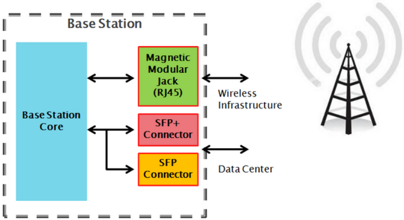
Sensor-net configuration



Compiler design

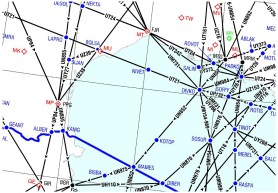


Base-station testing

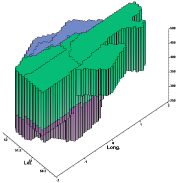


Applications in Air Traffic Management

Demand vs capacity



Airspace sectorisation



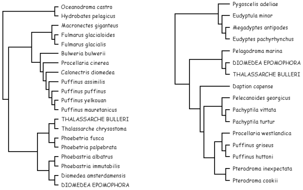
Contingency planning

Flow	Time Span	Hourly Rate
From: Arlanda	00:00 – 09:00	3
To: west, south	09:00 – 18:00	5
	18:00 – 24:00	2
From: Arlanda	00:00 – 12:00	4
To: east, north	12:00 – 24:00	3
...	...	...

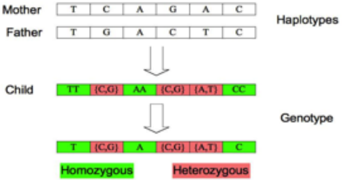
Workload balancing

Applications in Biology and Medicine

Phylogenetic supertree



Haplotype inference



Medical image analysis

Doctor rostering

Definitions

In a constraint problem, values have to be **found** for all the unknowns, called variables (in the mathematical sense; also called decision variables) and ranging over **given** sets, called domains, so that:

- All the given constraints on the decision variables are **satisfied**.
- Optionally: A given objective function on the decision variables has an optimal value: either a **minimal** cost or a **maximal** profit.

A candidate solution to a constraint problem maps each decision variable to a value within its domain; it is:

- feasible if all the constraints are satisfied;
- optimal if the objective function takes an optimal value.

The search space consists of all candidate solutions.
A solution to a satisfaction problem is feasible.
An optimal solution to an optimisation problem is feasible and optimal.

Search spaces are often larger than the universe!

Many important real-life problems are NP-hard or worse: their real-life instances can only be solved **exactly** and fast enough by intelligent search, unless P $=$ NP. ☞ NP-hardness is not where the fun ends, but where it begins!

Outline

A solving technology offers languages, methods, and tools for:

what: **Modelling** constraint problems in a declarative language.

and / or

how: **Solving** constraint problems intelligently:

- Search: Explore the space of candidate solutions.
- Inference: Reduce the space of candidate solutions.
- Relaxation: Exploit solutions to easier problems.

A solver is an off-the-shelf program that takes any model and data as input and tries to solve that problem instance.

Combinatorial (= discrete) optimisation covers satisfaction *and* optimisation problems for variables ranging over *discrete* sets: combinatorial problems.

Outline

Example (Agricultural experiment design, AED)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	✓	✓	✓	–	–	–	–
corn	✓	–	–	✓	✓	–	–
millet	✓	–	–	–	–	✓	✓
oats	–	✓	–	✓	–	✓	–
rye	–	✓	–	–	✓	–	✓
spelt	–	–	✓	✓	–	–	✓
wheat	–	–	✓	–	✓	✓	–

Constraints to be **satisfied**:

1. Equal growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, balance 1.

General term: balanced incomplete block design (BIBD).

Example (Agricultural experiment design, AED)

	plot1	plot2	plot3	plot4	plot5	plot6	plot7
barley	1	1	1	0	0	0	0
corn	1	0	0	1	1	0	0
millet	1	0	0	0	0	1	1
oats	0	1	0	1	0	1	0
rye	0	1	0	0	1	0	1
spelt	0	0	1	1	0	0	1
wheat	0	0	1	0	1	1	0

Constraints to be **satisfied**:

1. Equal growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

Instance: 7 plots, 7 grains, 3 grains/plot, 3 plots/grain, balance 1.

General term: balanced incomplete block design (BIBD).

In a BIBD, the plots are called blocks and the grains are called varieties:

Example (BIBD *integer* model: ✓ $\rightsquigarrow$ 1 and – $\rightsquigarrow$ 0)

```
enum Varieties; enum Blocks;
int: blockSize; int: sampleSize; int: balance;
array[Varieties,Blocks] of var 0..1: BIBD; % BIBD[v,b]=1 iff v is in b
solve satisfy;
constraint forall(b in Blocks)    (blockSize  = count(BIBD[..,b], 1));
constraint forall(v in Varieties) (sampleSize = count(BIBD[v,..], 1));
constraint forall(v, w in Varieties where v < w)
  (balance = count([BIBD[v,b]+BIBD[w,b] | b in Blocks], 2));
```

Example (Instance data for our AED)

```
Varieties = {barley,...,wheat}; Blocks = {plot1,...,plot7};
blockSize = 3; sampleSize = 3; balance = 1;
```

Reconsider the model fragment:

```
constraint forall(v in Varieties)(sampleSize = count(BIBD[v,..], 1));
```

This constraint is declarative (and by the way not within linear algebra), so read it using only the verb "to be" or synonyms thereof:

> *for all varieties* `v`,
> *the count of occurrences of* `1` *in row* `v` *of* `BIBD`
> *must equal* `sampleSize`

The constraint is not procedural:

> *for all varieties* `v`,
> *we first count the occurrences of* `1` *in row* `v`
> *and then check if that count equals* `sampleSize`

The latter reading is appropriate for solution checking, but solution finding performs no such procedural counting.

Example (Idea for another BIBD model)

barley	{plot1, plot2, plot3}
corn	{plot1, plot4, plot5}
millet	{plot1, plot6, plot7}
oats	{plot2, plot4, plot6}
rye	{plot2, plot5, plot7}
spelt	{plot3, plot4, plot7}
wheat	{plot3, plot5, plot6}

Constraints to be **satisfied**:

1. Equal growth load: Every plot grows 3 grains.
2. Equal sample size: Every grain is grown in 3 plots.
3. Balance: Every grain pair is grown in 1 common plot.

Example (BIBD *set* model: a block *set* per variety)

```
-3 enum Varieties; enum Blocks;
-2 int: blockSize; int: sampleSize; int: balance;
-1 array[Varieties] of var set of Blocks: BIBD; % BIBD[v] = blocks for v
 0 solve satisfy;
 1 constraint forall(b in Blocks)
     (blockSize  = sum(v in Varieties)(b in BIBD[v]));
 2 constraint forall(v in Varieties)
     (sampleSize = card(BIBD[v]));
 3 constraint forall(v, w in Varieties where v < w)
     (balance    = card(BIBD[v] intersect BIBD[w]));
```

Example (Instance data for our AED)

```
-3 Varieties = {barley,...,wheat}; Blocks = {plot1,...,plot7};
-2 blockSize = 3; sampleSize = 3; balance = 1;
```

Example (Doctor rostering)

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Doctor A	call	none	oper	none	oper	none	none
Doctor B	appt	call	none	oper	none	none	call
Doctor C	oper	none	call	appt	appt	call	none
Doctor D	appt	oper	none	call	oper	none	none
Doctor E	oper	none	oper	none	call	none	none

Constraints to be **satisfied**:

1. #on-call doctors / day $= 1$
2. #operating doctors / weekday ≤ 2
3. #operating doctors / week ≥ 7
4. #appointed doctors / week ≥ 4
5. day off after operation day
6. ...

Objective function to be **minimised**: Cost: ...

Example (Doctor rostering)

```
set of int: Days;  % d mod 7 = 1 iff d is a Monday
enum Doctors;
enum ShiftTypes = {appt, call, oper, none};
% Roster[i,j] = shift type of Dr i on day j:
array[Doctors,Days] of var ShiftTypes: Roster;
solve minimize ...;  % plug in an objective function
constraint forall(d in Days)(count(Roster[..,d],call) = 1);
constraint forall(d in Days where d mod 7 in 1..5)
  (count(Roster[..,d],oper) <= 2);
constraint count(Roster,oper) >= 7;
constraint count(Roster,appt) >= 4;
constraint forall(d in Doctors)
  (regular(Roster[d,..],"((oper none) | appt | call | none)*"));
...  % other constraints
```

Example (Instance data for our small hospital unit)

```
Days = 1..7;
Doctors = {Dr_A, Dr_B, Dr_C, Dr_D, Dr_E};
```

Example (Sudoku)

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

8	1	2	7	5	3	6	4	9
9	4	3	6	8	2	1	7	5
6	7	5	4	9	1	2	8	3
1	5	4	2	3	7	8	9	6
3	6	9	8	4	5	7	2	1
2	8	7	1	6	9	5	3	4
5	2	1	9	7	4	3	6	8
4	3	8	5	2	6	9	1	7
7	9	6	3	1	8	4	5	2

```
array[1..9,1..9] of var 1..9: Sudoku;
...   % load the hints
solve satisfy;
constraint forall(row in 1..9)(all_different(Sudoku[row,..]));
constraint forall(col in 1..9)(all_different(Sudoku[..,col]));
constraint forall(i,j in {0,3,6})
  (all_different(Sudoku[i+1..i+3,j+1..j+3]));
```

Modelling Languages

The following fully declarative modelling languages are powerful enough to encode NP-hard problems:

- Mixed integer programming (MIP): **satisfy** a set of linear equalities ($=$) and inequalities ($<$, $\leq$, $\geq$, $>$), but not disequalities ($\neq$), over real-number decision variables and integer decision variables weighted by real-number constants, such that a linear objective function is **optimised**.
- Boolean satisfiability solving (SAT): **satisfy** a set of disjunctions of possibly negated Boolean decision variables.
- SAT modulo theories (SMT) and constraint programming (CP) do not have such small standardised low-level modelling languages, but enable the higher level of the previous sample models.
  ☞ In course 1DL451: Modelling, we use such higher-level models in order to drive CP, MIP, SAT, SMT, … solvers.

UPPSALA UNIVERSITET

Outline

Examples (Solving technologies)

With general-purpose solvers, taking model and data as input:

- (Mixed) integer linear programming (IP and MIP)
- Boolean satisfiability (SAT)
- SAT (resp. optimisation) modulo theories (SMT and OMT)
- Constraint programming (CP)
- ...
- Hybrid technologies (LCG $=$ CP $+$ SAT, ...)

Methodologies, *usually without* modelling and solvers:

- Dynamic programming (DP)
- Greedy algorithms
- Approximation algorithms
- Stochastic local search (SLS)
- ...

Examples (Solving technologies)

With general-purpose solvers, taking model and data as input:

- (Mixed) integer linear programming (IP and MIP) — in AD3
- Boolean satisfiability (SAT) — in AD3
- SAT (resp. optimisation) modulo theories (SMT and OMT) — SMT in AD3
- Constraint programming (CP) — in 1DL442: COCP
- ...
- Hybrid technologies (LCG $=$ CP $+$ SAT, ...)

Methodologies, *usually without* modelling and solvers:

- Dynamic programming (DP) — in 1DL231: AD2
- Greedy algorithms — in 1DL231: AD2
- Approximation algorithms — in AD3
- Stochastic local search (SLS) — in AD3
- ...

Solvers

- Black-box solvers (for SAT, SMT, OMT, IP, MIP, . . .)
  have general-purpose search + inference + relaxation
  that is difficult to influence by the modeller.

  ☞ AD3

- Glass-box solvers (for CP, LCG, . . .)
  have general-purpose search + inference + relaxation
  that is easy to influence, if desired, by the modeller.

  ☞ 1DL442: COCP

- Special-purpose solvers (for TSP, . . .)
  exist for pure problems (that is: problems without side constraints).

Correctness Is Not Enough for Models

Modelling is an Art

There are good and bad models for each constraint problem:

☞ AD3 and 1DL451: Modelling

- Different models of a problem may take different time on the same solver for the same instance.
- Different models of a problem may scale differently on the same solver for instances of growing size.
- Different solvers may take different time on the same model for the same instance.

Good modellers are worth their weight in gold!

Use solvers: based on decades of cutting-edge research, they are very hard to beat on **exact** solving.

Outline

Outline

Common Thread: Coping with NP-Hardness

1. Mixed integer programming (MIP)
2. Stochastic local search (SLS)
3. Amortised analysis (CLRS4: Chapter 16)
4. Probabilistic analysis (Chapter 5)
5. Randomised algorithms: (Chapter 5)
   universal hashing, . . . (Section 11.3.4)
6. Proving NP-completeness by reduction (Chapter 34)
7. Approximation algorithms (Chapter 35)
8. Boolean satisfiability (SAT)
9. SAT modulo theories (SMT)

CLRS4 Textbook:

Introduction to Algorithms (4th edition) (errata).
T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein.
The MIT Press, 2022.

Outline

In a probabilistic algorithm analysis, we use probability theory: knowing or assuming the distribution of the inputs, we compute the average-case time (as opposed to the worst-case time) of a deterministic algorithm.

Example

The brute-force string matching algorithm for finding all occurrences of a pattern P of length m within a text T of length $n \geq m$ takes $\mathcal{O}(n - m + 1)$ time on average when P and T are random strings, but this is a completely unreasonable assumption. (Chapter 32 in CLRS4)

Probabilistic analysis helps gain insight into a problem and helps design an efficient algorithm for it, when we have a reasonable assumption on the distribution of the inputs.

A randomised algorithm (as opposed to a deterministic algorithm) itself makes random choices, independently of the actual distribution of the inputs. We refer to the time of a randomised algorithm as expected time (not: average time).

Examples

- A randomised algorithm by Karger-Klein-Tarjan (1993) computes in $\mathcal{O}(V + E)$ expected time a minimum spanning tree (MST) of a connected undirected graph with vertex set V and edge set E. (Chapter 21)
- A randomised algorithm computes in $\mathcal{O}(m)$ expected time a prime number larger than m for fingerprinting in the Rabin-Karp string matcher. (Chapter 32)

Many randomised algorithms have no worst-case input!

In an amortised analysis, we compute the worst-case time of a *chain* of data-structure operations, and average it over the operations. We refer to this time as an amortised time (as opposed to an average-case time, as no probability is used here, and to the possibly non-tight worst-case time).

Examples

- A chain of m find-and-compress-paths or union-by-rank operations on disjoint sets of n items takes $\mathcal{O}(m \cdot \lg^* n)$ time, where $\lg^* n \leq 5$ in practice. (Chapter 19)
- In a Fibonacci heap of n items, extracting a minimum takes $\mathcal{O}(\lg n)$ amortised time, and decreasing a key takes $\mathcal{O}(1)$ amortised time. (online chapter; not in AD2)
- Prim's MST algorithm takes at worst $\mathcal{O}(E + V \lg V)$ time when using a Fibonacci heap. (Chapter 21)

Dealing in polynomial time with (instances of) optimisation problems where brute-force or exact solving is too costly:

- A greedy algorithm builds a feasible solution decision variable by decision variable, making locally optimal choices in the hope of reaching an optimal solution. Greedy algorithms build either provably optimal solutions (for example, Prim's MST algorithm and Dijkstra's single-source shortest paths algorithm) or at-best optimal solutions.
- A local search algorithm repairs a possibly infeasible candidate solution, by reassigning some decision variables at every iteration, until an allocated resource (such as an iteration count or a time budget) is exhausted, in the hope of reaching a feasible or even optimal solution.
- An approximation algorithm for an NP-hard optimisation problem builds a feasible solution whose objective value is provably within a known factor of the optimum.

All techniques are orthogonal: there exist randomised local search algorithms, greedy approximation algorithms, etc.

Outline

In order to pass, the student must be able to:

- analyse NP-completeness of an algorithmic problem;
- use advanced algorithm analysis methods, such as amortised analysis and probabilistic analysis;
- use advanced algorithm design methods in order to approach hard algorithmic problems in a pragmatic way, such as by using:
  - randomised algorithms: universal hashing, . . .
  - approximation algorithms
  - stochastic local search: simulated annealing, tabu search, . . .
  - mixed integer programming (MIP)
  - Boolean satisfiability (SAT)
  - SAT modulo theories (SMT)
- present and discuss topics related to the course content orally and in writing with a skill appropriate for the level of education

  ☞ written reports and oral resubmissions!

Outline

Course Organisation and *Suggested* Time Budget

Period 3: January to March, budget = 133.3 hours:

- 12 lectures, including a ***mandatory*** guest lecture, budget = 18 hours
- 2 assignments with 3 help sessions, 1 grading session, 1 solution session per assignment, on 2 problems each, on *non*-exam topics, to be done by student-chosen duo team:
  suggested budget = average of 30 hours / assignment / student (2 credits)
- 1 written ***closed-book*** exam of 3 hours, to be done individually:
  suggested budget = 55 hours (3 credits)
- **Prerequisites:** Algorithms and Data Structures 2 (AD2) (course 1DL231) or equivalent

Examination

Modelling / programming, experimenting, and reporting:

- Mixed integer programming (MIP): Assignment 1
- Stochastic local search (SLS): Assignment 1
- Boolean satisfiability (SAT): Assignment 2
- SAT modulo theories (SMT): Assignment 2

Theory questions, drawn from a published list of potential exam questions:

- Amortised analysis and probabilistic analysis: exam
- Randomised algorithms: exam
- NP-completeness: 50% threshold at exam
- Approximation algorithms: exam

Exam study groups are allowed and encouraged, as the exam is individual.

2 Assignment Cycles of 3 Weeks

Let D_i be the deadline of Assignment i, with $i \in 1..2$:

- $D_i - 16$: publication and all needed material taught: start!
- $D_i - 10$: help session a: attendance strongly recommended!
- $D_i - 7$: help session b: attendance strongly recommended!
- $D_i - 2$: help session c: attendance strongly recommended!
- $D_i \pm 0$: submission, by 13:00 Swedish time, on a Friday
- $\leq D_i + 10$ at 16:00: your initial score $a_{ij} \in 0..5$ points for each Problem j of Assignment i, with $j \in 1..2$
- $D_i + 11$: teamwise oral grading session on *some* Problems j where $a_{ij} \in \{1, 2\}$: possibility of earning 1 extra point for your final score; otherwise final score $=$ initial score
- $D_i + 11 = D_{i+1} - 10$: solution session and help session a

2 Assignment Credits and Overall Influence

Let a_{ij} be your final score on Problem j of Assignment i, with $i, j \in 1..2$:

- **20% threshold:** $\forall i, j \in 1..2 : a_{ij} \geq 20\% \cdot 5 = 1\ (< 3)$
  You may not catastrophically fail on individual problems
- **30% threshold:** $\forall i : a_i = a_{i1} + a_{i2} \geq 30\% \cdot (5 + 5) = 3\ (< 5)$
  You can partially fail on individual problems or entire assignments
- **50% threshold:** $a = a_1 + a_2 \geq 50\% \cdot 2 \cdot (5 + 5) = 10$
  The formula for your assignment grade in 3..5 is at the course homepage
- **Worth going full-blast:** Your assignment score a is meshed with your exam score e in order to determine your overall course grade in 3..5, ***if*** $10 \leq a \leq 20$ ***and*** $10 \leq e \leq 20$:
  see the formula at the course homepage

Caution!

- There is a huge jump from AD2 (or equivalent) — with its mostly (pseudo-)polytime algorithms — to AD3, where only NP-hard problems are considered.
- Correctness is required (unlike in AD2), but very easy to achieve with the help of our provided polytime solution checkers or some revealed optima: we grade for speed (and memory usage).
- Especially the MIP, SAT, and SMT modelling tasks are totally unlike anything most of you have ever seen, and this takes time to wrap one's head around.
- Ease or success with the assignments in AD2 does not imply the same ease or the same level of success with the assignments in AD3: the help sessions are strongly recommended, and there is almost no internet help.

Assignment Rules

Register a team by Sunday 21 January 2024 at 23:59 at Studium:

- **Duo team:** Two consenting teammates sign up
- **Solo team:** Apply to the head teacher, who rarely agrees
- **Random partner?** Request from the helpdesk, else you are bounced

Other considerations:

- **Why (not) like this? Why no email reply?** See FAQ
- **Teammate swapping:** Allowed, but to be declared to the helpdesk
- **Teammate scores may differ** if no-show or passivity at grading session
- **No freeloader:** Implicit honour declaration in reports that each partner can individually explain everything; random checks will be made by us
- **No plagiarism:** Implicit honour declaration in reports; extremely powerful detection tools will be used by us; suspected cases of using or providing must be reported!

How To Communicate by Email or Studium?

- If you have a question about the **lecture material** or **course organisation**, then email the head teacher. An immediate answer will be given right before and after lectures, as well as during their breaks.

- If you have a question about the **assignments** or **infrastructure**, then contact the assistants at a help session or solution session for an immediate answer.

  Short *clarification* questions (that is: *not* about modelling or programming difficulties) that are either emailed (find the address at Studium) or posted (at the Studium discussion) to the **AD3 helpdesk** are answered as soon as possible during working days and hours: almost all the assistants' budgeted time is allocated to grading and to the help, grading, and solution sessions.

What Has Changed Since Last Time?

Changes triggered by the previous course evaluation:

- Deletion or simplification of some tasks of the assignment problems.
- Spacing of dependent lectures: NP-completeness (2); approximation (2).
- Emphasis on the existence of the AD3 helpdesk (by email + discussion).
- Emphasis that exam study groups are allowed and even encouraged.
- Emphasis why exam topics are taught early: maximal preparation time.

What To Do Now?

- Bookmark and read the entire AD3 website, especially the FAQ list.
- Get started on Assignment 1 and have questions ready for the first help session, which is on Tuesday 23 January 2024.
- Register a duo team by Sunday 21 January 2024 at 23:59, possibly upon advertising for a teammate at a course event or the discussion at Studium, and requesting a random partner from the AD3 helpdesk as a last resort.
- Install AMPL (see Studium for a free download with the classroom license) on your own hardware, if you have any.