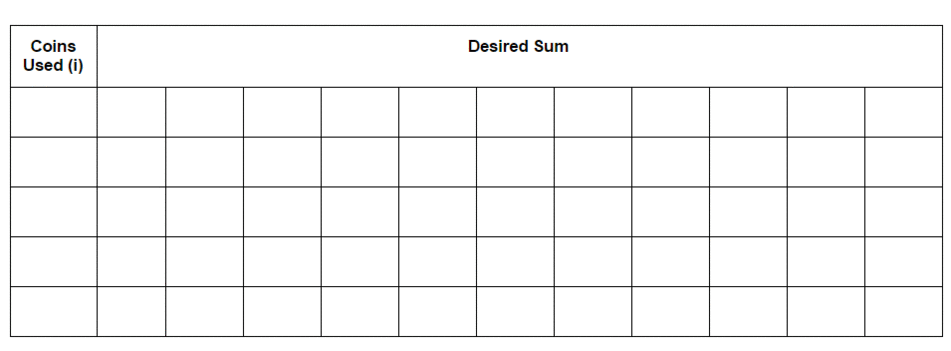
Algorithmic Complexities

From the last section, we were introduced to several different algorithmic approaches to problems, including Divide-and-Conquer, Greedy Strategies, and Dynamic Programming.

The general properties of these approaches can be seen below:

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Fast Algorithm?** |
| Divide and Conquer | Divide the input into very similar subproblems, solve these subproblems, and then combine these results to get an overall solution. This is usually approached via recursion. | Polynomial Time |
| Greedy Strategy | Greedy is an approach which finds the locally optimum at each stage, in an attempt to find the global optimum. This approach doesn’t guarantee a global optimum solution, but usually finds an approximate optimal global solution. | Polynomial Time |
| Dynamic Programming | Dynamic programming is an approach which takes advantage of overlapping subproblems. This approach when remembering the overlaps, leads to much faster processing, since the overlaps are not computed twice. | Can give Fast Algorithms, but can also be used when no fast algorithms are available. |

**Dynamic Programming Intuition:** Although I have already explained the process behind the money problem, the overlapping can be seen as the gray columns. The further down the process you go, the less columns need to be evaluated (less orange means less computations), since the algorithm has “remembered” the gray overlaps.



**Polynomial Time vs Exponential Algorithms:**

|  |  |
| --- | --- |
| **Polynomial Algorithms** | **Exponential Algorithms** |
| All algorithms with asymptotic complexity which can be represented in the form O(1), O(N), O(N2), … O(NM) where M is a constant | All algorithms which have worst case scenarios of O(2N), O(NN) or O(N!) |
| These are considered fast algorithms | These are considered to be very slow algorithms |
| These are called tractable solutions | These are called intractable solutions |
| O(1) O(N) O(N2) O(N4) O(N100)  output_yM8rVn.gif | O(2N) O(NN) O(N!) -- dots  output_IKsaSw.gif |

**Exponential algorithms arise from algorithmic techniques**

One such technique is called exhaustive enumeration. This is checking every single combination available, and checking each against the solution requirement. Imagine trying to find the shortest route visiting 50 cities in the UK (also known as the travelling salesman problem). This is intractable, and no current algorithms can solve this without going super-messy in execution time.

Intractable Problem I: Graph Colouring

* A graph consists of a set of nodes connected by edges. A colouring graph with k colours is an allocation of the colours to the nodes of the graph. There are some rules involved:
* Each node may only have one colour
* Nodes which are connected by an edge must have different colours

The game is to find the minimum number of colours needed to colour this graph. This minimum number is called the chromatic number of the graph.

**Exhaustive Enumeration:**

This is the simplest approach to colouring a graph with k colours:

1. Enumerate all the allocations of k colours to the nodes of the graph
2. Check each allocation to see if it is a valid colouring

If our graph has N nodes, and E edges, then there is kN allocations of k colours to the nodes. The amount of time to check each of the enumerations for correctness if O(E)

Hence the worst case complexity is O(EkN), which is an exponential complexity.

**Exhaustive Unbounded Backtracking:**

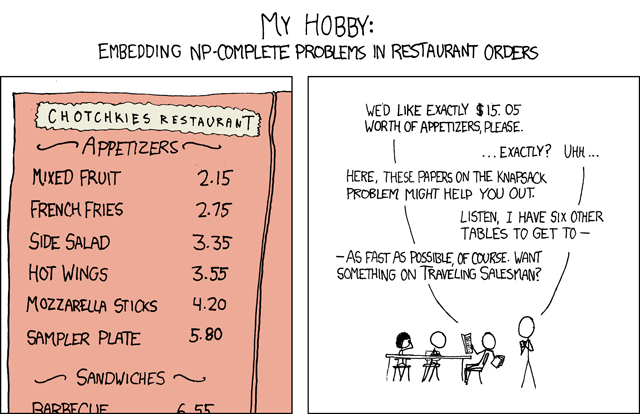
Number the nodes 1 to N. For each node encountered, it will successively try the colours. If they all fail, it will undo the colour of the previous node, and then try colours for it. If it succeeds, then it will move to the next node.

|  |
| --- |
| **function** colourGraph()  n = 1  **while** n <= N **do**  colour in the node n, with a colour which has not been tried yet  **if** there is no such colour **then**  **if** n > 1 **then**  n = n - 1  **else**  fail  **endif**  **else** if n = N **then**  print colouring  **else**  n = n + 1  **endif**  **endif**  **end function** |

All known algorithms for graph colouring, when we consider three or more colours are exponential. Graph colouring is a problem we call a NP-complete problem.

An NP-complete problem is a problem where the only known solutions are exponential. Whether or not an NP-complete problem can have a polynomial-time algorithm solution is one of the most important **open** problems in Computer Science. This is called the P=NP problem.

If you want an interesting read on what NP actually stands for, xkcd has a very [interesting article on it](http://www.explainxkcd.com/wiki/index.php/287:_NP-Complete).



Intractable Problem II: The Travelling Salesman

I mentioned this one earlier. You’re selling some sort of super interesting product, and as part of your job, you have to travel around different cities to make your sale. Since you want to be as profitable as possible, you want to do the following:

* Visit each city once -- visiting it again is a waste of time and fuel
* Return to your starting point
* Be of the minimum possible distance -- Want to use as least fuel as possible, right?

This is called an optimisation problem. We’re not trying to find the solution. We’re trying to find the best solution.

**Solution by Exhaustive Enumeration**

One approach to this, is to start at any city, and check every route possible. Of each of these routes, you write down the length of the route. Once you’ve checked every route, you choose the one of the minimum distance.

A problem quickly arises with this approach. If we have 8 cities, we have 7 choices, then 6, then 5, then four, … then one choice. This is (N-1)! choices. With 8 cities, we would have to check **5040** different possibilities. What about if we wanted to spend a week on the road, and visit 30 cities? That is 8841761993739701954543616000000 possibilities :). If it took a computer 0.0001 seconds to check each of these possibilities, then it would take approximately [several million times the age of the universe](http://www.wolframalpha.com/input/?i=%288841761993739701954543616000000+*+0.0001%29+seconds+in+years) to complete.

It’s pretty clear to see that this is an exponential solution.

Intractable Problem III: Knapsack Problem

The knapsack problem presents us with varying items of varying weights. Consider the case where we can carry 300 pounds of equipment in a knapsack. Each of the equipment has an associated weight, and value. We’re presented with various items, and we want to walk away with the highest value as possible. There’s various approaches to this, including enumeration, branch and bound, dynamic programming, and greedy methods. This is covered in one of our labs, which is examinable. Hence I’ll probably write on that at one point.

I myself remember this as the “Skyrim Problem”. You’re somewhere in a tomb with a gratuitous amount of valuables lying about. You want to choose what you want to take home, and what you’re going to leave, without over encumbering yourself.

Intractable Problem IV: Job Scheduling, Rostering and Timetabling

(Aka Software Engineering)

The problem is when there is a collection of tasks are needed to be allocated to slots. These slots could be time periods, machines, rooms, exam halls (urgh) or even all of these together. There’s usually a set of rules (constraints) these need to follow:

* **Compatibility:** Does the task fit the slot?
* **Scheduling:** Does this task need another task to be completed before it? Pretty much any time-related constraints.
* **Resources:** Do we have limited or unlimited resources? I.e number of desks in an exam hall.
* **Optimisation:** Solutions which maximise or minimise given measures. All tasks may need to be completed in a minimum time.

**Algorithms for Scheduling:**

|  |  |
| --- | --- |
| **Polynomial Algorithms** | **Exponential Algorithms** |
| Often simple allocation methods | Most problems have only exponential solutions |
| First come first serve, and greedy methods are linear | Exhaustive enumeration is one approach |
| Bus scheduling is one example | Graph Colouring is one example |

Puzzles and Heuristics

**Heuristics** are rules that can be used in decision making. They are intended to reach optimal solutions, but are not guaranteed to. These can sometimes be used to reduce an exponential time algorithm to a polynomial time, but there is often a trade-off, where you will only result with approximate solutions.

**Puzzles** are mostly only solvable by exponential solutions. Most approaches are exhaustive search based unbounded backtracking.