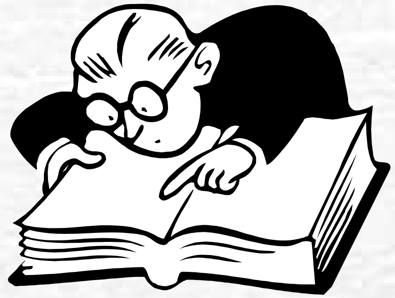
APPROXIMATION ALGORITHMS

Dario Fanucchi

Why Approximate?

Exact Solution Exists



But Searching…

Takes time

Outline

Specific Algorithms:

- Bin Packing
- Real Valued Knapsack
- Traveling Salesperson
- Graph Colouring
- Systems of Equations

General Considerations

- Trimming an exhaustive search
- Time-outs and implementation

Packing the Rubbish

The Problem:

- n real numbers $\{s_1, s_2, .., s_n\}$ in [0;1]
- pack them into minimum number of bins of size 1.

Exact Algorithm = $O(n^{n/2})$

Approximate Algorithm(FFD) = $O(n^2)$

At most $0.3\sqrt{n}$ extra bins used.

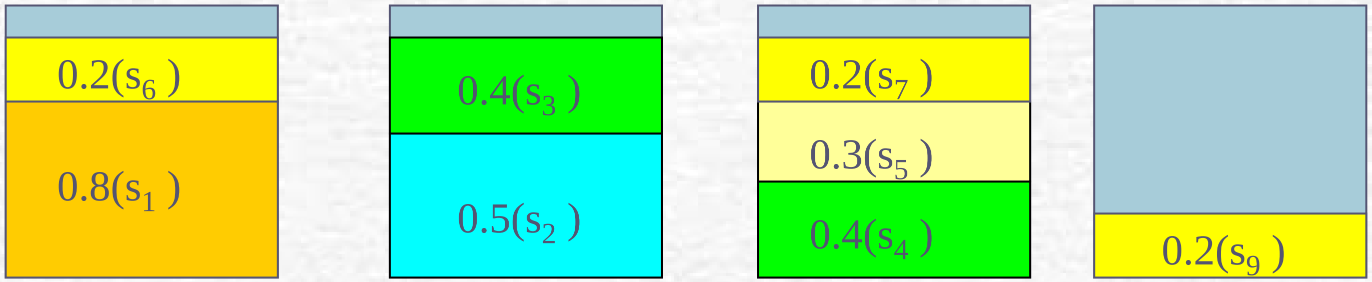
FFD Strategy

First Fit Decreasing Strategy

- **Sort the values of s_i**
- **Pack values into first bins they fit in.**

S = {0.2, 0.4, 0.3, 0.4, 0.2, 0.5, 0.2, 0.8}

Sorted: {0.8, 0.5, 0.4, 0.4, 0.3, 0.2, 0.2, 0.2}



Packing the Bags

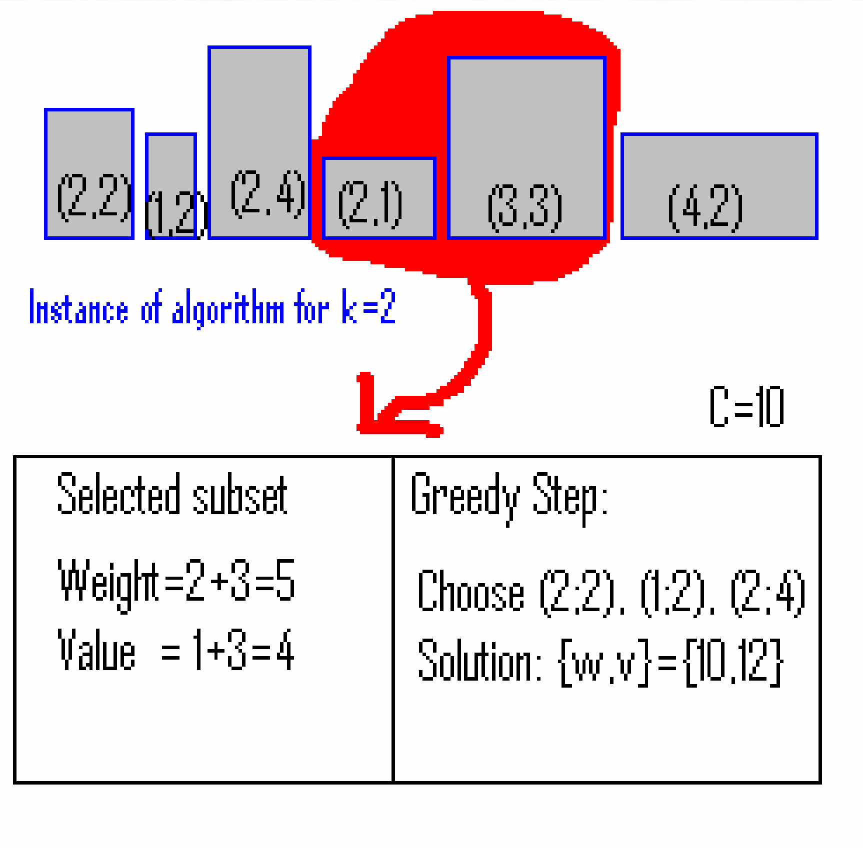
The Problem:

- Knapsack: real (weight, value) pairs
- Find a combination of maximal value that fits in boundry weight C.

Problem is NP-complete

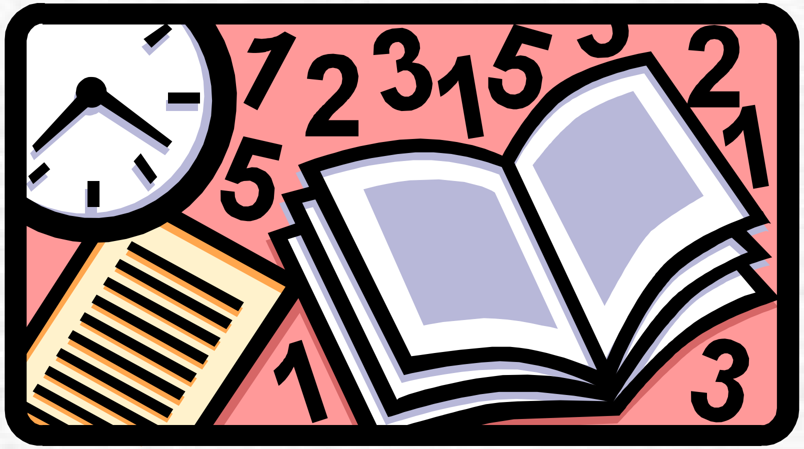
Many Approximations: Time vs. Accuracy Tradeoff…

The Algorithm



- **$sKnap_k$ Algorithm**
- Choose k
- Generate k-subsets of items
- Greedily add to subsets
- Take maximum

How close are we?



- **$sKnap_k$ accuracy**
- Ratio of 1+1/k to optimal!!
- $O(kn^{k+1})$
- Choose k wisely!

World Tour

The Problem:

- **Traveling Salesperson Problem**
- **Find minimal tour of the graph that visits each vertex exactly once.**

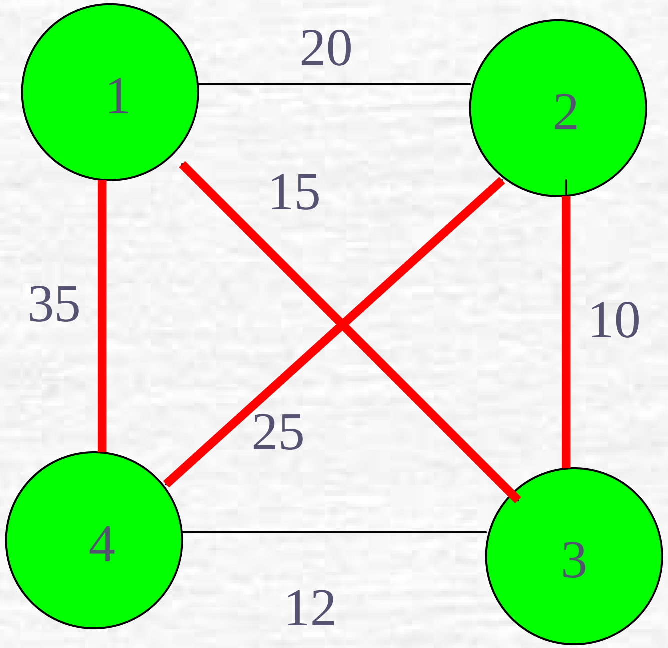
Famous NP-complete problem

Several Approximation strategies exist

But none is very accurate

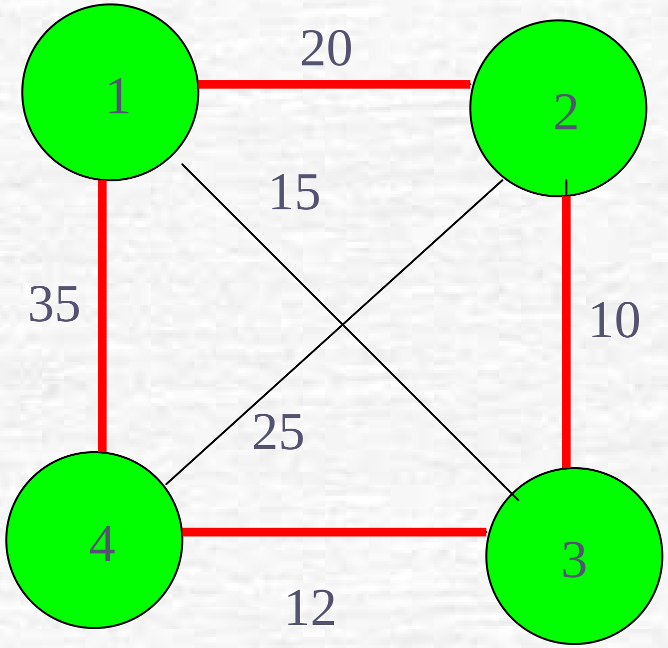
Nearest Neighbour

- Start at an arbitrary vertex
- At each step add the shortest edge to the end of the path
- No guarantee of being within a constant bound of accuracy.



Shortest-Link

- Works like Kruskal's Algorithm
- Find shortest edges
- Ensure no cycles
- Ensure no vertex with 3 edges
- Add edge



Salesperson's Dilemma

- Exact = Time Drain?
- Approximate = only a guess?
- Solution: Branch and Bound?

Colouring in

The Problem:

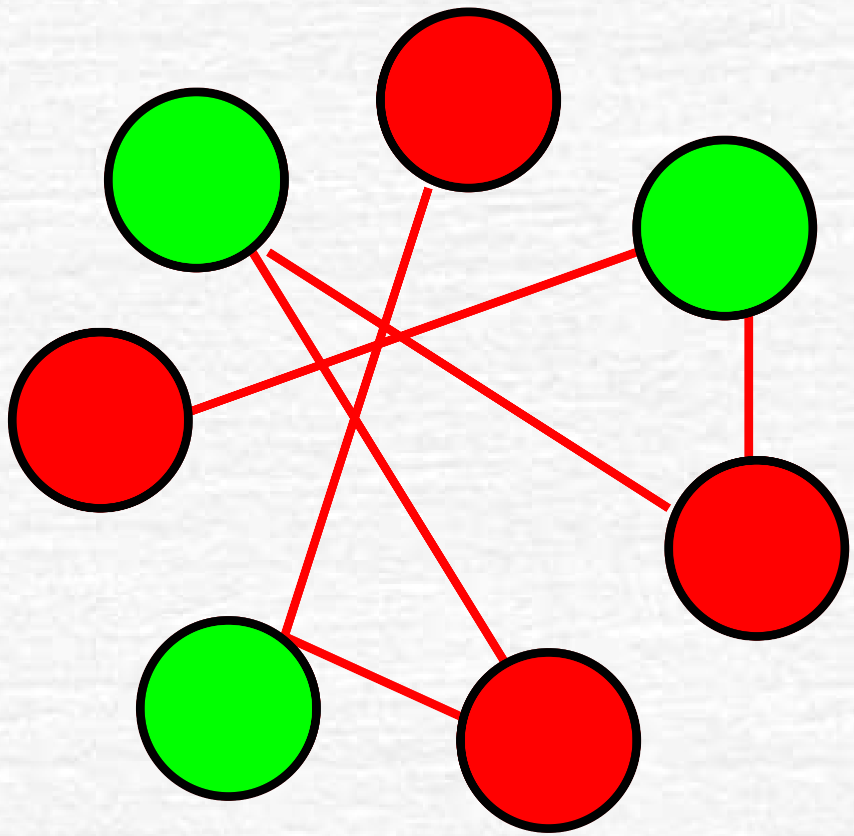
- **Graph colouring problem**
- **Exhibit a colouring of vertices with the smallest number of colours such that no edge connects two vertices of the same colour**

NP-Complete problem

Like TSP, approximations are unbounded

The Greedy One

- **Sequential Colouring Strategy**
- Assign minimum possible colour to each vertex that is not assigned to one of it's neighbours.

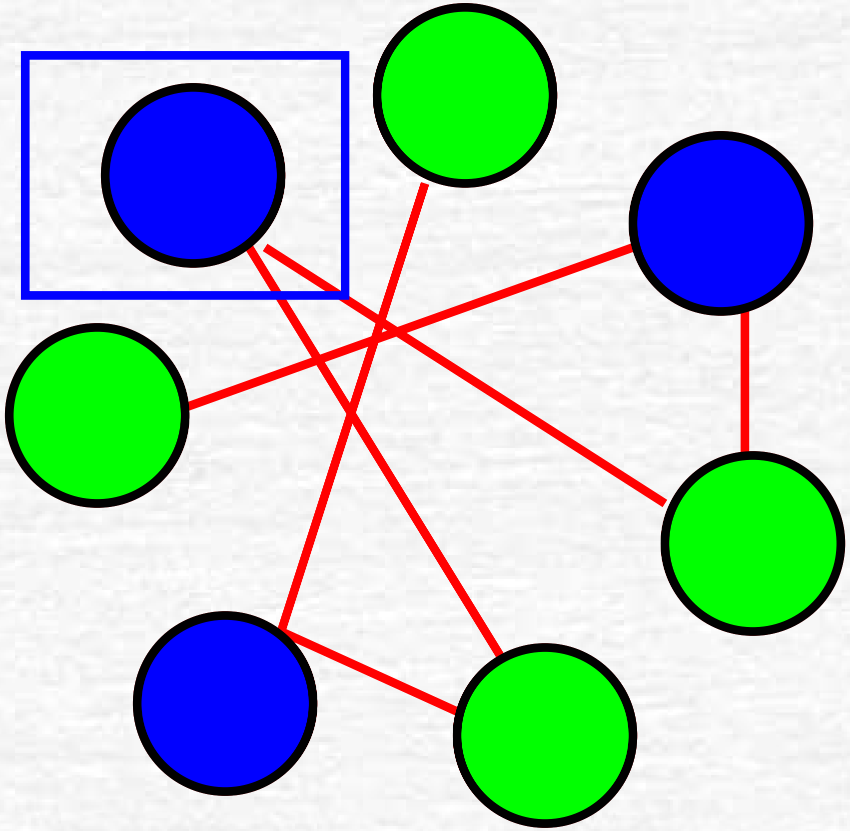


Widgerson Arrives

- Recursive Algorithm
- Base Case: 2 Colourable Graphs
- Find the subgraph of the Neighbourhood of a given vertex, recursively colour this subgraph.
- At most $3\sqrt{n}$ colours for an n-colourable graph.

Trace of Widgerson

- First run recursively on highest degree vertex
- Then run SC on the rest of the graph, deleting edges incident to N(v)



Solving Systems of Equations in Linear Time

- **Exact Algorithm** = Gaussian Elimination: $O(n^3)$
- **Approximate Algorithm**=Jacobi Method: Faster
- $\underline{\mathbf{x}}^{[m+1]}=D^{-1}[\underline{b}-(L+U)\underline{\mathbf{x}}^{[m]}]$
- $x_k^{[m+1]} =(1/a_{kk})(b_k-a_{k1}x1^{[m]}-\ldots-a_{kn}x1^{[m]})$

Gardening

- Trimming exhaustive search
- **Branch&Bound**
- **Backtracking**
- Mark a node as infeasible, and stop searching that point.

Leave while you're ahead

- Keep track always of the best solution so far
- Write this out when time is up
- Keeping track of time (C++)

```
#include<ctime>
clock_t t1,  t2;
t1 = clock();
//do stuff
t2 = clock();

double Time;
Time=double(t1)-double(t2);
Time/=CLOCKS_PER_SEC;
```

In Summation

- When exact code takes too long (and there are marks for being close to correct) approximate.
- Trade-off: Time vs. Accuracy
- Search for simplifications to problems that do not need Approx. Solutions.