Welcome to

CS 149: Introduction to Combinatorial Optimization

Kevin Tierney
Serdar Kadioglu
Max Barrows
Meinolf Sellmann

Prerequisites

- You know what algorithmic computer science is all about:
  - Given an input, compute an output according to a functional specification in finite time -> algorithmic problem specification.
  - In practice, finite termination is of course not enough: we need answers, say, within our lifetime (or that of our computer) -> NP-hardness.
  - Efficient data-structures are key ingredients of fast algorithms -> abstract data types like stacks, queues, min-heaps etc.

Prerequisites – Cont'd

- You know what linear algebra is all about:
  - Linear functions over rings can be modeled as matrices.
  - Linear equation systems can be solved using Gaussian elimination.
  - There are clear conditions under which equation systems are not solvable, uniquely solvable, or have a whole set of solutions.
- C++ - You know to program in an imperative and object oriented computer language.

Therefore

- You are prepared to get some real work done!
- Real work, that is for us not:
  - to make your computer work – operating systems
  - to make your computer understand you - compilers
  - to study models of computation - theory
  - to manage data – data bases
- Real work is for us to use the sound foundations given by the above branches of CS to solve real problems of real people outside of computer science!

Knapsack Problem

- Items 1,.., n with
- profits $p_1,\ldots, p_n$ and
- weights $w_1,\ldots, w_n$

- Limited Capacity C
- $w^T x \leq C$
- $x \in \{0,1\}^n$
- $p^T x \rightarrow$ max!

Knapsack Problem

It is easy to find a first Solution

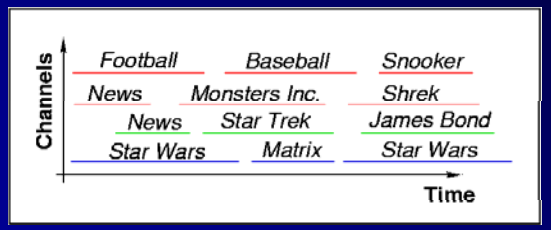
Value: B$

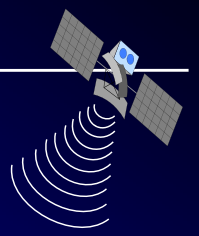
- Are there feasible, improving solutions at all?
- Is an item always taken in all feasible, improving solutions?
- Is an item never taken in all feasible, improving solutions?

Knapsack Problem

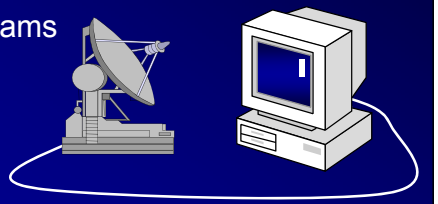
- Okay, maybe the burglar example is not that realistic. =)
- The problem of making optimal use of a limited resource that exhausts linearly is highly relevant, though!
- For example:
  - Burn a music CD while making best use of the available space.
  - Make optimal use of available bandwidth.

Automatic Recording

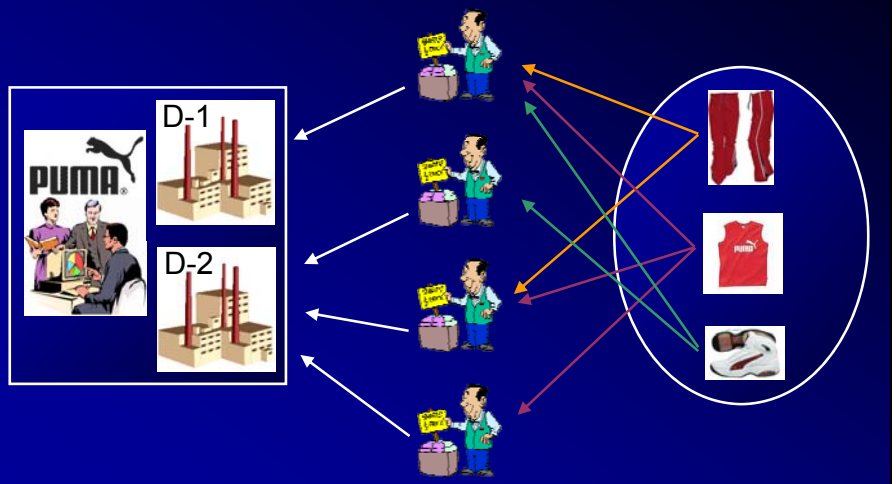




- automatic weighting of programs
- only one recording unit
- limited disc space



The Market Split Problem

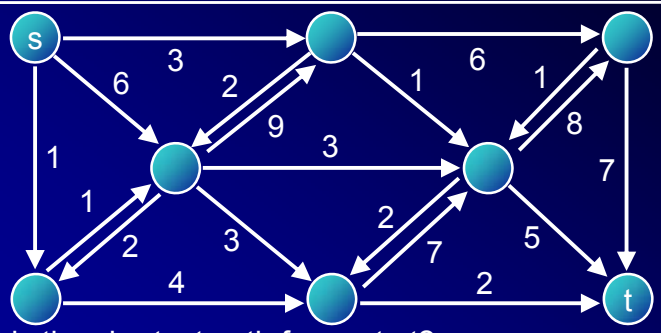


The Market Split Problem – Cont'd

70%

30	0	70	0	=	70
15	0	20	15	=	35
24	56	0	0	=	56

Network Problems



- What is the shortest path from s to t?
- What is the minimum spanning tree?
- What is the maximum flow that we can send from s to t?
- What is the minimum cut that separates s and t?

Satisfiability

- Given Boolean Variables X1,…,Xn, we define Literals as the variables themselves or their negation: L1 := X1 or L2 := ¬X3 (read '¬' as 'not')
- We define Clauses as disjunction of literals: C1 := (X1 v X2 v ¬X3) or C2 := (¬X1 v X3) (read 'v' as 'or')
- We say that a Boolean formula is in Conjunctive Normal Form if it is given as a conjunction of clauses: F1 := (X1 v X2 v ¬X3) ∧ (¬X1 v X3) (read '∧' as 'and')

Satisfiability

- Given a Boolean formula in conjunctive normal form, the Satisfiability Problem (SAT) consists in deciding whether there exists a Truth Allocation to the variables such that the formula is fulfilled!
- F1 := (X1 v X2 v ¬X3) ∧ (¬X1 v X3) is fulfilled, for example, when setting
  X1 := false, X2 := false, X3 := false
- Is F2 = X1 ∧ X2 ∧ (¬X1 v X3) ∧ (¬X1 v ¬X2 v ¬X3) satisfiable?

Satisfiability

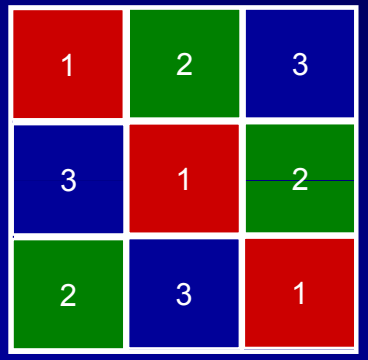
- SAT is of high practical importance!
- To solve logic systems:
  - X1 ≅ "Rain." X2 ≅ "Umbrella". X3 ≅ "I Get Wet."
    Fact C1: "Rain". Fact C2: "No Umbrella."
    Rule C3: "Rain and no Umbrella implies I Get Wet."
    C3 = (X1 ∧ ¬X2) ⇨ X3 = ¬ (X1 ∧ ¬X2) v X3 =
    (¬X1 v X2 v X3)
  - Question: Do I Get Wet?
  - It can be inferred that I Get Wet if and only if
    F := X1 ∧ ¬X2 ∧ (¬X1 v X2 v X3) ∧ ¬X3
    is not satisfiable!
- Hardware verification makes heavy use of SAT-solvers!

Puzzles

S E N D
+ M O R E

M O N E Y

Puzzles - Latin Square Completion



- Latin Square: An n x n square with n times the numbers 1,…,n such that in each row and each column we find a permutation of 1,…,n!
- Latin Squares are easy to construct.
- Harder problem: Given a partially filled square, decide whether there exists a completion to a Latin Square or not!

Why is it called "Combinatorial Optimization"?

- For all problems that we looked at, there exists a finite set of possible solutions:
  - Knapsack: There are 2^n combinations of items.
  - SAT: There are 2^n truth allocations.
  - Latin Square Completion: There are not more than $n^{n \times n}$ possible fillings.
- Solving our problems therefore calls for the investigation of large finite sets of possible solutions.
  - Mathematician: That is easy!
  - Theoretical Computer Scientist: But KP, ARP, Market Split, SAT, and Latin Square Completion are NP-hard!

Are we crazy?

- Okay, so we want to solve NP-hard problems.
- We are no super-heroes, but we are not crazy either. ☺
- We need to solve NP-hard problems every day – and not only to burn a music CD.
- Our challenge is two-fold:
  - We want algorithms that run fast in practice to solve instances as large as possible.
  - We do not want to do pure engineering. Therefore, we need to identify tractable subtasks for which we then develop poly-time algorithms that will speed up the overall computation.

What you will learn

- Understand how to model optimization problems!
- Learn more about state-of-the-art methods and algorithms how to solve optimization problems!
  - Linear Programming (important for CS 250 "Advanced Algorithms", CS 258 "Combinatorial Optimization", CS 295 "Approximation Algorithms" and "Stochastic Optimization")
  - Branch and Bound, Integer Programming
  - Constraint Programming
  - Local Search
- Learn how to use standard solvers like Cplex and Ilog Solver!

Thank you!