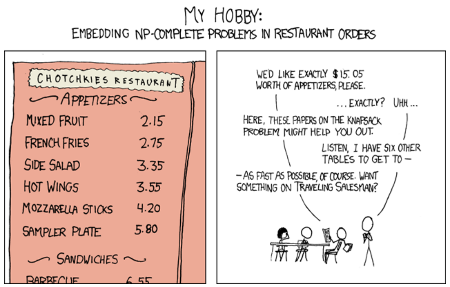
CSE 421: Algorithms

Winter 2014

Lecture 26: (more!) Poly-time reductions

Reading:
Section 8.8



subset sum

- **SUBSET-SUM:** Given natural numbers $w_1, \ldots, w_n$ and an integer W, is there a subset that adds up to exactly W?

- Ex: { 1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344 }, W = 3754.
- Yes. 1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754.

- Remark. With arithmetic problems, input integers are encoded in binary. Polynomial reduction must be polynomial in binary encoding.

- **Claim:** 3-SAT $\leq_P$ SUBSET-SUM.
- Pf. Given an instance Φ of 3-SAT, we construct an instance of SUBSET-SUM that has solution iff Φ is satisfiable.

subset sum

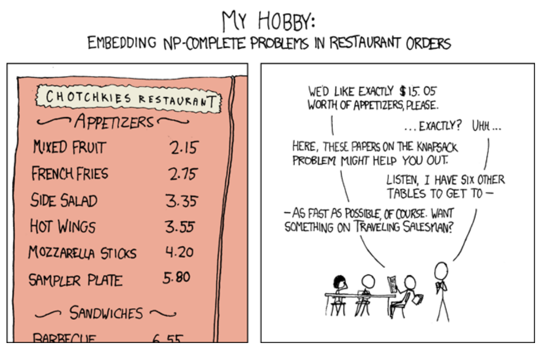
- **Construction:** Given 3-SAT instance Φ with n variables and k clauses, form 2n + 2k decimal integers, each of n+k digits, as illustrated below.
- **Claim:** Φ is satisfiable iff there exists a subset that sums to W.
- Proof: No carries possible.

$C_1 = \bar{x} \vee y \vee z$
$C_2 = x \vee \bar{y} \vee z$
$C_3 = \bar{x} \vee \bar{y} \vee \bar{z}$

	x	y	z	C_1	C_2	C_3	
x	1	0	0	0	1	0	100,010
¬ x	1	0	0	1	0	1	100,101
y	0	1	0	1	0	0	10,100
¬ y	0	1	0	0	1	1	10,011
z	0	0	1	1	1	0	1,110
¬ z	0	0	1	0	0	1	1,001
	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
	0	0	0	0	1	0	10
	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444

dummies to get clause columns to sum to 4

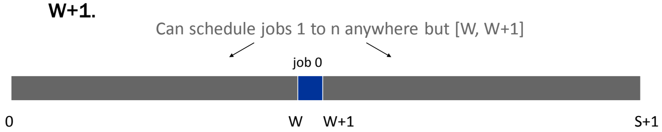
xkcd



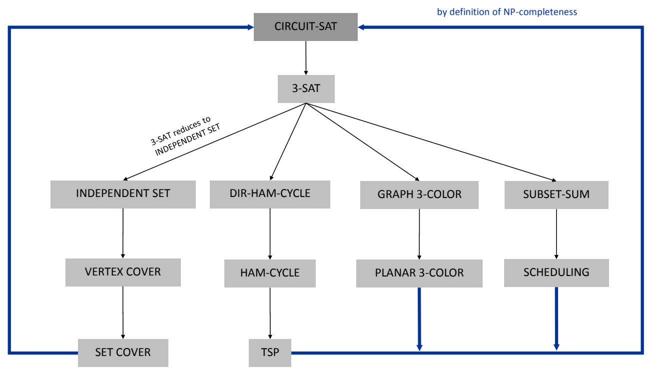
scheduling with release times

- **SCHEDULE-RELEASE-TIMES.** Given a set of n jobs with processing time t_i, release time r_i, and deadline d_i, is it possible to schedule all jobs on a single machine such that job i is processed with a contiguous slot of t_i time units in the interval $[r_i, d_i]$?

- **Claim: SUBSET-SUM $\leq_P$ SCHEDULE-RELEASE-TIMES.**
- Pf. Given an instance of SUBSET-SUM $w_1, \ldots, w_n$, and target W,
  - Create n jobs with processing time $t_i = w_i$, release time $r_i = 0$, and no deadline ($d_i = 1 + \Sigma_j w_j$).
  - Create job 0 with $t_0 = 1$, release time $r_0 = W$, and deadline $d_0 = W+1$.



NP-completeness



NP-completeness

Candy Crush is NP-hard

Toby Walsh

NICTA and University of NSW, Sydney, Australia

Abstract

We prove that playing Candy Crush to achieve a given score in a fixed number of swaps is NP-hard.

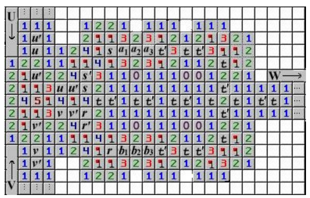
Keywords: computational complexity, NP-completeness, Candy Crush.

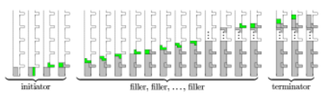
1. Introduction

Candy Crush Saga is currently the most popular game on Facebook. It has been installed half a billion times on Facebook, and on iOS and Android devices. Candy Crush is a variant of match-three games like Bejeweled and Diamond Mine. These games themselves have also been very popular. For example, PopGap Games sold over 75 million copies of Bejeweled. But what makes Candy Crush (and its ancestors) so addictive? In this paper, we suggest one answer. Namely, part of its addictiveness may be that Candy Crush is a computationally hard puzzle to solve. It thus joins the ranks of other computationally intractable puzzles that have fascinated us like Minesweeper [Kay00], Sudoku [KPS08], and other matching problems like Tetris [DHLN03], and KPlumber [KMS+04]. It raises a number of open questions. For example, is the infinite version Turing-complete? How do we generate Candy Crush problems which are truly puzzling?

To provide a formal result, we need a precise description of the puzzle. We focus on the early rounds of Candy Crush Saga where a player has to achieve a given score with a fixed number of swaps. A swap interchanges two neighbouring candies to create a chain of identical candies. Such chains are deleted from the board and the candies above drop down into their place. We also focus on simple game play which creates chains of three identical candies. In all the boards we consider, chains of more than three identical candies cannot be formed. As in previous work [TW06], we consider a generalized version of Candy Crush in which the board size is not fixed.

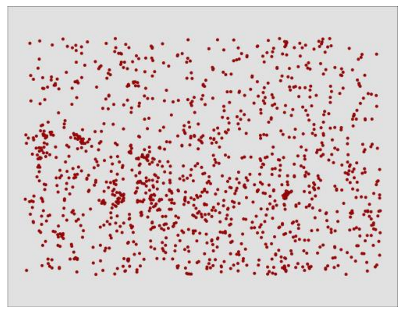
Preprint submitted to Elsevier *March 11, 2014*





coping with NP-completeness

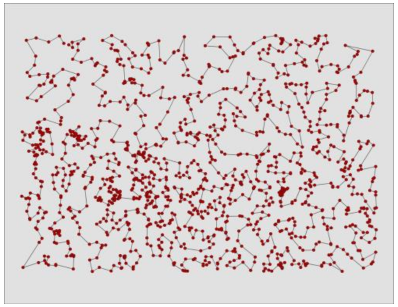
Approximation.



Euclidean TSP

coping with NP-completeness

Approximation.

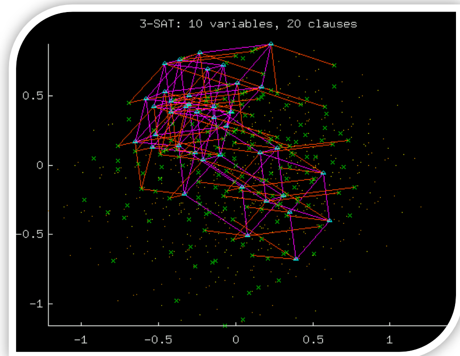


Euclidean TSP

coping with NP-completeness

Average case inputs: **E.g. random 3-SAT**

$$\Phi = (x_1 \vee \neg x_2 \vee x_5) \wedge (x_2 \vee x_4 \vee \neg x_7) \wedge (\neg x_3 \vee x_8 \vee \neg x_9) \wedge \cdots$$



new algorithmic frontiers