CSC 373 Lecture 22

Review from lecture 20: PRIME and FACTOR

transforming 3CNF to SUBSET-SUM

- Finish up transformation to SUBSET-SUM and the resulting NP completeness for PARTITION and makespan.
- Start (actually return) to local search
- Oblivious vs non-oblivious
- (Weighted) Max Cut and Exact Max-k-SAT

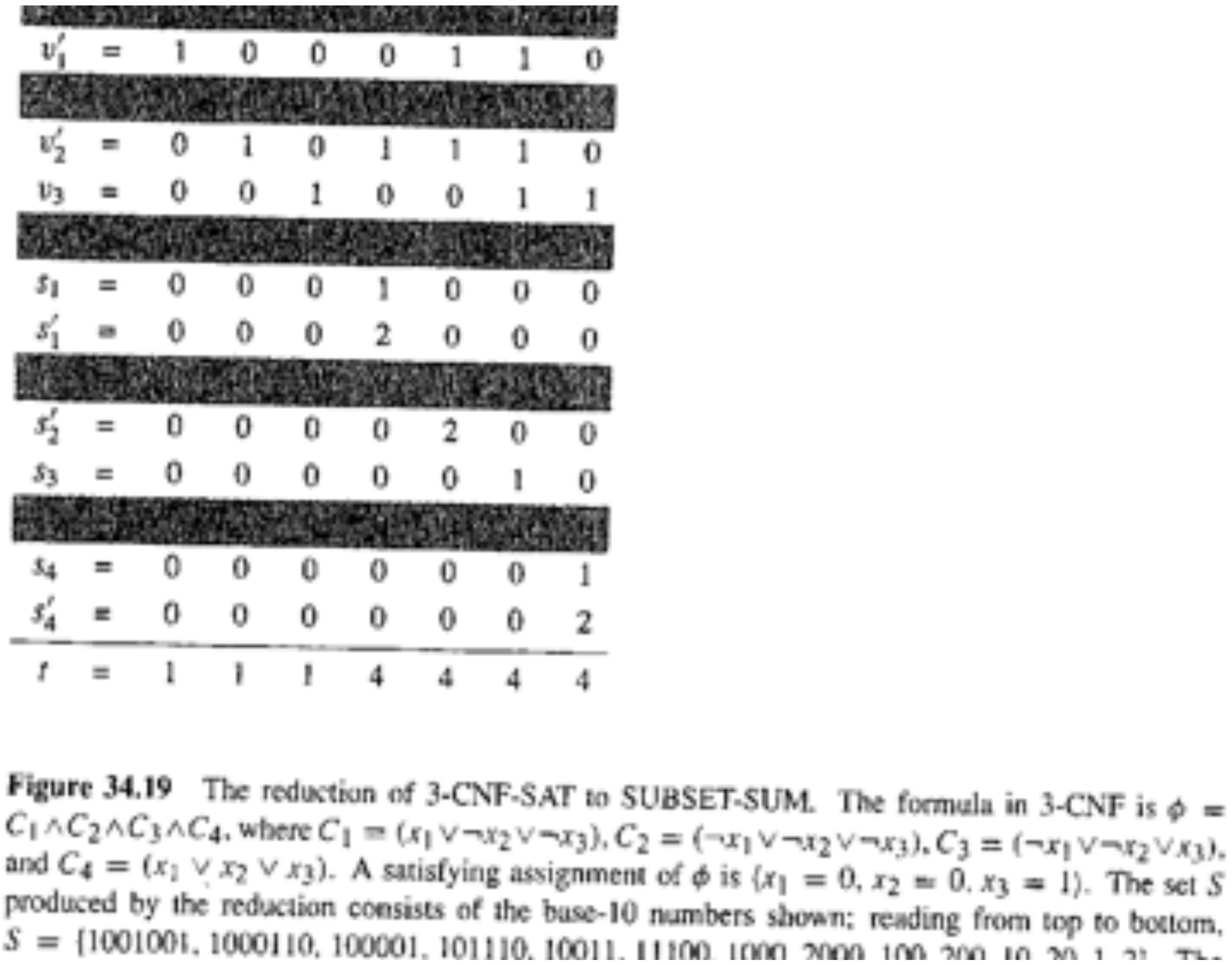
Exact 3SAT transforms to SUBSET-SUM

- We need to transform an exact 3CNF formula F (say with n variables and k clauses) to an input instance, call it phi(F) = (U,t), for SUBSET-SUM so that F is satisfiable iff phi(F) is in SUBSET-SUM.
- We create a (2n+2k) x (n+k) array as follows: we have 2 rows for each variable, one corresponding to setting the variable true and the other for false. We also have 2 rows for each clause to adjust for how many literals are being made true for a given assignment. .

3SAT to SUBSET-SUM continued

- The first n columns correspond to variables and the last k columns correspond to clauses. For a given variable row, the corresponding variable column has a 1, and each clause has a 0 or 1 depending on whether or not the variable setting satisfies the clause. Each row is then viewed as one of the integer inputs in U and the target t is the number (say in decimal so that there are no carries) represented by 111…4444 where there are 1's in the column corresponding to the variables and 4's in the columns corresponding to clauses. See figure 34.19

Figure 34.19 of CLRS



[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
v'_1	=	1	0	0	0	1	1	0
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
v'_2	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

Figure 34.19 The reduction of 3-CNF-SAT to SUBSET-SUM. The formula in 3-CNF is $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, $C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$, and $C_4 = (x_1 \vee x_2 \vee x_3)$. A satisfying assignment of ϕ is $\langle x_1 = 0, x_2 = 0, x_3 = 1 \rangle$. The set S produced by the reduction consists of the base-10 numbers shown; reading from top to bottom, $S = \{1001001, 1000110, 100001, 101110, 10011, 11100, 1000, 2000, 100, 200, 10, 20, 1, 2\}$. The

Local Search

- We now return to the topic of local search which we briefly introduced in the context of the Ford Fulkerson FF algorithm for max flow (although it is not usually presented in this context).
- We will consider a very small selection of reasonably basic applications of local search following material is sections 12.2,12.4-12.7 of KT as well as material from my spring 2011 CSC373 course.

The basic local search meta-algorithm

- Initialize *S*

  ```
  While there is a "better solution" S' in Nbhd(S)
      S := S'
  End While
  ```

- Here "better" can mean different things. For a search problem, it can mean "closer" to being feasible (in some sense); for an optimization problem it usually means being an improved solution.

Many issues concerning local search

- How do we define the local neighbourhood and how do we choose an *S'* in *Nbhd(S)*?
- Can we guarantee that a local search algorithm will terminate? And if so, how fast will the algorithm terminate?
- Upon termination how good is the local optimum that results from a local search optimization?
- How can we escape from a local optimum (assuming it is not optimal)?

Locality gap

- The worst case ratio (over all local optimum) between a local optimum and a global optimum is called the *locality gap*.
- As such the approximation ratio is at least as good as the locality gap and could sometimes be better if (say) we never reach a worst case local optimum from the initial solution.

Some options for the vanilla scheme

- How to choose the initial solution? Often one takes a naive or trivial solution, or a random solution, or a solution computed by a simple or efficient method such as a greedy algorithm.
- How to define the local neighborhood Nbhd(S)of a solution *S*? In problems where a solution is a subset or equivalently a vector <*x_1, ...,n_n*> then a natural neighborhood (but not the only one) is a Hamming distance d neighborhood; i.e. the set of vectors *z: z_i* not equal to *x_i* for at most *d* indices.

Non oblivious local search

- For an optimization problem by ''better'' we usually mean with respect to the given objective function. Such local search algorithms have been called ``oblivious local search''.
- If instead we interpret ``better'' with respect to some related potential function, this is called non oblivious local search. (We will soon see an example of non-oblivious local search).
- For either oblivious or non-oblivious local search do we take any better solution or do we search for the best solution (i.e. a "greedy local search") in *Nbhd(S)*?

Stochastic local search

- Beyond all these issues, most heuristic (and one might say practical) applications of local search allow ways to escape local optima in some controlled way. Usually this is done in some randomized method and the terminology used is ``stochastic local search''.
- Perhaps the best known generic method in this regard is a parameterized method called ``simulated annealing''. For satisfiability problems, there is a popular class of stochastic local search methods under the name WALKSAT.

Local search with Hamming Nbhd

- We will consider three applications of local search approximation algorithms where the neighborhood is defined by a small Hamming distance; namely, max cut, max-2-sat, and max independent set in *k+1* claw-free graphs.
- We start with weighted max cut which is from the text (sec 12.4). This is an NP-hard problem and the best approx ratio known for a polynomial time algorithm is .878 (or 1/.878~ 1.139 for ratios > 1) using semi-definite programming SDP.

The weighted max cut problem

- Let *G = (V,E)* be a graph with non-negative edge weights. In the (weighted) max cut problem, the goal is to find a cut so as to maximize the cardinality (resp. weight) of the cut.
- As in min cuts, a cut is a partition *(A,B)* of the vertices and the weight of the cut (what we called the capacity in the max flow-min cut setting) is the sum of weights of edges *(u,v)* such that *u* in *A, v* in *B*.
- There is a simple local search algorithm that achieves approximation ratio 2 and it is still an open problem if any greedy-like or local search algorithm can do better than this ratio.

Single move local search

- Let *(A,B)* be a partition. (Note that in this problem every partition is feasible.) Then *N_d(A,B)* (i.e. say as denoted by the characteristic vector of *A*) is the neighbourhood of partitions (i.e. char. vectors) at distance at most d from *(A,B)*.
- Choose any initial partition *(A,B)*

```
While there is a better partition (A',B') in N_1(A,B)
(A,B) : = (A',B')
End While
```