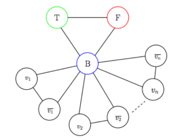
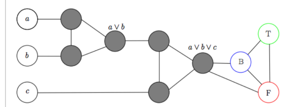
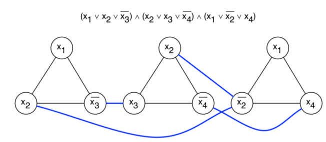
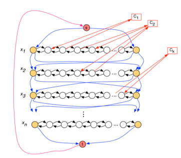
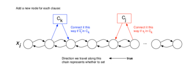
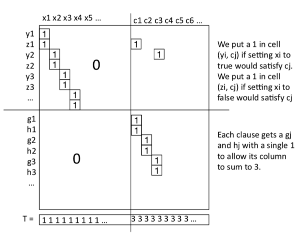
Bipartite Matching < Network Flow < Min-Cost Max-Flow < LP

Algorithms	Runtime/Key Results	NOTE
DP shortest path		
Dijkstra's	O(mlogn)-using heap O(m + nlogn) - using Fibonacci heap	Only works for non-neg weights
Bellman-Ford	$O(m\,n)$	Works for general case $D(v,k)$ = the min path from s to v using k or fewer edges $D(v,k) = \begin{cases} 0 & k=0 \text{ and } v=s \\ \infty & k=0 \text{ and } v \neq s \\ min\,D(v,k-1), min\,D(x,k-1)+len(x+v) & \text{else} \end{cases}$
APSP		
Matrix Products	$O(n^3 \log n)$	
Floyd-Warshall	$O(n^3)$	
Adapting Dijkstra	$O(n(n+m)\log n)$	Run Dijkstra n times for each starting vertex
TSP		
	$O(n^2 2^n)$	Use subset DP $C(S,t)$ = the min cost path from x to t and visits all vertices $C(s,t) = \begin{cases} len(x,t) & \text{if } S = x,t \\ min_{t' \in S, t' \neq t, t \neq x} C(S-t,t') + len(t',t) & \text{else} \end{cases}$
Network Flow		
Ford-Fulkerson	$O(m\,F)$	
Edmond-Karp (Fattest Path)	$O(m^2 \log F)$	
Edmond-Karp (Shortest Path)	$O(m^2 n)$	
LP		
Naive	$O(m^3)$	Find $\binom{n}{2}$ intersections and check if each intersection satisfies the other constraints. Choose one that give max objective function
Less naive	$O(m^2)$	Order the constraints, then recursively find the optimum point
Seidel's	$O(m)$ $O(d\,!m)$ - higher dimension	Order the constraint *randomly*, then recursively find the optimum. Works bc there's a good chance that we already have seen the two constraints that define the true optimum. Other algos: Simplex, Karmarkar, Ellipsoid
Approximation Algorithms		
Makespan (Greedy)	2 OPT	Pick any unassigned job and assign it to the machine with the least current load Remember $p_{\text{average}} \leq$ OPT and $p_{max} \leq$ OPT
Makespan (Sorted Greedy)	1.5 OPT	Pick largest job and assign to machine with the least current load
Vertex Cover (ILP)	2 OPT	Pick an arbitrary edge, choose both endpoints and discard all edges covered. The Integer LP is $x_v \in \{0,1\}$, min $\sum_v x_v$ s.t. $\forall \text{edge}(u,v) x_u + x_v \geq 1$
Vertex Cover (LP)	2 OPT	Allow for fractional solution and then round up by picking the vertex i such that $v_i \geq 1/2$. The LP is $x_v \in [0,1]$, min $\sum_v x_v$ s.t $x_u + x_v \geq 1$
Set Cover (Greedy)	If OPT uses k sets, then algorithm uses at most $O(k \ln n)$	Pick the set that covers the most points. Throw out all the points covered. Repeat (Think about exponential decay and recall $(1-\frac{1}{x})^{xn} = (\frac{1}{e})^n$
Online Algorithms		
Rent/Buy	Comp. Ratio $= 2 - \frac{\text{rent price}}{\text{buy price}}$	Better-late-than-never gives the best possible competitive ratio when buy price is a multiple of rent price
List Update (MTF)	$AC_{MTF} \leq 4C_{adversary}$	Use potential function for analysis $\phi_t = 2 \cdot$ (#pairs of elements ordered differently in MTF's and adversary's list) Start out with the same list. Analyze Access(x) for MTF first and then allow adversary to make swaps, then analyze

NP-Complete Problems		
3-SAT -Reduced from CIRCUIT-SAT (not shown)		
3-Coloring -Reduced from 3-SAT		
Independent Set (Set of vertices, no two of which have a common edge) -Reduced from 3-SAT		
Vertex Cover (Every edge is covered by at least one vertex) -Reduced from IND-SET	If C is vertex cover, then V-C is an independent set Just map instance (G,k) for INDEPENDENT SET —> Instance (G,n-k) for VERTEX COVER	
Set Cover (smallest collection of subsets whose union is the entire set) -Reduced from Vertex Cover	Instance G of Vertex Cover —> Instance of Set Cover where the set is the edges and the subset S_v contains all edges that are incident to vertex v for $v \in V$	
Hamilton Cycle (a cycle in a graph that visists every vertex exactly once) -Reduced from 3-SAT	**7.1 Hamiltonian Cycle** A *Hamiltonian cycle* is a cycle in a graph that visits every vertex exactly once. The HAM CYCLE problem asks, given a directed graph G, is there a Hamiltonian cycle. **Theorem 13** HAM CYCLE *is NP-complete* **Proof:** HAM CYCLE is in NP, since a certificate is just the cycle. We reduce from 3SAT. Let ϕ be an arbitrary 3SAT instance with clauses $c_1, \ldots, c_m$ and variables $x_1, \ldots, x_n$. Construct the following gadget that represents all possible truth assignments:  This includes a node (squares) for each clause. We connect the clauses up to the paths in an orientation that depends on whether the variable needs to be true or false to satisfy the clauses: 	
Subset Sum (Find a subset of integers from a given set that produce the exact sum) -Reduced from 3-SAT		
Clique (Largest complete subgraph) -Reduced from 3-SAT	**Definition 15** CLIQUE: *Given a graph G, find the largest clique (set of nodes such that all pairs in the set are neighbors). Decision problem: "Given G and integer k, does G contain a clique of size* $\geq k$*?"* **Theorem 16** CLIQUE *is* **NP**-*Complete.* **Proof:** Note that CLIQUE is clearly in **NP**; the witness is the set of vertices that are all connected to each other via edges. Next, we reduce 3-SAT to CLIQUE. Specifically, given a 3-CNF formula F of m clauses over n variables, we construct a graph as follows. First, for each clause c of F we create one node for every assignment to variables in c that satisfies c. E.g., say we have: $F = (x_1 \vee x_2 \vee \overline{x}_4) \wedge (\overline{x}_3 \vee x_4) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge \ldots$	Then in this case we would create nodes like this: $(x_1=0, x_2=0, x_4=0)$ $(x_3=0, x_4=0)$ $(x_2=0, x_3=0)$ $\ldots$ $(x_1=0, x_2=1, x_4=0)$ $(x_3=0, x_4=1)$ $(x_2=0, x_3=1)$ $(x_1=0, x_2=1, x_4=1)$ $(x_3=1, x_4=1)$ $(x_2=1, x_3=0)$ $(x_1=1, x_2=0, x_4=0)$ $(x_1=1, x_2=0, x_4=1)$ $(x_1=1, x_2=1, x_4=0)$ $(x_1=1, x_2=1, x_4=1)$ We then put an edge between two nodes if the partial assignments are consistent. Notice that the maximum possible clique size is m because there are no edges between any two nodes that correspond to the same clause c. We claim that the maximum size is m if and only if the original formula has a satisfying assignment. Suppose the 3-SAT problem *does* have a satisfying assignment, then in fact there *is* an m-clique (just pick some satisfying assignment and take the m nodes consistent with that assignment). For the other direction, we can either show that if there *isn't* a satisfying assignment to F then the maximum clique in the graph has size $< m$, or argue the contrapositive and show that if there is a m-clique in the graph, then there is a satisfying assignment for the formula. Specifically, if the graph has an m-clique, then this clique must contain one node per clause c. So, just read off the assignment given in the nodes of the clique: this by construction will satisfy all the clauses. So, we have shown this graph has a clique of size m iff F was satisfiable. Finally, to complete the proof, we note that our reduction is polynomial time since the graph produced has total size at most quadratic in the size of the formula F ($O(m)$ nodes, $O(m^2)$ edges). Therefore CLIQUE is **NP**-complete. ■

***Theorems*:**

1. Weak Duality: If x is a feasible solution of the primal, and y is a feasible solution of the dual, then
2. Strong Duality: If primal is feasible and bounded, then dual is feasible and bounded. If x* is optimal solution to the primal, and y* is optimal solution to dual, then $c^T x^* = b^T y^*$

Suppose primal is feasible and bounded, all the possible scenarios are

Primal/Dual	Impossible	F&B	Unbounded
Impossible	Y	N	Y
Feasible & Bounded	N	Y	N
Unbounded	Y	N	N

3. More Theorems…..