CS 331 NP Notes 3

Graph Coloring

- Problem Definition: Given a graph $G = (V, E)$, find the smallest number of different colors to assign for each node G so that no two nodes of the same color share an edge.
  - Decision Version: Given a graph G and a bound k, does G have a k-coloring?

Why this problem is useful:

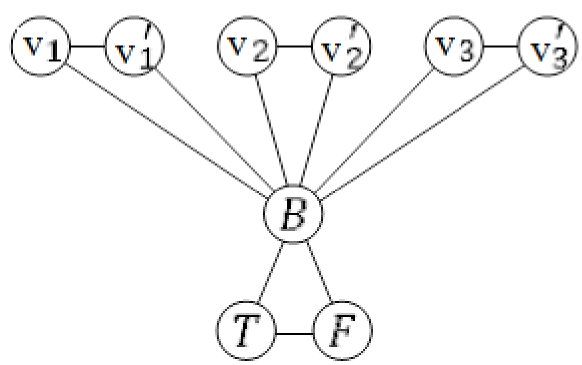
An application of this problem is when allocating resources in the presence of conflict. For example, given a set of variables and k registers. You need to map variables to registers. If two variables are use at a common point in time, they cannot be assigned to the same register. We build a graph G on the set of variables, joining two variables by an edge if they are both in use at the same time.

2-Coloring Problem: when $k = 2$, the graph coloring problem is straightforward to solve. We need to check whether a graph is bipartite.

Lemma: A graph $G$is colorable if and only if it is bipartite.

3-Coloring Problem: is a very difficult problem**.**

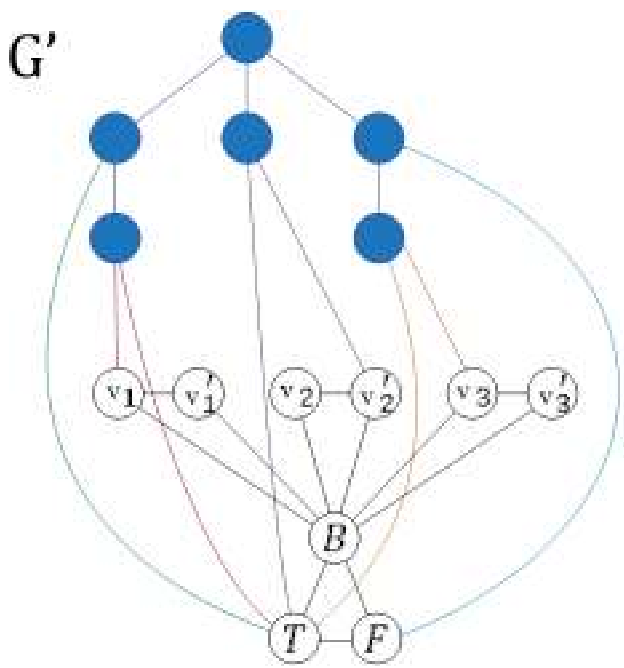
- Need to show the 3-coloring problem is $\mathcal{NP}$-Complete
  - Show 3-Coloring $\in \mathcal{NP}$
  - Show 3-SAT $\leq_P$ 3-Coloring
- Use the simplest version of 3-SAT i.e. 1 clause
  - $F = (x_1 \vee x_2' \vee x_3')$
    - First, we need to find a way to turn this problem into a graph
      - For each variable x_i, create 2 nodes in G, one for v_i and one for v_i' and connect these two nodes by an edge.
      - Define 3 "special" nodes, T, F, and B, joined in a triangle (you can think of T as the color True, F as the color False, and B as the color Base.
      - Connect every variable to B. (see graph below)



    - Note that this graph implicitly determines a truth assignment for the variables in the 3-SAT problem.
    - The graph above has some useful properties
      - Each v_i and v_i' must get different colors

    - Each v_i and v_i' must get a different color from B
    - T, F, and B must get different colors
    - if x_i is set to 1 is the 3-SAT formula, the node v_i will get the T color.
    - Hence, any 3-Coloring of the nodes of G implicitly defines a a valid truth assignment for the 3-SAT instance
- We need to grow G so that only satisfying assignments can be extended to the 3-coloring of the full graph.
- Consider the following clause: $x_1 \vee x_2' \vee x_3$, in the language of 3-coloring of G, it says that:

  "At least one of the nodes v_1, v_2', v_3 should get the T color. "
- We need to construct a graph with this property
    - Need to modify our graph G with a “gadget” to form G'. (See Next Page).



- Note: line color does not denote coloring of vertices; it is simply there for readability.
- The six-node “gadget” attaches to the rest of G at five existing nodes: True, False, v_1, v_2', and v_3.
    - This ensures that in the event that neither x_1, x_2', or x_3' are assigned true, that the lowest two shaded nodes in the subgraph MUST receive the B color, the three shaded nodes above them must receive, respectively, the F, B, and T colors, and hence there’s no color that can be assigned to the topmost shaded node.

- Meaning, that if no valid assignment is possible, then there will be no valid 3-Coloring for G'.

- **Proof:**
  - Show 3-Coloring $\in \mathcal{NP}$
    - If we are given 3 subsets and told that no edges exist between the 3 this is easy to check.
      - For every node simply check all neighbors and ensure that no neighbor exists in the same subset to which it itself is a member.
        - It takes $O(n^2)$ to verify a graph of n nodes in this fashion
  - Show 3-SAT $\leq_P$ 3-Coloring
    - For a larger problem, start with the graph G such that there is a pair of nodes x_i, x_i' for each $x_i \in X$ connected by edges to a base node B. Then, for each clause in the 3-SAT problem attach a six-node gadget to G to form G'
    - Claim: The given 3-SAT instance is satisfiable if and only if G' has a 3-coloring.
      - Suppose there is a satisfying assignment for the 3-SAT instance. We define a coloring of G' by first coloring the B, T, and F nodes arbitrarily with the 3 given colors. Next, for each i assign v_i the color of T if $x_i = 1$ and assign v_i the color of F if $x_i = 0$. Next assign each v_i' the only remaining color such that no conflict exists. This assignment should be able to be extended into G' by the reasoning mentioned in the setup.

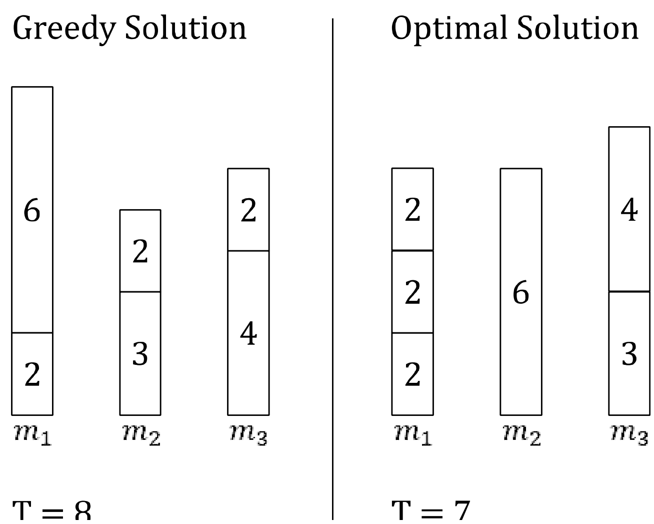
        Conversely, suppose G' has a 3-coloring. In this coloring each node v_i is assigned either the true color or the false color; and we set the literal in F correspondingly. We now claim that in each clause of the 3-SAT instance, at least one of the terms in the clause has a truth value of one; otherwise then all 3 of the nodes in the clause would have to have the false coloring. However, as explained in the setup this would result in a graph G' which was NOT 3-Colorable which is a contradiction to our original claim. ■

Approximation Algorithms

- Given an $\mathcal{NP}$-Complete problem to solve, no polynomial time algorithm exists to provide an optimal solution
  - An algorithm must be:
    - Deterministic
    - Always correct
    - Bounded by a poly. time function of its input size.
- Can we develop a polynomial time algorithm that is guaranteed to provide a close to optimal solution?
- Challenge: Need to prove a solution's value is close to optimal without knowing the optimum value.

Greedy Algorithms and Bounds on the Optimum: A Load Balancing Problem

- Problem: Given M machines $m_1, m_2, \dots m_M$ and n jobs where each job j has a processing time t_j.
  - Need to assign each job to a machine
  - Need to balance the loads across all machines
- Formulation
  - Let $A(i)$ be the set of jobs assigned to M_i.
  - The total load on M_i : $T_i = \sum_{j \in A(i)} t_j$ where t_j = the processing time for a job
  - We want to minimize the *makespan* (the maximum load on any machine) $T = \max_i T_i$
- Example 1:
  - $n = \{2, 3, 4, 6, 2, 2\}$
  - $M = \{m_1, m_2, m_3\}$
  - Greedy Algorithm: Always place the job on the least loaded machine
    - However, this does NOT provide the optimal solution in this case



- The greedy algorithm
  - Start with no jobs assigned
  - Set $T_i = 0$ and $A(i) = \emptyset$ for all m_i
  - For $j = 1, \ldots, n$
    - Let m_i be a machine that achieves the minimum $\min_k T_k$
    - Assign job j to machine m_i
    - Set $A(i) \leftarrow A(i) \cup \{j\}$
    - Set $T_i \leftarrow T_i + T_j$
  - EndFOR

<u>**Proving the Greedy Algorithm is a Correct Approximation**</u>

- Need to show the greedy solution is no more than a factor of 2 from the optimal solution
- Therefore…
  - Let T = The makespan of our greedy solution
  - Let T^* = The makespan of the optimal solution
    - We know that the optimal makespan must be at least:
      $$T^* \geq \frac{1}{m} \sum_j t_j$$
      Since the value of the makespan equals the max value over all machines.
    - However, this formula is not very useful if you have one extremely long job in relation to all the shorter jobs.
      - In this case the greedy solution actually provides the optimal solution
        - Need a different lower bound for T^* to reflect this however
        - In this case the optimal makespan is at least
          $$T^* \geq \max_j t_j$$
  - Note: We need the two formulas mentioned above for our proof

- <u>Claim</u>: Our greedy algorithm produces an assignment of n jobs to M machines within a factor of 2 of the optimal solution; i.e. $T \leq 2T^*$
    - **Proof:**
        - Consider M_i attains the max load T in the assignment
            - Consider the last job j that was placed on M_i.

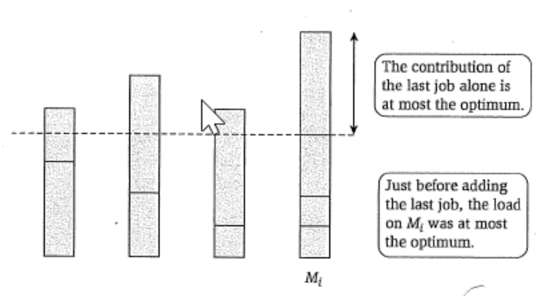


Figure 11.2 Accounting for the load on machine M_i in two parts: the last job to be added, and all the others.

- When we assign job j to M_i, M_i had the smallest load of all machines $T_i - t_j$ before adding j.
- Every machine had a load at least $T_i - t_j$
- Add up the load on all machines and we obtain $\sum_k t_k \geq m(T_i - t_j) \equiv \frac{1}{m}\sum_k t_k \geq T_i - t_j$
- Now apply the <u>two formulas above</u> to form a sequence of equations
  $T_i - t_j \leq \frac{1}{m}\sum_k t_k \leq T^*$(when t_j is **not an** extremely long job)
  $T_i - t_j \leq T^*$ (1)
  $t_j \leq T^*$ (2) (when t_j is an extremely long job)
  By adding (1) and (2), we get the following.
  $T_i \leq 2T^*$
  Therefore,
  $T_i \leq 2T^* \Rightarrow T \leq 2T^*$ by our assumption for the proof that T_i is the machine with the maximum makespan ■

- We can optimize the algorithm by sorting jobs in decreasing order first
    - Optimized greedy algorithm
        - Start with no jobs assigned
        - Set $\mathrm{T_i} = 0$ and $\mathrm{A(i)} = \emptyset$ for all machines $\mathrm{m_i}$
        - Sort jobs in decreasing order of processing time $\mathrm{t_j}$
        - Assume that $\mathrm{t_1 \geq t_2 \geq \cdots \geq t_n}$
        - For $\mathrm{j = 1, \ldots, n}$
            - Let $\mathrm{m_i}$ be a machine that achieves the minimum $\mathrm{min_k T_k}$
            - Assign job j to machine $\mathrm{m_i}$
            - Set $\mathrm{A(i) \leftarrow A(i) \cup \{j\}}$
            - Set $\mathrm{T_i \leftarrow T_i + T_j}$
        - EndFOR

- However, just saying this is better is not enough, we need to prove it
    - More specifically, we need to prove that $T \leq \frac{3}{2}T^*$
- First however, we need to prove that doing this will produce an optimal solution
    - <u>Lemma</u>: If there are more than M jobs, then $T^* \geq 2t_{M+1}$
        - Consider the first $M+1$ jobs in decreasing order
        - Each job MUST need at least t_{M+1} time to process
        - There are $M+1$ jobs but only M machines
        - ∴ it must be the case that one machine has at least two jobs
        - ∴ it must be the case that the processing time of the machine will be at least $2t_{m+1}$
- Proving $T \leq \frac{3}{2}T^*$
    - Consider $\mathrm{M_i}$ attains the max load T in the assignment
    - If M_i only holds a single job, then the schedule is optimal. Otherwise, assume machine M_i has at least two jobs, and let t_j be the last job assigned to the machine
        - It must be the case that $j \geq M+1$ since the algorithm assignes the first M jobs to M distinct machines
        - Thus $t_j \leq t_{m+1}$ (jobs are sorted in decreasing order)
        - Therefore, $t_j \leq t_{m+1} \leq \frac{1}{2}T^*$ ←occurs because of what we proved above ($\mathrm{T^* \geq 2t_{m+1} \equiv \frac{1}{2}T^* \geq t_{m+1}}$)
        - Therefore, repeating the inequalities from the first proof we now have

$$T_i - t_j \leq \frac{1}{m}\sum_k t_k \leq T^*$$

$$T_i - t_j \leq T^* \ (1)$$

$$t_j \leq \frac{1}{2}T^* \quad (2)$$

Adding (1) and (2), we get the following.

$T_i \leq \frac{3}{2}T^* \Rightarrow T \leq \frac{3}{2}T^*$ ■