CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Lecture 12: Interval Scheduling, Independent Sets and Matching

Harvard SEAS - Fall 2024 *2024-10-10*

1 Announcements

Recommended Reading: CLRS Sec 16.1–16.2

- Mid term Oct-17
- SRE 4 next Tuesday Oct-15.

2 Interval Scheduling

In Lecture 11, we modelled the conflicts arising in RAM allocation as a Coloring problem and discussed greedy algorithms for the problem. We found that the greedy - but careful - way of coloring via BFS worked well for 2-colorable graphs.

Another example where the greedy approach to algorithm design works well is Interval Scheduling. The following decision version of Interval Scheduling was introduced in Lecture 4.

Input	**:** A collection of intervals $[a_0, b_0], \ldots, [a_{n-1}, b_{n-1}]$, where each $a_i, b_i \in \mathbb{Q}$ and $a_i \leq b_i$
Output	**:** YES if the intervals are disjoint (for all $i \neq j$, $[a_i, b_i] \cap [a_j, b_j] = \emptyset$) NO otherwise

Computational Problem IntervalScheduling-Decision

We saw that we could solve this problem in time $O(n \log n)$ by reduction to Sorting. However, if the answer is NO, we might be satisfied by trying to schedule *as many* intervals *as possible*:

Input	**:** A collection of intervals $[a_0, b_0], \ldots, [a_{n-1}, b_{n-1}]$, where each $a_i, b_i \in \mathbb{Q}$ and $a_i \leq b_i$
Output	**:** A maximum-size subset $S \subseteq [n]$ such that $\forall i \neq j \in S$, $[a_i, b_i] \cap [a_j, b_j] = \emptyset$.

Computational Problem IntervalScheduling-Optimization

A greedy algorithm for IntervalScheduling-Optimization is as follows.

```
GreedyIntervalScheduling(x)
  Input  : A list x of n intervals [a, b], with a, b ∈ ℚ
  Output : A "large" subset of the input intervals that are disjoint from each other
Sort the intervals with                                   ;
S = ∅;
foreach i = 0 to n − 1 do
    |                                                     ;
    | return S
```

Theorem 2.1. `GreedyIntervalScheduling(`x`)` *finds an optimal solution to IntervalScheduling-Optimization, and can be implemented in time* $O(n \log n)$.

Proof.

□

3 Independent Set

In Lecture 11, we used graph theoretic modelling on RAM allocation to rephrase it as a Coloring problem. Graph theoretic modelling can also be applied to the IntervalScheduling-Optimization problem. For example, consider the following set of intervals:

This is the notion of an independent set, which is defined as follows for a general graph.

Definition 3.1. Let $G = (V, E)$ be a graph. An *independent set* in G is a subset $S \subseteq V$ such that there are no edges entirely in S. That is, $\{u, v\} \in E$ implies that $u \notin S$ or $v \notin S$.

Finding the largest set of conflict-free intervals becomes equivalent to finding the largest number of vertices with no edges between them. For a general graph, the analogous problem is known as the Independent Set problem.

Input	**:** A graph $G = (V, E)$
Output	**:** An independent set $S \subseteq V$ in G of maximum size

Computational Problem Independent Set

Example:

Remarks:

- **Independent Set vs IntervalScheduling-Decision:**

- **Independent Set vs Coloring:**

There are no known efficient algorithms for the Independent Set problem (we will discuss this more in future lectures). However, a greedy algorithm along the lines of `GreedyIntervalScheduling` and `GreedyColoring` can be designed which outputs a somewhat large independent set.

```
1 GreedyIndSet(G)
  Input          : A graph G = (V, E)
  Output         : A "large" independent set in G
2 Choose an ordering v_0, v_1, v_2, ..., v_{n-1} of V;
3 S = ∅;
4 foreach i = 0 to n − 1 do
5 |                                          ;
6 |  return S
```

How large is the independent set in the above algorithm? Similarly to coloring, we can only prove fairly weak bounds on the performance of the greedy algorithm in general:

Theorem 3.2. *For every graph G with n vertices and m edges,* `GreedyIndSet(G)` *can be implemented in time $O(n + m)$ and outputs an independent set of size at least $n/(\deg_{max} + 1)$, where $\deg_{max}$ is the maximum vertex degree in G.*

Proof.

□

4 Matching

While Independent Set is a graph theoretic notion about "non-conflicting" vertices, matching is about "non-conflicting" edges.

Definition 4.1. For a graph $G = (V, E)$, a *matching* in G is a subset $M \subseteq E$ such that every vertex $v \in V$ is incident to at most one edge in M. Equivalently, no two edges in M share an endpoint.

The problem of finding the largest matching in a graph is called Maximum Matching.

Input	**:** A graph $G = (V, E)$
Output	**:** A matching $M \subseteq E$ in G of maximum size

Computational Problem Maximum Matching

An example of maximum matching is depicted below

We can use a greedy strategy to try finding a maximum matching. We can start with an edge and try to add more edges till we can no longer add edges. An example is depicted below

Remarkably, sometimes it is possible to do more sophisticated operations than just adding an edge and still grow the matching. For this, we need the notions of alternating walk and augmenting path.

Definition 4.2. Let $G = (V, E)$ be a graph, and M be a matching in G. Then:

1. An *alternating walk* W in G with respect to M is

2. An *augmenting path* P in G with respect to M is

An example of alternating path

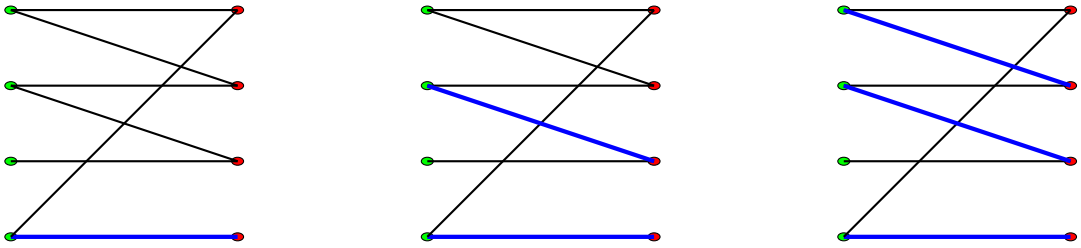

Figure 1: Alternating walk (also an augmenting path) for the matching of size 3 in Figure **??**.

This suggests a natural algorithm for maximum matching: repeatedly try to find an augmenting path and use it to grow our matching. But we need to argue that augmenting paths always exist and we can find them efficiently. An important piece in this argument is the following theorem.

Theorem 4.3 (Berge's Theorem)**.** *Let $G = (V, E)$ be a graph, and $M \subseteq E$ be a matching. If (and only if) M is not a maximum-size matching, then G has an augmenting path with respect to M.*