CSC2420: Algorithm Design, Analysis and Theory Fall 2017

Allan Borodin and Nisarg Shah

September 20, 2017

Lecture 2

Announcements:

- I plan to have (tonight?) the first few questions for assignment 1.

Todays agenda:

- Continue discussion of greedy algorithms
- The priority model (which we briefly introduced in Lecture 1).
- A few easy results and a few (perhaps) surprising results
- A perspective on algorithm design and analysis
  - When an algorithmic approach works for a given problem, we can ask if that approach still is useful for a generalization of that problem?
  - When an approach (provably) does not work for a problem, is there a way to extend that approach?

Continuing the discussion of priority algorithms

We ended last week with some discussion of the priority framework for modeling greedy and greedy-like algorithms. In particular, in our last slide for Lecture 1, we said:

- If we do not insist on greediness, then priority algorithms would best have been called myopic algorithms.
- We have both fixed order priority algorithms (e.g. unweighted interval scheduling and LPT makespan) and adaptive order priority algorithms (e.g. the set cover greedy algorithm and Prim's MST algorithm).
- The key concept is to indicate how the algorithm chooses the order in which input items are considered. We cannot allow the algorithm to choose say "an optimal ordering".
- We might be tempted to say that the ordering has to be determined in polynomial time but that gets us into the "tarpit" of trying to prove what can and can't be done in (say) polynomial time.

The priority model definition

- We take an information theoretic viewpoint in defining the orderings we allow.
- Lets first consider deterministic fixed order priority algorithms. Since I am using this framework mainly to argue negative results (e.g. a priority algorithm for the given problem cannot achieve a stated approximation ratio), we will view the semantics of the model as a game between the algorithm and an adversary.
- Initially there is some (possibly infinite) set $\mathcal{J}$ of potential inputs. The algorithm chooses a total ordering π on $\mathcal{J}$. Then the adversary selects a subset $\mathcal{I} \subset \mathcal{J}$ of actual inputs so that $\mathcal{I}$ becomes the input to the priority algorithm. The input items $I_1, \ldots, I_n$ are ordered according to π.
- In iteration k for $1 \leq k \leq n$, the algorithm considers input item I_k and based on this input and all previous inputs and decisions (i.e. based on the current state of the computation) the algorithm makes an irrevocable decision D_k about this input item.

The fixed (order) priority algorithm template

$\mathcal{J}$ is the set of all possible input items
Decide on a total ordering π of $\mathcal{J}$
Let $\mathcal{I} \subset \mathcal{J}$ be the input instance
$S := \varnothing$ % S is the set of items already seen
$i := 0$ % $i = |S|$
while $\mathcal{I} \setminus S \neq \varnothing$ **do**
    $i := i + 1$
    $\mathcal{I} := \mathcal{I} \setminus S$
    $I_i := \min_\pi \{I \in \mathcal{I}\}$
    make an irrevocable decision D_i concerning I_i
    $S := S \cup \{I_i\}$
end

Figure: The template for a fixed priority algorithm

Some comments on the priority model

- A special (but usual) case is that π is determined by a function $f : \mathcal{J} \to \Re$ and and then ordering the set of actual input items by increasing (or decreasing) values $f()$. (We can break ties by say using the input identifier of the item to provide a total ordering of the input set.) N.B. We make no assumption on the complexity or even the computability of the ordering π or function f.
- NOTE: Online algorithms are fixed order priority algorithms where the ordering is given *adversarially*; that is, the items are ordered by the input identifier of the item.
- As stated we do not give the algorithm any additional information other than what it can learn as it gradually sees the input sequence.
- However, we can allow priority algorithms to be given some (hopefully easily computed) global information such as the number of input items, or say in the case of the makespan problem the minimum and/or maximium processing time (load) of any input item. (Some inapproximation results can be easily modified to allow such global information.)

The adaptive priority model template

$\mathcal{J}$ is the set of all possible input items
$\mathcal{I}$ is the input instance
$S := \varnothing$ % S is the set of items already considered
$i := 0$ % $i = |S|$
while $\mathcal{I} \setminus S \neq \varnothing$ **do**
    $i := i + 1$
    decide on a total ordering π_i of $\mathcal{J}$
    $\mathcal{I} := \mathcal{I} \setminus S$
    $I_i := \min_{\leq_{\pi_i}} \{I \in \mathcal{I}\}$
    make an irrevocable decision D_i concerning I_i
    $S := S \cup \{I_i\}$
    $\mathcal{J} := \mathcal{J} \setminus \{I : I \leq_{\pi_i} I_i\}$
    % some items cannot be in input set
end

Figure: The template for an adaptive priority algorithm

Inapproximations with respect to the priority model

Once we have a precise model, we can then argue that certain approximation bounds are not possible within this model. Such inapproximation results have been established with respect to priority algorithms for a number of problems but for some problems much better approximations can be established using extensions of the model.

1. For the weighted interval selection (a *packing problem*) with arbitrary weighted values (resp. for proportional weights $v_j = |f_j - s_j|$), no priority algorithm can achieve a constant approximation (respectively, better than a 3-approximation).
2. For the knapsack problem, no priority algorithm can achieve a constant approximation. We note that the maximum of two greedy algorithms (sort by value, sort by value/size) is a 2-approximation.
3. For the set cover problem, the natural greedy algorithm is essentially the best priority algorithm.
4. As previously mentioned, for deterministic fixed order priority algorithms, there is an $\Omega(\log m / \log \log m)$ inapproximation bound for the makespan problem in the restricted machines model.

More on provable limitations of the priority model

The above mentioned inapproximations are with respect to deterministic priority algorithms. For an adaptive algorithm, the game between an algorithm and an adversary can conceptually be naturally viewed an alternating sequence of actions;

- The adversary eliminates some possible input items
- The algorithm makes a decision for the item with highest priority and chooses a new ordering for all possible remaining input items.

However, we note that for deterministic algorithms, since the adversary knows precisely what the algorithm will do in each iteation, it could initially set the input $\mathcal{I}$ once the algorithm is known.

More on provable limitations of the priority model

The above mentioned inapproximations are with respect to deterministic priority algorithms. For an adaptive algorithm, the game between an algorithm and an adversary can conceptually be naturally viewed an alternating sequence of actions;

- The adversary eliminates some possible input items
- The algorithm makes a decision for the item with highest priority and chooses a new ordering for all possible remaining input items.

However, we note that for deterministic algorithms, since the adversary knows precisely what the algorithm will do in each iteation, it could initially set the input $\mathcal{I}$ once the algorithm is known.

For randomized algorithms, there is a difference between an *oblivious adversary* that creates an initial subset $\mathcal{I}$ of items vs an *adaptive adversary* that is playing the game adaptively reacting to each decision by the algorithm. Why?

Unless stated otherwise we usually analyze randomized algorithms (for any type of algorithm) with respect to an oblivious adversary.

Interval selection and greedy priority algorithms

To illustrate the limitations of greedy algorithms (as modeled by priority algorithms), we consider the (weighted) interval selection problem. Here we are given n intervals $I_1, \ldots, I_n$ where intervals are given by $I_j = [s_j, f_j)$. I_j and I_k are conflicting if $I_j \cap I_k \neq \varnothing$. (I use half closed, half open intervals to indicate that we allow intervals to intersect just at an endpoint.) Each interval I_j has a weight or value v_j. The goal is to find a subset of non-conflicting intervals so as to maximize the sum of the values of the selected intervals.

It is well known that for the unweighted case (i.e., when $v_j = 1$ for all j) that the fixed order greedy priority algorithm Earliest Finishing Time (EFT) is optimal.

Priority algorithms for one machine interval selection continued

1. For proportional values (i.e., when $v_j = f_j - s_j$ for all j), the Longest Processing Time (LPT) greedy algorithm achieves a 3-approximation which can be proven by a charging argument. This is essentially the best possible for any deterministic priority algorithm as shown onthe following slide.
   Note: Perhaps surprinsingly for $m = 2$ machines, there is a priority 2-approximation algorithm. This algorithm can be re-stated to provide a randomized priority 2-approximation.
2. For arbitrary values $\{v_j\}$, no deterministic priority algorithm can achieve a constant approximation. More precisely, if we let $\delta_j = \frac{v_j}{f_j - s_j}$ and $\Delta = \frac{\max_j \delta_j}{\min_j \delta_j}$, then no deterministic priority algorithm can achieve ratio better than Δ. The LPT algorithm yield a 3Δ approximation for arbitrary profits.
   Note: If $\min_i \delta_i$ and $\max_i \delta_i$ are known then there is a randomized priority $O(\log \Delta)$ approximation.

A simple priority algorithm inapproximation for proportional profit

The nemisis sequence consists of long and short jobs. The long jobs are depicted in the figure. There is a small ϵ overlap between intervals on the top and bottom. In addition for each long interval of length (= value) v_i, there are three short intervals of length $\frac{v_i - 2\epsilon}{3}$ included in that long interval. The adversary will be able to create a subset of these intervals to force a bound arbitrarily close to 3 (for sufficiently small ϵ and large q).

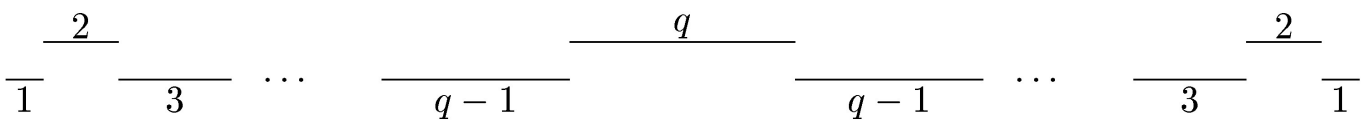

Figure: The long jobs in the nemesis input

The first interval (which must be selected) considered by the priority algorithm will allow the adversary to remove enough items to force the bound. There are four cases: I_1 is a short interval, I_1 has length 1, I_1 has length ℓ for $1 < \ell < q$, and I_1 has length q.

Extensions of the priority order model

In discussing more general online frameworks, we already implicitly suggested some extensions of the basic priority model (that is, the basic model where we have one-pass and one irrevocable decision). The following online or priority algorithm extensions can be made precise:

- Decisions can be *revocable* to some limited extent or at some cost. For example, we know that in the basic priority model we cannot achieve a constant approximation for weighted interval scheduling. However, if we are allowed to permanently discard previously accepted intervals (while always maintaining a feasible solution), then we can achieve a 4-approximation. (but provably not optimality).
- While the knapsack problem cannot be approximated to within any constant, we can achieve a 2-approximation by taking the maximum of 2 greedy algorithms. More generally we can consider some "small" number k of priority (or online) algorithms and take the best result amongst these k algorithms. The partial enumeration greedy algorithm for the makespan and knapsack problems are an example of this type of extension.

Extensions of the priority order model continued

- Closely related to the "best of k online" model is the concept of online algoitthms with "advice". (One could also study priority algorithms with advice but that has not been done to my knowledge.) There are two advice models, a model where one measures the maximum number of advice bits per input item, and a model where we are given some number ℓ of advice bits at the start of the computation. The latter model is what I will mean by "online with advice." Online (resp. priority) with ℓ advice bits is equivalent to the max of $k = 2^\ell$ online (resp. priority) model.

Extensions of the priority order model continued

- Closely related to the "best of k online" model is the concept of online algoitthms with "advice". (One could also study priority algorithms with advice but that has not been done to my knowledge.) There are two advice models, a model where one measures the maximum number of advice bits per input item, and a model where we are given some number ℓ of advice bits at the start of the computation. The latter model is what I will mean by "online with advice." Online (resp. priority) with ℓ advice bits is equivalent to the max of $k = 2^\ell$ online (resp. priority) model.
  NOTE: This model is a very permissive in that the advice bits can be a function of the entire input. Of course, in practice we want these advice bits to be "easily determined" (e.g., the number of input items, or the ratio of the largest to smallest weight/value) but in keeping with the information theoretic perspective of onine and priority algorithms, one doesn't impose any such restriction.
- There are more general parallel priority based models than "best of k" algorithms. Namely, parallel algorithms could be spawning or aborting threads (as in the nBT model to be discussed later)

Multipass algorithms

- Another model that provides improved results is to allow multiple passes (over the input items) rather than just one pass.
- This is not a well studied model but there are two relatively new noteworthy results that we will be discussing:
  1. There is deterministic 3/4 approximation for weighted Max-Sat that is achieved by two "online passes" (i.e., the input sequence is determined by an adversary) over the input sequence whereas there is evidence that no one pass deterministic online or priority algorithm can acheive this ratio.
     Note: Poloczek and Williamson use this algorithm to produce excellent results on realistic benchmarks.
  2. There is a $\frac{3}{5}$ approximation for bipartitie matching that is achieved by two online passes whereas no deterministic online or priority algorithm can do asymptotically better than a $\frac{1}{2}$ approximation.
- It is not clear how best to formalize these multi-pass algorithms. Why?

Multipass algorithms

- Another model that provides improved results is to allow multiple passes (over the input items) rather than just one pass.
- This is not a well studied model but there are two relatively new noteworthy results that we will be discussing:
  1. There is deterministic 3/4 approximation for weighted Max-Sat that is achieved by two "online passes" (i.e., the input sequence is determined by an adversary) over the input sequence whereas there is evidence that no one pass deterministic online or priority algorithm can acheive this ratio.
     Note: Poloczek and Williamson use this algorithm to produce excellent results on realistic benchmarks.
  2. There is a $\frac{3}{5}$ approximation for bipartitie matching that is achieved by two online passes whereas no deterministic online or priority algorithm can do asymptotically better than a $\frac{1}{2}$ approximation.
- It is not clear how best to formalize these multi-pass algorithms. Why? What information should we allow to convey between passes?

Greedy algorithms for the set packing problem

One of the new areas in theoretical computer science is algorithmic game theory and mechanism design and, in particular, auctions including what are known as *combinatorial auctions*. The underlying combinatorial problem in such auctions is the set packing problem.

The set packing problem

We are given n subsets $S_1, \ldots, S_n$ from a universe U of size m. In the weighted case, each subset S_i has a weight w_i. The goal is to choose a disjoint subcollection $\mathcal{S}$ of the subsets so as to maximize $\sum_{S_i \in \mathcal{S}} w_i$. In the s-set packing problem we have $|S_i| \leq s$ for all i.

- This is a well studied problem and by reduction from the max clique problem, there is an $m^{\frac{1}{2}-\epsilon}$ hardness of approximation assuming $NP \neq ZPP$. For s-set packing with constant $s \geq 3$, there is an $\Omega(s/\log s)$ hardness of approximation assuming $P \neq NP$.
- We will consider two "natural" greedy algorithms for the s-set packing problem and a non obvious greedy algorithm for the set packing problem. These greedy algorithms are all fixed order priority

The first natural greedy algorithm for set packing

Greedy-by-weight ($Greedy_{wt}$)

Sort the sets so that $w_1 \geq w_2 \ldots \geq w_n$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

- In the unweighted case (i.e. $\forall i, w_i = 1$), this is an online algorithm.
- In the weighted (and hence also unweighted) case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- The approximation bound can be shown by a charging argument where the weight of every set in an optimal solution is charged to the first set in the greedy solution with which it intersects.

The second natural greedy algorithm for set packing

Greedy-by-weight-per-size

Sort the sets so that $w_1/|S_1| \geq w_2/|S_2| \ldots \geq w_n/|S_n|$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

- In the weighted case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- For both greedy algorithms, the approximation ratio is tight; that is, there are examples where this is essentially the approximation. In particular, these algorithms only provide an m-approximation where $m = |U|$.
- We usually assume $n >> m$ and note that by just selecting the set of largest weight, we obtain an n-approximation. So the goal is to do better than $\min\{m, n\}$.

Improving the approximation for set packing

- In the unweighted case, greedy-by-weight-per-size can be restated as sorting so that $|S_1| \leq |S_2| \ldots \leq |S_n|$ and it can be shown to provide an $\sqrt{m}$-approximation for unweighted set packing.
- On the other hand, greedy-by-weight-per-size does not improve the m-approximation for weighted set packing.

Greedy-by-weight-per-squareroot-size

Sort the sets so that $w_1/\sqrt{|S_1|} \geq w_2/\sqrt{|S_2|} \ldots \geq w_n/\sqrt{|S_n|}$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

Theorem: Greedy-by-weight-per-squareroot-size provides a $2\sqrt{m}$-approximation for the set packing problem. And as noted earlier, this is asymptotically the best possible approximation assuming $NP \neq ZPP$.

Another way to obtain an $O(\sqrt{m})$ approximation

There is another way to obtain the same aysmptototic improvement for the weighted set packing problem. Namely, we can use the idea of partial enumeration greedy; that is somehow combining some kind of brute force (or naive) approach with a greedy algorithm.

Partial Enumeration with Greedy-by-weight ($PGreedy_k$)

Let Max_k be the best solution possible when restricting solutions to those containing at most k sets. Let G be the solution obtained by $Greedy_{wt}$ applied to sets of cardinality at most $\sqrt{m/k}$. Set $PGreedy_k$ to be the best of Max_k and G.

- Theorem: $PGreedy_k$ achieves a $2\sqrt{m/k}$-approximation for the weighted set packing problem (on a universe of size m)
- In particular, for $k = 1$, we obtain a $2\sqrt{m}$ approximation and this can be improved by an arbitrary constant factor $\sqrt{k}$ at the cost of the brute force search for the best solution of cardinality k; that is, at the cost of say n^k.

$(k+1)$-claw free graphs: generalizing s-set packing

A graph $G = (V, E)$ is $(k+1)$-claw free if for all $v \in V$, the induced subgraph of $Nbhd$(v) has at most k independent vertices (i.e. does not have a $k+1$ claw as an induced subgraph).

$(k+1)$-claw free graphs abstract a number of interesting applications.

- In particular, we are interested in the (weighted) maximum independent set problem (W)MIS for $(k+1)$-claw free graphs. Note that it is hard to approximate the MIS for an arbiitrary n node graph to within a factor $n^{1-\epsilon}$ for any $\epsilon > 0$.
- We can (greedily) k-approximate WMIS for $(k+1)$-claw free graphs.
- The (weighted) s-set packing problem is an instance of (W)MIS on $s+1$-claw free graphs. What algorithms generalize?
- There are many types of graphs that are $k+1$ claw free for small k; in particular, the intersection graph of translates of a convex object in the two dimensional plane is a 6-claw free graph. For rectangles, the intersection graph is 5-claw free.

Vertex cover: where (again) the "natural greedy" is not best

- We consider another example (weighted vertex cover) where the "natural greedy algorithm" does not yield a good approximation.
- The vertex cover problem: Given node weighted graph $G = (V, E)$, with node weights $w(v), v \in V$.
  Goal: Find a subset $V' \subset V$ that covers the edges (i.e. $\forall e = (u, v) \in E$, either u or v is in V') so as to mininize $\sum_{v \in V'} w(v)$.
- Even for unweighted graphs, the problem is known to be NP-hard to obtain a 1.3606 approximation and under another (not so universally believed) conjecture (UGC) one cannot obtain a $2 - \epsilon$ approximation.
- For the unweighted problem, there are simple 2-approximation greedy algorithms such as just taking V' to be any maximal matching.
- The set cover problem is as follows: Given a weighted collection of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ over an n element universe U with set weights $w(S_i)$.
  Goal: Find a subcollection $\mathcal{S}'$ that covers the universe so as to minimize $\sum_{S_i \in \mathcal{S}'} w(S_i)$.

The natural greedy algorithm for weighted set cover

Natural greedy algorithm for set cover

$\mathcal{S}' = \varnothing$

While there are uncovered elements in the universe U

Let $j = argmin_i\{w(S_i)/|S_i \cap U|$

$\mathcal{S}' = \mathcal{S}' \cup \{S_i\}$

$U = U \setminus \{S_i\}$

End While

- The set cover problem is one of the first NP-complete problems.
- Johnson[1974] and Lovasz[1975] independently showed that this natural greedy provides a $H(n) \approx \ln n$ approximation for the unweighted case. This was extended by Chvatal[1979] to the weighted case.
- Under a reasonable complexity assumption, Feige[1979] showed that it was not possile to acheive a $(1 - \epsilon) \ln n$ approximation even for the unweighted case.

The natural greedy algorithm for weighted vertex cover (WVC)

If we consider vertex cover as a special case of set cover (how?), then the natural greedy (which is essentially optimal for set cover) becomes the following:

$d'(v) := d(v)$ for all $v \in V$
% $d'(v)$ will be the residual degree of a node
While there are uncovered edges
Let v be the node minimizing $w(v)/d'(v)$
Add v to the vertex cover;
remove all edges in $Nbhd(v)$;
recalculate the residual degree of all nodes in $Nbhd)v)$
End While

Figure: Natural greedy algorithm for weighted vertex cover. Approximation ratio $H_n \approx \ln n$ where $n = |V|$.

Clarkson's [1983] modified greedy for WVC

$d'(v) := d(v)$ for all $v \in V$

% $d'(v)$ will be the residual degree of a node

$w'(v) := w(v)$ for all $v \in V$

% $w'(v)$ will be the residual weight of a node

While there are uncovered edges

Let v be the node minimizing $w'(v)/d'(v)$

$w := w'(v)/d'(v)$

$w'(u) := w'(u) - w$ for all $u \in Nbhd(v)$

% For analysis only, set $we(u, v) = w$

Add v to the vertex cover;

remove all edges in $Nbhd(v)$;

recalculate the residual degree of all nodes in $Nbhd(v)$

End While

Figure: Clarkson's greedy algorithm for weighted vertex cover. Approximation ratio 2. Invariant: $w(v) = w'(v) + sum_{e \in E} we(e)$

Extending problems and extending the priority paradigm

Now that we know that for arbitrary profits priority algorithms cannot achieve a constant approximation for the weighted interval selection problem (WISP), lets return to ways to extend the model and the problem.

The interval selection problem is a packng problem so that any subset of a feasible solution is a feasible solution.

We already mentioned *priority algorithms with revocable acceptances* which can be used for any packing problem and in particular yields a constant approximation of WISP and also to the following NP-hard generalization of the WISP problem.

The (weighted) job interval selection problem WJISP

A *job* is a set of intervals. In addition to the start and finishing times of each interval, we will say that intervals belong to exactly one job. A feasible set of intervals are non-intersecting (as in WISP) and there is at most one interval per job.

The $Greedy_\alpha$ algorithm for WJISP

The algorithm as stated by Erlebach and Spieksma (and called ADMISSION by Bar Noy et al) is as follows:

$S := \varnothing$ % S is the set of currently accepted intervals
Sort input intervals so that $f_1 \leq f_2 \ldots \leq f_n$
for $i = 1..n$
    $C_i :=$ min weight subset of S s.t. $(S/C_i) \cup \{I_i\}$ feasible
    if $v(C_i) \leq \alpha \cdot v(I_i)$ **then**
        $S := (S/C_i) \cup \{I_i\}$
    end if
END FOR

Figure: Priority algorithm with revocable acceptances for WJISP

The $Greedy_\alpha$ algorithm (which is not greedy by my definition) has a tight approximation ratio of $\frac{1}{\alpha(1-\alpha)}$ for WISP and $\frac{2}{\alpha(1-\alpha)}$ for WJISP.

Priority Stack Algorithms

- For packing problems, instead of immediate permanent acceptances, in the first phase of a priority stack algorithm, items (that have not been immediately rejected) can be placed on a stack. After all items have been considered (in the first phase), a second phase consists of popping the stack so as to insure feasibility. That is, while popping the stack, the item becomes permanently accepted if it can be feasibly added to the current set of permanently accepted items; otherwise it is rejected. Within this priority stack model (which models a class of primal dual with reverse delete algorithms and a class of local ratio algorithms), the weighted interval selection problem can be computed optimally.
- For covering problems (such as min weight set cover and min weight Steiner tree), the popping stage is insure the minimality of the solution; that is, while popping item I from the stack, if the current set of permanently accepted items plus the items still on the stack already consitute a solution then I is deleted and otherwise it becomes a permanently accepted item.

Chordal graphs and perfect elimination orderings

An interval graph is an example of a chordal graph. There are a number of equivalent definitions for chordal graphs, the standard one being that there are no induced cycles of length greater than 3.

We shall use the characterization that a graph $G = (V, E)$ is chordal iff there is an ordering of the vertices $v_1, \ldots, v_n$ such that for all i, $Nbdh(v_i) \cap \{v_{i+1}, \ldots, v_n\}$ is a clique. Such an ordering is called a perfect elimination ordering (PEO).

It is easy to see that the interval graph induced by interval intersection has a PEO (and hence is chordal) by ordering the intervals such that $f_1 \leq f_2 \ldots \leq f_n$. Using this ordering we know that there is a greedy (i.e. priority) algorithm that optimally selects a maximum size set of non intersecting intervals. The same algorithm (and proof by charging argument) using a PEO for any chordal graph optimally solves the unweighted MIS problem. The following priority stack algorithm provides an optimal solution for the WMIS problem on chordal graphs.

The optimal priority stack algorithm for the weighted max independent set problem (WMIS) in chordal graphs

$Stack := \varnothing$ % $Stack$ is the set of items on stack
Sort nodes as in a PEO.
For $i = 1..n$
    $C_i :=$ nodes on stack that are adjacent to v_i
    If $w(v_i) > w(C_i)$ **then** push v_i onto stack, **else** reject
End For
$S := \varnothing$ % S will be the set of accepted nodes
While $Stack \neq \varnothing$
    Pop next node v from $Stack$
    If v is not adjacent to any node in S, **then** $S := S \cup \{v\}$
End While

Figure: Priority stack algorithm for chordal WMIS

A k-PEO and inductive k-independent graphs

- An alternative way to describe a PEO is to say that $Nbhd(v_i) \cap v_{i+1}, \ldots, v_n\}$ has independence number 1.
- We can generalize this to a k-PEO by saying that $Nbhd(v_i) \cap v_{i+1}, \ldots, v_n\}$ has independence number at most k.
- We will say that a graph is an inductive k-independent graph if it has a k-PEO.
- Inductive k-independent graphs generalize both chordal graphs and $k+1$-claw free graphs.
- The intersection graph induced by the JISP problem is an inductive 2-independent graph.
- Using a k-PEO, a fixed-order priority algorithm (resp. a priority stack algorithm) is a k-approximation algorithm for MIS (resp. for WMIS) wrt inductive k-independent graphs.