NP-completeness in practice

Abhiram Ranade

April 6, 2016

A scheduling problem

A scheduling problem

Scheduling with release times, processing times and deadlines: As input we have R[1..n], P[1..n], D[1..n], where R[i] is the time when job i becomes available, P[i] is the time it requires to be processed, D[i] is the time when it must finish. Assuming only one job can be processed at a time, can all jobs be finished by their deadlines?

A scheduling problem

Scheduling with release times, processing times and deadlines: As input we have R[1..n], P[1..n], D[1..n], where R[i] is the time when job i becomes available, P[i] is the time it requires to be processed, D[i] is the time when it must finish. Assuming only one job can be processed at a time, can all jobs be finished by their deadlines?

Show that SRPD(R,P,D) is NPC.

A scheduling problem

Scheduling with release times, processing times and deadlines: As input we have R[1..n], P[1..n], D[1..n], where R[i] is the time when job i becomes available, P[i] is the time it requires to be processed, D[i] is the time when it must finish. Assuming only one job can be processed at a time, can all jobs be finished by their deadlines?

Show that SRPD(R,P,D) is NPC.

Note: polytime if no release times, or deadlines.

SRPD $\in$ NP

SRPD $\in$ NP

Certificate = starting time y[i] of each job

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.
  Let y[i] = time at which job i starts.

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.
  Let y[i] = time at which job i starts.
  For this y, all checks will succeed.

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.
  Let y[i] = time at which job i starts.
  For this y, all checks will succeed.
- Suppose satisfying y exists.

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.
  Let y[i] = time at which job i starts.
  For this y, all checks will succeed.
- Suppose satisfying y exists.
  Start the jobs at y[i]. This will produce a valid schedule.

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.
  Let y[i] = time at which job i starts.
  For this y, all checks will succeed.
- Suppose satisfying y exists.
  Start the jobs at y[i]. This will produce a valid schedule.
  Contrapositive: No valid schedule $\Rightarrow$ No y[i] that will cause Verify to return true.

SRPD $\in$ NP

Certificate = starting time y[i] of each job

Verify(R,P,D,y[1..n]){
For each i, check that R[i] $\leq$ y[i], y[i]+P[i] $\leq$ D[i].
For each i, check no other y[j] is between y[i], y[i]+p[i].
Iff all checks succeed return true.
}

- Suppose SRPD(R,P,D) = true.
  Let y[i] = time at which job i starts.
  For this y, all checks will succeed.
- Suppose satisfying y exists.
  Start the jobs at y[i]. This will produce a valid schedule.
  Contrapositive: No valid schedule $\Rightarrow$ No y[i] that will cause Verify to return true.
- Verify runs in polytime.

Partition $\leq_K$ SRPD

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$

s $= \sum_i a_i$

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$

s $= \sum_i a_i$

for all i=1..n: R[i] $= 0$, D[i] $= 1 + s$, P[i] $= a_i$

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$

s $= \sum_i a_i$

for all i=1..n: R[i] $= 0$, D[i] $= 1 + s$, P[i] $= a_i$

R[n+1] $= s/2$, $P[n+1] = 1$, D[n+1] $= s/2 + 1$.

Partition $\leq_K$ SRPD

Instance-map($a_1, \ldots, a_n$){
s = $\sum_i a_i$

for all i=1..n: R[i] = 0, D[i] = $1 + s$, P[i] = a_i

R[n+1] = $s/2$, $P[n+1] = 1$, D[n+1] = $s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
}

Partition $\leq_K$ SRPD

Instance-map($a_1, \ldots, a_n$){
s = $\sum_i a_i$

for all i=1..n: R[i] = 0, D[i] = $1 + s$, P[i] = a_i

R[n+1] = $s/2$, $P[n+1] = 1$, D[n+1] = $s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
}

- SS has a solution $\Rightarrow \exists$ subsets adding to $s/2$.

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$
s $= \sum_i a_i$

for all i=1..n: R[i] $= 0$, D[i] $= 1 + s$, P[i] $= a_i$

R[n+1] $= s/2$, $P[n+1] = 1$, D[n+1] $= s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
$\}$

- SS has a solution $\Rightarrow \exists$ subsets adding to $s/2$.
  Schedule tasks in one partition before time s/2 in any order.
  They will fit in the duration 0..t.

Partition $\leq_K$ SRPD

Instance-map($a_1, \ldots, a_n$){
s = $\sum_i a_i$

for all i=1..n: R[i] = 0, D[i] = $1 + s$, P[i] = a_i

R[n+1] = $s/2$, $P[n+1] = 1$, D[n+1] = $s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
}

- SS has a solution $\Rightarrow$ $\exists$ subsets adding to $s/2$.
  Schedule tasks in one partition before time s/2 in any order.
  They will fit in the duration 0..t.
  Schedule other tasks after s/2+1. They will fit in the duration t+1..$1 + \sum_i a_i$.

Partition $\leq_K$ SRPD

Instance-map($a_1, \ldots, a_n$){
s = $\sum_i a_i$

for all i=1..n: R[i] = 0, D[i] = $1 + s$, P[i] = a_i

R[n+1] = $s/2$, $P[n+1] = 1$, D[n+1] = $s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
}

- SS has a solution $\Rightarrow \exists$ subsets adding to $s/2$.
  Schedule tasks in one partition before time s/2 in any order.
  They will fit in the duration 0..t.
  Schedule other tasks after s/2+1. They will fit in the duration t+1..$1 + \sum_i a_i$.
  $\Rightarrow$ SRPD has a solution.

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$
s $= \sum_i a_i$

for all i=1..n: R[i] $= 0$, D[i] $= 1 + s$, P[i] $= a_i$

R[n+1] $= s/2$, $P[n+1] = 1$, D[n+1] $= s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
$\}$

- SS has a solution $\Rightarrow \exists$ subsets adding to $s/2$.
  Schedule tasks in one partition before time s/2 in any order.
  They will fit in the duration 0..t.
  Schedule other tasks after s/2+1. They will fit in the duration t+1..$1 + \sum_i a_i$.
  $\Rightarrow$ SRPD has a solution.
- Converse is also true.

Partition $\leq_K$ SRPD

Instance-map$(a_1, \ldots, a_n)\{$
s $= \sum_i a_i$

for all i=1..n: R[i] $= 0$, D[i] $= 1 + s$, P[i] $= a_i$

R[n+1] $= s/2$, $P[n+1] = 1$, D[n+1] $= s/2 + 1$.

Return R[1..n+1], P[1..n+1], D[1..n+1]
$\}$

- SS has a solution $\Rightarrow \exists$ subsets adding to $s/2$.
  Schedule tasks in one partition before time s/2 in any order.
  They will fit in the duration 0..t.
  Schedule other tasks after s/2+1. They will fit in the duration t+1..$1 + \sum_i a_i$.
  $\Rightarrow$ SRPD has a solution.
- Converse is also true.
- Instance map runs in polytime.

Graph Colouring

GC(G,k) : Is it possible to assign a colour c[u] to each vertex u of a graph such that $1 \leq c[u] \leq k$, and for each edge (u,v): $c[u] \neq c[v]$?

Graph Colouring

GC(G,k) : Is it possible to assign a colour c[u] to each vertex u of a graph such that $1 \leq$ c[u] $\leq$ k, and for each edge (u,v): c[u] $\neq$ c[v]?

Trivial example: Given enrollment for each course, is it possible to schedule all courses in k slots s.t. courses taken by any student are in different slots?

Graph Colouring

GC(G,k) : Is it possible to assign a colour c[u] to each vertex u of a graph such that $1 \leq c[u] \leq k$, and for each edge (u,v): $c[u] \neq c[v]$?

Trivial example: Given enrollment for each course, is it possible to schedule all courses in k slots s.t. courses taken by any student are in different slots?

Call scheduling (G,P,k): G is a graph in which each vertex is a computer, and each edge is an optical communication link. Transmissions can happen on each edge in any of k frequency bands. P is a list of paths on which connection needs to be established. Each connection must be assigned a frequency band s.t. two connections do not have the same frequency band if their paths have a common edge. Can all connections be established?

GC $\leq_K$ Call scheduling

GC $\leq_K$ Call scheduling

For each vertex u of G we will have a path p(u) in Graph G' of the call scheduling instance.

GC $\leq_K$ Call scheduling

For each vertex u of G we will have a path p(u) in Graph G' of the call scheduling instance.
We must make sure that iff (u,v) is an edge of G, then p(u),p(v) share an edge in G'.

GC $\leq_K$ Call scheduling

For each vertex u of G we will have a path p(u) in Graph G' of the call scheduling instance.
We must make sure that iff (u,v) is an edge of G, then p(u),p(v) share an edge in G'.

Instance-map(G,k){
Vertex set of G':
vertices s_u, t_u : for each vertex u of G.
vertices $(u, v, 1)$, $(u, v, 2)$ for each edge (u, v) of G.

GC $\leq_K$ Call scheduling

For each vertex u of G we will have a path p(u) in Graph G' of the call scheduling instance.
We must make sure that iff (u,v) is an edge of G, then p(u),p(v) share an edge in G'.

Instance-map(G,k){
Vertex set of G':
vertices s_u, t_u : for each vertex u of G.
vertices $(u, v, 1)$, $(u, v, 2)$ for each edge (u, v) of G.
$(v, u, 1)$ also refers to $(u, v, 1)$, and similarly $(v, u, 2)$.

GC $\leq_K$ Call scheduling

For each vertex u of G we will have a path p(u) in Graph G' of the call scheduling instance.
We must make sure that iff (u,v) is an edge of G, then p(u),p(v) share an edge in G'.

Instance-map(G,k){
Vertex set of G':
vertices s_u, t_u : for each vertex u of G.
vertices $(u, v, 1)$, $(u, v, 2)$ for each edge (u, v) of G.
$(v, u, 1)$ also refers to $(u, v, 1)$, and similarly $(v, u, 2)$.
Edges in G':
We will have edges from (u,v,1) to (u,v,2) for all u,v.
$p(u) =$ starting from s_u put edges to form a path through all ((u,v,1),(u,v,2)) edges, for all v, in any order and continue to t_u.

GC $\leq_K$ Call scheduling

For each vertex u of G we will have a path p(u) in Graph G' of the call scheduling instance.
We must make sure that iff (u,v) is an edge of G, then p(u),p(v) share an edge in G'.

Instance-map(G,k){
Vertex set of G':
vertices s_u, t_u : for each vertex u of G.
vertices $(u, v, 1)$, $(u, v, 2)$ for each edge (u, v) of G.
$(v, u, 1)$ also refers to $(u, v, 1)$, and similarly $(v, u, 2)$.
Edges in G':
We will have edges from (u,v,1) to (u,v,2) for all u,v.
$p(u)$ = starting from s_u put edges to form a path through all ((u,v,1),(u,v,2)) edges, for all v, in any order and continue to t_u.

Return G', $\{p(u)\}$, k.
}

Hamiltonian Cycle

Hamiltonian Cycle

HC(G) : Does G contain a cycle that passes through every vertex exactly once?

Hamiltonian Cycle

HC(G) : Does G contain a cycle that passes through every vertex exactly once?

Roundtable seating problem: Given information about which pairs in a group of people are friends, is there a way to seat them at a roundtable so that each person sits next to a friend?

Hamiltonian Cycle

HC(G) : Does G contain a cycle that passes through every vertex exactly once?

Roundtable seating problem: Given information about which pairs in a group of people are friends, is there a way to seat them at a roundtable so that each person sits next to a friend?

G : vertices = people, edges : connect friends.

Remarks

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.
- Some problems are generalizations of NP-complete problems.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.
- Some problems are generalizations of NP-complete problems.

  Example: Bin packing. Choosing 2 bins and bin capacities suitably gave us partition.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.
- Some problems are generalizations of NP-complete problems.

  Example: Bin packing. Choosing 2 bins and bin capacities suitably gave us partition.

  Example: Set cover. Choosing sets = vertices, edges = elements gives VC.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.
- Some problems are generalizations of NP-complete problems.

  Example: Bin packing. Choosing 2 bins and bin capacities suitably gave us partition.

  Example: Set cover. Choosing sets = vertices, edges = elements gives VC.
- Gadget reductions: We have to form some structure using the rhs problem to represent structure in the lhs problem. These can be very creative.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.
- Some problems are generalizations of NP-complete problems.

  Example: Bin packing. Choosing 2 bins and bin capacities suitably gave us partition.

  Example: Set cover. Choosing sets = vertices, edges = elements gives VC.
- Gadget reductions: We have to form some structure using the rhs problem to represent structure in the lhs problem. These can be very creative.

  CNFSAT to IS reduction.

Remarks

- Many real life problems, when expressed abstractly, are seen directly to be NP-complete problems.

  These are the easiest reductions. But you cannot overlook them.
- Some problems are generalizations of NP-complete problems.

  Example: Bin packing. Choosing 2 bins and bin capacities suitably gave us partition.

  Example: Set cover. Choosing sets = vertices, edges = elements gives VC.
- Gadget reductions: We have to form some structure using the rhs problem to represent structure in the lhs problem. These can be very creative.

  CNFSAT to IS reduction.

  SS to SRPD reduction (relatively simple)

Some thumb rules on what to reduce from

Some thumb rules on what to reduce from

You want to prove $Q \in$ NPC. What R to reduce from?

Some thumb rules on what to reduce from

You want to prove Q $\in$ NPC. What R to reduce from?

- Does Q involves picking entities avoiding pairwise conflicts? Consider R $=$ IS.

Some thumb rules on what to reduce from

You want to prove $Q \in NPC$. What R to reduce from?

- Does Q involves picking entities avoiding pairwise conflicts? Consider $R = IS$.
- Does Q involve partitioning into several groups to avoid conflicts? Consider $R = GC$.

Some thumb rules on what to reduce from

You want to prove $Q \in NPC$. What R to reduce from?

- Does Q involves picking entities avoiding pairwise conflicts? Consider R = IS.
- Does Q involve partitioning into several groups to avoid conflicts? Consider R = GC.
- Does Q involve entities, each of which serves some goals, and you want to pick fewest entities so as to satisfy ("cover") all goals? Consider R = VC (or set cover).

Some thumb rules on what to reduce from

You want to prove Q $\in$ NPC. What R to reduce from?

- Does Q involves picking entities avoiding pairwise conflicts? Consider R = IS.
- Does Q involve partitioning into several groups to avoid conflicts? Consider R = GC.
- Does Q involve entities, each of which serves some goals, and you want to pick fewest entities so as to satisfy ("cover") all goals? Consider R = VC (or set cover).
- Does Q involve ordering a set of objects? Consider R = HC.

Some thumb rules on what to reduce from

You want to prove $Q \in NPC$. What R to reduce from?

- Does Q involves picking entities avoiding pairwise conflicts? Consider $R = IS$.
- Does Q involve partitioning into several groups to avoid conflicts? Consider $R = GC$.
- Does Q involve entities, each of which serves some goals, and you want to pick fewest entities so as to satisfy ("cover") all goals? Consider $R = VC$ (or set cover).
- Does Q involve ordering a set of objects? Consider $R = HC$.
- Does Q involve acheiving numerical targets? Consider $R = SS$.

Some thumb rules on what to reduce from

You want to prove Q ∈ NPC. What R to reduce from?

- Does Q involves picking entities avoiding pairwise conflicts? Consider R = IS.
- Does Q involve partitioning into several groups to avoid conflicts? Consider R = GC.
- Does Q involve entities, each of which serves some goals, and you want to pick fewest entities so as to satisfy ("cover") all goals? Consider R = VC (or set cover).
- Does Q involve ordering a set of objects? Consider R = HC.
- Does Q involve acheiving numerical targets? Consider R = SS.
- Does Q involving picking subsets that cover the universe and are disjoint? Consider R = 3 dimensional matching.

If you cannot reduce from R as suggested above, mimic the reduction that was used to prove R to be NP-hard.