APTAS for Bin Packing

Bin Packing has an asymptotic PTAS (APTAS) [de la Vega and Leuker, 1980]

For every fixed $\varepsilon > 0$ algorithm outputs a solution of size $(1+\varepsilon)OPT + 1$ in time polynomial in $n$
APTAS for Bin Packing

Split items into large and small
item $i$ is large if $s_i \geq \varepsilon$, otherwise small

Pack large items
Pack small items using greedy on top of large items
Packing large items: shifting trick

**Idea:** change instance such that # of distinct sizes is constant. Can then solve problem using dynamic programming

\[ \text{OPT} \geq s(L) = \sum_{i \in L} s_i \geq \varepsilon |L| \]

\( L \): large items
Grouping large items

For any integer $1 \leq k \leq \frac{|L|}{k}$, can partition $L$ into $L_1, L_2, \ldots, L_k$ such that:

- for $1 < i \leq k$, items in $L_i$ are all smaller than the smallest item in $L_{i+1}$
- $|L_1| \leq |L_2| = |L_3| = \ldots = |L_k|$

Sort items and pick $L_k = \text{largest } \frac{|L|}{k}$ items, $L_{k-1}$ the next largest $\frac{|L|}{k}$ items and so on
Shifting

\( a_i \) : size of smallest item in \( L_i \)

For \( i = 1 \) to \( k-1 \) do
  \[ \text{if } j \in L_i, \text{ set } s'_j = a_{i+1} \]

Let \( L' = L_1 \cup L_2 \cup ... \cup L_{k-1} \) with new sizes

Claim: \( L' \) can be packed in \( \text{OPT} \) number of bins

Claim: \# of distinct sizes in \( L' \) is \( k-1 \)
Packing large items

$L'$ can be packed in $\text{OPT}$ bins in $O(n^{2k})$ time using dynamic programming (can do better using other methods)

Pack $L_k$ in $|L_k|$ bins - each item separately

# of bins used is $\text{OPT} + |L_k| \leq \text{OPT} + 2|L|/k$

Choose $k = 2/\varepsilon^2$ (assuming $|L| \geq k^2$)
Packing large items

L’ can be packed in OPT bins in $O(n^{2k})$ time using dynamic programming.

Number of bins used is $OPT + |L_k| \leq OPT + 2|L|/k$

Choose $k = 2/\epsilon^2$

Number of bins used is $OPT + \epsilon^2 |L| \leq OPT + \epsilon OPT$ (since $OPT \geq \epsilon |L|$)

$\leq (1+\epsilon) OPT$

Running time is $O(n^{4/\epsilon^2})$
Packing large items

Suppose $|L| \leq 4/\varepsilon^4$?

**Lemma:** Optimum solution for a Bin Packing problem can be computed in $O(n \log n 2^n)$ time.

If $|L| \leq 4/\varepsilon^4$ can compute optimum solution in $2^{O(1/\varepsilon^4)}$ time.
Packing small items

$m$: bins used to pack large items
Run greedy with small items
$m'$: total # of bins used after small item packing

Claim: $m' \leq \max \{ m, \lceil (\sum_i s_i)/(1-\epsilon) \rceil \}$

$\lceil (\sum_i s_i)/(1-\epsilon) \rceil \leq (\sum_i s_i)(1+2\epsilon) + 1$ for sufficiently small $\epsilon$
Packing small items

\( m \): bins used to pack large items

Run greedy with small items

\( m' \): total # of bins used after small item packing

Claim: \( m' \leq \max \{ m, \lceil (\sum_i s_i)/(1-\varepsilon) \rceil \} \)

Proof: if \( m' > m \) then at most one bin \((1-\varepsilon)\)-full

\( (1-\varepsilon) (m'-1) < \sum_i s_i \)
Packing small items

$m$: bins used to pack large items

Run greedy with small items

$m'$: total # of bins used after small item packing

Claim: $m' \leq \max \{ m, \left( \sum_i s_i \right)/(1-\epsilon) \}$

$m \leq (1+\epsilon) \text{OPT}$

therefore $m' \leq (1+2\epsilon) \text{OPT} + 1$ for sufficiently small $\epsilon \leq \frac{1}{2}$
Summary of APTAS

$m' \leq (1+2\varepsilon) \text{OPT} + 1$

Running time: dominated for packing large items

$n^{O(1/\varepsilon^2)} + 2^{O(1/\varepsilon^4)}$
Better algorithms for Bin Packing

[Karmarkar-Karp, 1982]: sophisticated ideas

Polytime algorithm that to pack using
OPT + log^2 OPT bins

Also AFPAS (asymptotic FPTAS): (1+ε) OPT + f(ε) bins in
time poly(n, 1/ε)

Open problem: is there a poly-time algorithm to pack using
OPT + 1 bins? OPT + c bins where c is some absolute
constant? Sub-exponential algorithm?
Multiprocessor scheduling with precedence constraints

Jobs $J_1, J_2, \ldots, J_n$ to be executed on $m$ identical processors/machine $M_1, M_2, \ldots, M_m$

Job $J_i$ has processing time/size $p_i$

Jobs have *precedence constraints* between them. $J_i \prec J_k$ implies $J_i$ cannot be done before $J_k$ completes.

Directed acyclic graph DAG $G$ encodes constraints.
Precedence constraints

Encode dependencies which prevent parallelism
Data and control dependencies
Two important applications
  - Parallel programming
  - Instruction scheduling in multi-issue processors
Scheduling Problem

Jobs to be assigned to machines
Non-preemptive schedule: once job is started cannot interrupt it

\( C_j \): finish time of job \( j \)
\( s_j = C_j - p_j \): start time of job \( j \)
\( j \) occupies time slots \( s_j+1, ..., C_j \)

\textit{minimize} \( \max_j C_j \)
Example

$m = 3$

$C_{\text{max}} = 10$

Processing times: red
List Scheduling

List: ordering of jobs according to some priority
Higher priority to jobs earlier in the list
Assume that list is a topological sort of the dag

Greedy list scheduling algorithm: schedule as early as possible

Job $i$ is *ready* at $t$ if all jobs in $\text{Pred}(i)$ completed by $t$
List Scheduling

\[
\text{for } k = 1 \text{ to } m \text{ do}
\]
\[
\quad a_k = 0 \quad // \ a_k \text{ time when } M_k \text{ is available}
\]
\[
\text{for } t = 0 \text{ to } T_{\text{max}} \text{ do}
\]
\[
\quad \begin{cases}
\text{if no jobs left to schedule, break} \\
\text{for } k = 1 \text{ to } m \text{ do}
\end{cases}
\]
\[
\quad \begin{cases}
\text{if } a_k > t \text{ continue} \\
\text{if no ready jobs, break} \\
i \leftarrow \text{highest priority ready job} \\
a_k \leftarrow t + p_i \\
\text{remove } i \text{ from list}
\end{cases}
\]
Predecessors and Successors

Take transitive closure or DAG

\[ \text{Pred}(i) = \{ j \mid j \prec i \} \] predecessors of \( i \)
\[ \text{Succ}(i) = \{ j \mid i \prec j \} \] successors of \( i \)

Chain: \( i_1 \prec i_2 \prec \ldots \prec i_k \)
Theorem: List scheduling with any list is a \(2 - \frac{1}{m}\) approximation

Analysis:

Two lower bounds:
\[
\text{OPT} \geq \sum_i \frac{p_i}{m} \text{ (average load)} \quad (LB1)
\]
\[
\text{OPT} \geq \sum_{i \in A} p_i \text{ for any chain } A \quad (LB2)
\]
\[
\text{OPT} \geq \max_{A: A \text{ chain}} p(A) \quad (LB2)
\]
List Scheduling Analysis

Two lower bounds:
\[ \text{OPT} \geq \frac{1}{m} \sum_i p_i \] (average load)
\[ \text{OPT} \geq \sum_{i \in A} p_i \text{ for any } \text{chain } A \]

\[ C_{\text{max}} = \max_j C_j \] maximum completion time in list schedule

Theorem: \[ C_{\text{max}} \leq \text{LB1} + \text{LB2} \]
List Scheduling Analysis

Theorem: \( C_{\text{max}} \leq LB1 + LB2 \)

\( t \in (0, C_{\text{max}}] \)

\( t \) is a full time slot if all machines busy in \( t \)
otherwise \( t \) is a partial time slot

\# of full time slots \( \leq \sum_i p_i/m = LB1 \)

To prove: \# of partial time slots \( \leq p(A) \) for some chain \( A \)
List Scheduling Analysis

Theorem: \( C_{\text{max}} \leq LB1 + LB2 \)

# of full time slots \( \leq \sum_i p_i/m = LB1 \)
To prove: # of partial time slots \( \leq p(A) \) for some chain \( A \)

\[
C_{\text{max}} = \# \text{ of full time slots} + \# \text{ of partial slots} \\
\leq LB1 + p(A) \leq LB1 + LB2 \\
\# \text{ of full time slots} \leq (\sum_i p_i - p(A))/m \\
\text{hence } C_{\text{max}} \leq LB1 + (1-1/m) LB2 \leq (2-1/m) \text{ OPT} \]
List scheduling analysis

Create chain inductively

\( i_1 : \) job such that \( C_{i_1} = C_{\text{max}} \) (last job to complete)
let \( t_1 = C_{i_1} \)

\( t_2 : \) max integer in \([0, s_{i_1}]\) s.t \( t_2 \) is a partial slot
if \( t_2 = 0 \) stop: chain is only \( i_1 \)

Claim: some predecessor of \( i_1 \) is executing at \( t_2 \), otherwise \( i_1 \) would have been scheduled at \( t_2 \)
List scheduling analysis

i₁ : job such that Cᵢ₁ = C_{max} (last job to complete)
let t₁ = Cᵢ₁

\( t₂: \) max integer in \([0, s_{i₁}]\) s.t \( t₂ \) is a partial slot
if \( t₂ = 0 \) stop: chain is only \( i₁ \)

i₂ : (any) predecessor of \( i₁ \) executing at \( t₂ \)
\( t₃ = \) max integer in \([0, s_{i₂}]\) s.t \( t₃ \) is a partial slot
List scheduling analysis

inductively

t_k: max integer in \([0, s_{ik}]\) s.t \(t_k\) is a partial slot
stop if \(t_k = 0\)
else
\(i_{k+1}\): predecessor of \(i_k\) that is executing at \(t_k\)

Let \(i_1 > i_2 > \ldots > i_h\) be the chain created
\(t_1 > t_2 > \ldots > t_k \geq t_{k+1} = 0\) be associated times
Note: execution intervals of jobs in chain don’t overlap
List scheduling analysis

Let $i_1 > i_2 > ... > i_h$ be the chain created
$t_1 > t_2 > ... > t_k \geq t_{k+1} = 0$ be associated times

Claim: for $1 \leq j \leq k$

1. all slots in $[t_{j+1}+1, s_{ij}]$ are full
2. $i_j$ executing in $[s_{ij}+1, t_j]

All slots that are partial can be charged to some interval in which a job from chain is executing
All slots full
Critical Path Scheduling

Create a good list: give importance to jobs that are “critical” for other jobs

\( i_1 \): job with longest chain (break ties arbitrarily)
Remove \( i_1 \) from DAG

\( i_2 \): job with longest chain
Remove \( i_2 \) from DAG

...
Critical Path Scheduling

Works very well in practice

Several rules to break ties

However there are examples which show that critical path scheduling does not improve 2-approx (irrespective of how you break ties)
More on scheduling

Open problem: obtain $2-\varepsilon$ approx ($\geq$ 35 years) even for the case when jobs are all unit size

For unit job sizes

Theorem: Unless $P=NP$ no approximation ratio better than $4/3-\varepsilon$ for any $\varepsilon > 0$

Proof: reduction from clique
Reduction from clique

Q: Given $G=(V,E)$ and integer $k$, does $G$ have a clique of size $k$?

NP-Complete

Given $G=(V,E)$, $k$ create a scheduling instance as follows

Two classes of jobs: graph jobs, dummy jobs

$m$ machines where $m = \frac{n(n-1)}{2} + 1$
Reduction from clique

Two classes of jobs: graph jobs, dummy jobs

|E| + |V| graph jobs, one for each edge and one for each vertex

If \( J_e \) is a job corresponding to edge \( e = uv \) then \( J_u \) and \( J_v \) are predecessors for \( J_e \)

Dummy Jobs: three sets of jobs, \( D_1, D_2, D_3 \)

\[ |D_1| = m - k \]
\[ |D_2| = m - k(k-1)/2 \]
\[ |D_3| = m - |E| + k(k-1)/2 \]
Reduction from clique

Dummy Jobs: three sets of jobs, $D_1$, $D_2$, $D_3$

$|D_1| = m - k$

$|D_2| = m - k(k-1)/2 - |V| + k$

$|D_3| = m - |E| + k(k-1)/2$

Each job in $D_1$ is a predecessor to each job in $D_2$
Each job in $D_2$ is a predecessor to each job in $D_3$

Clearly $OPT \geq 3$ from dummy jobs
Reduction from clique

Claim: If $G$ has a clique of size $k$ then $\text{OPT} = 3$

Dummy jobs have to be scheduled in order
All $D_1$ in slot 1, all $D_2$ in slot 2, all $D_3$ in slot 3

Schedule clique vertices in slot 1
Schedule edges of clique in slot 2 + schedule remaining vertices in slot 2
Schedule remaining edges in slot 3
Claim: If $\text{OPT} = 3$ then $G$ has a clique of size $k$

Essentially the same as previous claim.

Therefore if $G$ does not have a clique of size $k$ then $\text{OPT} \geq 4$

Implies that $\frac{4}{3} - \varepsilon$ approx not possible unless $P=NP$
Open Problem

For unit jobs and \( m \) fixed (independent of # of jobs) it is unknown whether the problem is NP-Complete even for \( m=3 \! \)!

Open for more than 30 years

Obtain \( 2-\varepsilon \) approximation for \( m \) fixed