Using Decision Procedures Efficiently for Optimization

Matthew Streeter and Stephen F. Smith
Carnegie Mellon University
Introduction

• Optimization problems can be solved by asking a decision procedure questions of the form “is there a solution of cost \( \leq k \)” (e.g., SATPLAN, MAXPLAN)

• Many possible strategies for determining what question to ask next:
  • ramp-up (SATPLAN)
  • ramp-down (MAXPLAN)
  • geometric (Rintanen’04)

• Which is best?
Motivations

- Query strategy can dramatically affect the time needed to find an approximately optimal solution.

Time required by siege (SAT solver used by SATPLAN) to determine if there exists a plan of length $\leq k$.

$\tau(k)$

- ≥ 100 hours
Query Strategies

• A query \((k,t)\) runs the decision procedure with time limit \(t\), and asks it “is there a solution of cost \(\leq k\)?” Result can be “yes”, “no”, or “timeout”.

• A query strategy determines the next query to execute, as a function of the results of previous queries.
Query Strategies

- A query \((k,t)\) runs the decision procedure with time limit \(t\), and asks it “is there a solution of cost \(\leq k\)?” Result can be “yes”, “no”, or “timeout”.

- A query strategy determines the next query to execute, as a function of the results of previous queries.

Notation:

- \(T(k)\) = time required by decision proc. on input \(k\)
- \(\text{OPT}\) = minimum solution cost
Metrics & Assumptions
Metrics & Assumptions

- Performance metric: worst-case competitive ratio. Equals max, over all $k$, of

  \[
  \tau(k)
  \]

  time required to prove $k \leq \text{OPT}$ or $k \geq \text{OPT}$
Metrics & Assumptions

- Performance metric: worst-case competitive ratio.
  Equals max, over all \( k \), of
  \[
  \text{time required to prove } k \leq \text{OPT or } k \geq \text{OPT}
  \]
  \( \tau(k) \)

- Without any assumptions about \( \tau(k) \), can’t do better than trying all \( k \)-values in parallel.
  Competitive ratio = \#(possible \( k \)-values)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

> 100 hours —

1 second —
Metrics & Assumptions

• Performance metric: worst-case competitive ratio. Equals max, over all $k$, of
  
  \[
  \text{time required to prove } k \leq \text{OPT or } k \geq \text{OPT}
  \]

\[\tau(k)\]

• Without any assumptions about $\tau(k)$, can’t do better than trying all $k$-values in parallel. Competitive ratio = #(possible $k$-values)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> 100 hours —

1 second —

• We’ll assume $\tau(k)$ is (approximately) increasing-then-decreasing
Query strategy $S_2$
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat

![Graph showing the query strategy $S_2$](image)
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- Initialize $T \leftarrow 1$
- Use two-sided binary search to find range of $k$-values such that $\tau(k) > T$
- Double $T$ and repeat
Query strategy $S_2$

- **Theorem:** if $\tau(k)$ is increasing-then-decreasing, then $S_2$ has competitive ratio $O(\log \#(\text{possible } k\text{-values}))$
Theorem: if $\tau(k)$ is increasing-then-decreasing, then $S_2$ has competitive ratio $O(\log \#(\text{possible } k\text{-values}))$

If $\tau(k)$ becomes increasing-then-decreasing after multiplying each $\tau(k)$ by a factor $\leq \Delta$, ratio goes up by factor $\leq \Delta$
Experiments: planning

• Created modified version of SATPLAN that uses $S_2$.

• Ran both versions on benchmarks from ICAPS’06 planning competition, one hour time limit per benchmark.

• Also tried geometric strategy $S_g$ based on Rintanen (2004).
Experiments: planning

Results on instances from *pathways* domain

<table>
<thead>
<tr>
<th>Instance</th>
<th>SATPLAN (S₂) [lower,upper]</th>
<th>SATPLAN (geom.) [lower,upper]</th>
<th>SATPLAN (orig.) [lower,upper]</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>[5,5]</td>
<td>[5,5]</td>
<td>[5,5]</td>
</tr>
<tr>
<td>p2</td>
<td>[7,7]</td>
<td>[7,7]</td>
<td>[7,7]</td>
</tr>
<tr>
<td>p3</td>
<td>[8,8]</td>
<td>[8,8]</td>
<td>[8,8]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>p27</td>
<td>[19,34]</td>
<td>[20,31]</td>
<td>[20,∞]</td>
</tr>
<tr>
<td>p28</td>
<td>[19,27]</td>
<td>[20,∞]</td>
<td>[21,∞]</td>
</tr>
<tr>
<td>p29</td>
<td>[19,29]</td>
<td>[18,29]</td>
<td>[18,∞]</td>
</tr>
<tr>
<td>p30</td>
<td>[20,60]</td>
<td>[21,∞]</td>
<td>[21,∞]</td>
</tr>
</tbody>
</table>
Experiments: scheduling

- We next used $S_2$ in a branch and bound algorithm for job shop scheduling (Brucker et al. 1994).

- Here we execute query $(k,t)$ by setting upper bound to $k+1$ and seeing if problem is feasible.
### Experiments: scheduling

#### Results on instances from OR Library

<table>
<thead>
<tr>
<th>Instance</th>
<th>Brucker ($S_2$) [lower,upper]</th>
<th>Brucker (orig.) [lower,upper]</th>
</tr>
</thead>
<tbody>
<tr>
<td>abz7</td>
<td>[650,712]</td>
<td>[650,726]</td>
</tr>
<tr>
<td>abz8</td>
<td>[622,725]</td>
<td>[597,767]</td>
</tr>
<tr>
<td>abz9</td>
<td>[644,728]</td>
<td>[616,820]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>yn1</td>
<td>[813,987]</td>
<td>[763,992]</td>
</tr>
<tr>
<td>yn2</td>
<td>[835,1004]</td>
<td>[795,1037]</td>
</tr>
<tr>
<td>yn3</td>
<td>[812,982]</td>
<td>[793,1013]</td>
</tr>
<tr>
<td>yn4</td>
<td>[899,1158]</td>
<td>[871,1178]</td>
</tr>
</tbody>
</table>
Questions?