All meals for a dollar and other vertex enumeration problems

David Avis

Kyoto University and McGill University
http://cgm.cs.mcgill.ca/~avis

October 18, 2018
Vertex Enumeration

Reverse Search

Parallel Reverse Search
Outline of talk

Diet problem

- **Situation**: You need to choose some food in the supermarket to feed yourself properly for just $1 per day.
Diet problem

- **Situation**: You need to choose some food in the supermarket to feed yourself properly for just $1 per day.
- **Decision variables**: How much of each product you will buy.
Diet problem

- **Situation**: You need to choose some food in the supermarket to feed yourself properly for just $1 per day.
- **Decision variables**: How much of each product you will buy.
- **Constraints**: There are minimum daily requirements for calories, vitamins, calcium, etc. There is a maximum amount of each food you can eat.
Diet problem

- **Situation:** You need to choose some food in the supermarket to feed yourself properly for just $1 per day.
- **Decision variables:** How much of each product you will buy.
- **Constraints:** There are minimum daily requirements for calories, vitamins, calcium, etc. There is a maximum amount of each food you can eat.
- **Objective** Eat for less than $1.
## Sample data

<table>
<thead>
<tr>
<th>Food</th>
<th>Serv. Size</th>
<th>Energy (kcal)</th>
<th>Protein (g)</th>
<th>Calcium (mg)</th>
<th>Price ($)</th>
<th>Max Serv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1) Oatmeal</td>
<td>28g</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>(x_2) Chicken</td>
<td>100g</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>(x_3) Eggs</td>
<td>2 large</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>(x_4) Milk</td>
<td>237ml</td>
<td>160</td>
<td>8</td>
<td>285</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>(x_5) Cherry Pie</td>
<td>170g</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>(x_6) Pork w. beans</td>
<td>260g</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Min. Daily Amt.</td>
<td></td>
<td>2000</td>
<td>55</td>
<td>800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The decision variables are \(x_1, x_2, \ldots, x_6\).
Fractional servings are allowed.

From *Linear Programming*, Vasek Chvátal, 1983
## Linear programming formulation for diet problem

<table>
<thead>
<tr>
<th>Food</th>
<th>Serv. Size</th>
<th>Energy (kcal)</th>
<th>Protein (g)</th>
<th>Calcium (mg)</th>
<th>Price</th>
<th>Max Serv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁ Oatmeal</td>
<td>28g</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>x₂ Chicken</td>
<td>100g</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>x₃ Eggs</td>
<td>2 large</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>x₄ Milk</td>
<td>237ml</td>
<td>160</td>
<td>8</td>
<td>285</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>x₅ Cherry Pie</td>
<td>170g</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>x₆ Pork w. beans</td>
<td>260g</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Min. Daily Amt.</td>
<td></td>
<td>2000</td>
<td>55</td>
<td>800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{min } z &= 3x₁ + 24x₂ + 13x₃ + 9x₄ + 20x₅ + 19x₆ \\
\text{s.t. } & \quad 110x₁ + 205x₂ + 160x₃ + 160x₄ + 420x₅ + 260x₆ \geq 2000 \\
& \quad 4x₁ + 32x₂ + 13x₃ + 8x₄ + 4x₅ + 14x₆ \geq 55 \\
& \quad 2x₁ + 12x₂ + 54x₃ + 285x₄ + 22x₅ + 80x₆ \geq 800 \\
& \quad 0 \leq x₁ \leq 4, \quad 0 \leq x₂ \leq 3, \quad 0 \leq x₃ \leq 2, \\
& \quad 0 \leq x₄ \leq 8, \quad 0 \leq x₅ \leq 2, \quad 0 \leq x₆ \leq 2
\end{align*}
\]
Linear programming solution

<table>
<thead>
<tr>
<th>Food</th>
<th>Serv. Size</th>
<th>Energy (kcal)</th>
<th>Protein (g)</th>
<th>Calcium (mg)</th>
<th>Price ¢</th>
<th>Max Serv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 ) Oatmeal</td>
<td>28g</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( x_2 ) Chicken</td>
<td>100g</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>( x_3 ) Eggs</td>
<td>2 large</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>( x_4 ) Milk</td>
<td>237ml</td>
<td>160</td>
<td>8</td>
<td>285</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>( x_5 ) Cherry Pie</td>
<td>170g</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>( x_6 ) Pork w. beans</td>
<td>260g</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Min. Daily Amt.</td>
<td></td>
<td>2000</td>
<td>55</td>
<td>800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Linear programming solution

<table>
<thead>
<tr>
<th>Food</th>
<th>Serv. Size</th>
<th>Energy (kcal)</th>
<th>Protein (g)</th>
<th>Calcium (mg)</th>
<th>Price ¢</th>
<th>Max Serv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 ) Oatmeal</td>
<td>28g</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( x_2 ) Chicken</td>
<td>100g</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>( x_3 ) Eggs</td>
<td>2 large</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>( x_4 ) Milk</td>
<td>237ml</td>
<td>160</td>
<td>8</td>
<td>285</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>( x_5 ) Cherry Pie</td>
<td>170g</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>( x_6 ) Pork w. beans</td>
<td>260g</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Min. Daily Amt.</td>
<td></td>
<td>2000</td>
<td>55</td>
<td>800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( x_1 = 4 \) (oatmeal) \( x_4 = 4.5 \) (milk) \( x_5 = 2 \) (pie) cost=92.5 ¢
Linear programming solution

<table>
<thead>
<tr>
<th>Food</th>
<th>Serv. Size</th>
<th>Energy (kcal)</th>
<th>Protein (g)</th>
<th>Calcium (mg)</th>
<th>Price</th>
<th>Max Serv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$ Oatmeal</td>
<td>28g</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$x_2$ Chicken</td>
<td>100g</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>$x_3$ Eggs</td>
<td>2 large</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>$x_4$ Milk</td>
<td>237ml</td>
<td>160</td>
<td>8</td>
<td>285</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$x_5$ Cherry Pie</td>
<td>170g</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>$x_6$ Pork w. beans</td>
<td>260g</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
</tbody>
</table>

- $x_1 = 4$(oatmeal) $x_4 = 4.5$(milk) $x_5 = 2$(pie) cost=$92.5$¢
- Where are the chicken, eggs and pork?
### Linear programming solution

<table>
<thead>
<tr>
<th>Food</th>
<th>Serv. Size</th>
<th>Energy (kcal)</th>
<th>Protein (g)</th>
<th>Calcium (mg)</th>
<th>Price</th>
<th>Max Serv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>oatmeal</td>
<td>28g</td>
<td>110</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>chicken</td>
<td>100g</td>
<td>205</td>
<td>32</td>
<td>12</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>eggs</td>
<td>2 large</td>
<td>160</td>
<td>13</td>
<td>54</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>milk</td>
<td>237ml</td>
<td>160</td>
<td>8</td>
<td>285</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>cherry pie</td>
<td>170g</td>
<td>420</td>
<td>4</td>
<td>22</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>pork w. beans</td>
<td>260g</td>
<td>260</td>
<td>14</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
</tbody>
</table>

- $x_1 = 4 \text{(oatmeal)}$, $x_4 = 4.5 \text{(milk)}$, $x_5 = 2 \text{(pie)}$, cost = $92.5$¢
- Where are the chicken, eggs and pork?
- Do I have to eat the same food every day?
Problems with the solution

- Many desirable items were not included in the optimum solution.
- We obtained a unique optimum solution, but ...
- ... people (and managers) like to make choices!
- Ask the right question!

What are all the meals I can eat for at most $1?
Problems with the solution

- Many desirable items were not included in the optimum solution

... we obtained a unique optimum solution, but ...

... people (and managers) like to make choices!

Ask the right question!

What are all the meals I can eat for at most $1?
Problems with the solution

• Many desirable items were not included in the optimum solution
• We obtained a unique optimum solution, but …
Problems with the solution

- Many desirable items were not included in the optimum solution
- We obtained a unique optimum solution, but ...
- ... people (and managers) like to make choices!
Problems with the solution

- Many desirable items were not included in the optimum solution
- We obtained a unique optimum solution, but ...
- ... people (and managers) like to make choices!
- Ask the right question!
Problems with the solution

- Many desirable items were not included in the optimum solution
- We obtained a unique optimum solution, but ...
- ... people (and managers) like to make choices!
- Ask the right question!
- What are all the meals I can eat for at most $1?
All meals for a dollar

Replace the objective function by an inequality:

\[ 3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \leq 100 \]

\[ 110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \geq 2000 \]
\[ 4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \geq 55 \]
\[ 2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \geq 800 \]
\[ 0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2, \]
\[ 0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2 \]
All meals for a dollar

Replace the objective function by an inequality:

\[ 3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \leq 100 \]

\[ 110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \geq 2000 \]
\[ 4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \geq 55 \]
\[ 2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \geq 800 \]
\[ 0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2, \]
\[ 0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2 \]

• Any solution to these inequalities is a meal for under $1$
All meals for a dollar

Replace the objective function by an inequality:

\[ 3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \leq 100 \]

\[ 110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \geq 2000 \]
\[ 4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \geq 55 \]
\[ 2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \geq 800 \]
\[ 0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2, \]
\[ 0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2 \]

- Any solution to these inequalities is a meal for under $1$
- But this is just a restatement of the problem .......
All meals for a dollar

Replace the objective function by an inequality:

\[ 3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6 \leq 100 \]

\[ 110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \geq 2000 \]
\[ 4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \geq 55 \]
\[ 2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \geq 800 \]
\[ 0 \leq x_1 \leq 4, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 2, \]
\[ 0 \leq x_4 \leq 8, \quad 0 \leq x_5 \leq 2, \quad 0 \leq x_6 \leq 2 \]

• Any solution to these inequalities is a meal for under $1
• But this is just a restatement of the problem .......
• ... how do I find these solutions?
# A more useful solution

## All menus for a $1

### All (17) Extreme Solutions to the Diet Problem with Budget $1.00

<table>
<thead>
<tr>
<th>Cost</th>
<th>Oatmeal</th>
<th>Chicken</th>
<th>Eggs</th>
<th>Milk</th>
<th>Cherry Pie</th>
<th>Pork Beans</th>
</tr>
</thead>
<tbody>
<tr>
<td>92.5</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>4.5</td>
<td>2.</td>
<td>0</td>
</tr>
<tr>
<td>97.3</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.67</td>
<td>0</td>
</tr>
<tr>
<td>98.6</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>2.23</td>
<td>2.</td>
<td>1.40</td>
</tr>
<tr>
<td>100</td>
<td>1.65</td>
<td>0</td>
<td>0</td>
<td>6.12</td>
<td>2.</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>2.81</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.98</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>3.74</td>
<td>0</td>
<td>0</td>
<td>2.20</td>
<td>2.</td>
<td>1.53</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>2.18</td>
<td>1.88</td>
<td>1.62</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>2.21</td>
<td>2.</td>
<td>1.48</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>5.33</td>
<td>2.</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.42</td>
<td>0.40</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.80</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0</td>
<td>0.50</td>
<td>8.</td>
<td>0.48</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>1.88</td>
<td>2.63</td>
<td>2.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0.17</td>
<td>2.27</td>
<td>2.</td>
<td>1.24</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0.19</td>
<td>0</td>
<td>8.</td>
<td>0.58</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>0.60</td>
<td>3.73</td>
<td>2.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4.</td>
<td>1.03</td>
<td>2.21</td>
<td>2.</td>
<td>0.78</td>
<td></td>
</tr>
</tbody>
</table>
A more useful solution

<table>
<thead>
<tr>
<th>All menus for a $1</th>
</tr>
</thead>
</table>

All (17) Extreme

Solutions to the Diet Problem with Budget $1.00

<table>
<thead>
<tr>
<th>Cost</th>
<th>Oatmeal</th>
<th>Chicken</th>
<th>Eggs</th>
<th>Milk</th>
<th>Cherry</th>
<th>Pork</th>
<th>Pie</th>
<th>Beans</th>
</tr>
</thead>
<tbody>
<tr>
<td>92.5</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>4.7</td>
<td>2.</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97.3</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.67</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98.6</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>2.23</td>
<td>2.</td>
<td>1.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>1.65</td>
<td>0</td>
<td>0</td>
<td>6.12</td>
<td>2.</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>2.81</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.98</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>3.74</td>
<td>0</td>
<td>0</td>
<td>2.20</td>
<td>2.</td>
<td>1.53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>2.18</td>
<td>1.88</td>
<td>1.62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>2.21</td>
<td>2.</td>
<td>1.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>5.33</td>
<td>2.</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.42</td>
<td>0.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0</td>
<td>0</td>
<td>8.</td>
<td>0.80</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0</td>
<td>0.50</td>
<td>8.</td>
<td>0.48</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>1.88</td>
<td>2.63</td>
<td>2.</td>
<td>0.24</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0.17</td>
<td>2.27</td>
<td>2.</td>
<td>1.24</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0.19</td>
<td>0</td>
<td>8.</td>
<td>0.58</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>0.60</td>
<td>3.73</td>
<td>2.</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.</td>
<td>4.</td>
<td>1.03</td>
<td>2.21</td>
<td>2.</td>
<td>0.78</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A more useful solution

<table>
<thead>
<tr>
<th>All menus for a $1</th>
</tr>
</thead>
</table>

All (17) Extreme Solutions to the Diet Problem with Budget $1.00

<table>
<thead>
<tr>
<th>Cost</th>
<th>Oatmeal</th>
<th>Chicken</th>
<th>Eggs</th>
<th>Milk</th>
<th>Cherry</th>
<th>Pork</th>
<th>Pie</th>
<th>Beans</th>
</tr>
</thead>
<tbody>
<tr>
<td>92.5</td>
<td>4.0</td>
<td>0.0</td>
<td>4.5</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97.3</td>
<td>4.0</td>
<td>0.0</td>
<td>8.0</td>
<td>0.67</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98.6</td>
<td>4.0</td>
<td>0.0</td>
<td>2.23</td>
<td>2.0</td>
<td>1.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>1.65</td>
<td>0.0</td>
<td>6.12</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>2.81</td>
<td>0.0</td>
<td>8.0</td>
<td>0.98</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>3.74</td>
<td>0.0</td>
<td>2.20</td>
<td>2.0</td>
<td>1.53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>2.18</td>
<td>1.88</td>
<td>1.62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>2.21</td>
<td>2.0</td>
<td>1.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>5.33</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>8.0</td>
<td>0.42</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>8.0</td>
<td>0.80</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>0.50</td>
<td>8.0</td>
<td>0.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.0</td>
<td>1.88</td>
<td>2.63</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.17</td>
<td>2.27</td>
<td>2.0</td>
<td>1.24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.19</td>
<td>8.0</td>
<td>0.58</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>0.60</td>
<td>3.73</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>4.0</td>
<td>1.03</td>
<td>2.21</td>
<td>2.0</td>
<td>0.78</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Taking convex combinations of rows gives new meals
A more useful solution

- Taking convex combinations of rows gives new meals
- Eg. Taking half each of the last two rows gives a $1 meal with all foods
Example in $\mathbb{R}^3$

H-representation:

\[
\begin{align*}
1 - x_1 + x_3 & \geq 0 \\
1 - x_2 + x_3 & \geq 0 \\
1 + x_1 + x_3 & \geq 0 \\
1 + x_2 + x_3 & \geq 0 \\
-x_3 & \geq 0 
\end{align*}
\]

V-representation:

\[
\begin{align*}
v_1 = (-1, 1, 0), & \quad v_2 = (-1, -1, 0), & \quad v_3 = (1, -1, 0), \\
v_4 = (1, 1, 0), & \quad v_5 = (0, 0, -1) 
\end{align*}
\]
Two representations of a bounded polyhedron
Two representations of a bounded polyhedron

- **H-representation (Half-spaces):** \( \{ x \in R^n : Ax \leq b \} \)
Two representations of a bounded polyhedron

- **H-representation** (Half-spaces): \( \{ x \in R^n : Ax \leq b \} \)
- **V-representation** (Vertices): \( \nu_1, \nu_2, \ldots, \nu_N \) are the vertices of \( P \)

\[
x = \sum_{i=1}^{N} \lambda_i \nu_i
\]

where \( \sum_{i=1}^{N} \lambda_i = 1, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N \)
Two representations of a bounded polyhedron

- **H-representation** (Half-spaces): \( \{ x \in \mathbb{R}^n : Ax \leq b \} \)
- **V-representation** (Vertices): \( v_1, v_2, \ldots, v_N \) are the vertices of \( P \)

\[
x = \sum_{i=1}^{N} \lambda_i v_i
\]

where \( \sum_{i=1}^{N} \lambda_i = 1, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N \)

- **Vertex enumeration**: H-representation \( \Rightarrow \) V-representation
Two representations of a bounded polyhedron

- **H-representation** (Half-spaces): \( \{ x \in \mathbb{R}^n : Ax \leq b \} \)
- **V-representation** (Vertices): \( v_1, v_2, \ldots, v_N \) are the vertices of \( P \)

\[
x = \sum_{i=1}^{N} \lambda_i v_i
\]

where \( \sum_{i=1}^{N} \lambda_i = 1, \lambda_i \geq 0, i = 1, 2, \ldots, N \)

- **Vertex enumeration**: H-representation \( \Rightarrow \) V-representation
- **Convex hull problem**: V-representation \( \Rightarrow \) H-representation
Two representations of a bounded polyhedron

- **H-representation** (Half-spaces): \( \{ x \in \mathbb{R}^n : Ax \leq b \} \)
- **V-representation** (Vertices): \( v_1, v_2, \ldots, v_N \) are the vertices of \( P \)

\[
x = \sum_{i=1}^{N} \lambda_i v_i
\]

where \( \sum_{i=1}^{N} \lambda_i = 1, \quad \lambda_i \geq 0, \quad i = 1, 2, \ldots, N \)

- **Vertex enumeration**: H-representation \( \Rightarrow \) V-representation
- **Convex hull problem**: V-representation \( \Rightarrow \) H-representation
- **Solution methods**: double description (cdd) and reverse search (lrs)
Who uses vertex enumeration?
Who uses vertex enumeration?

- Wide variety of users: scientists, engineers, economists, operations researchers...
Who uses vertex enumeration?

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
Who uses vertex enumeration?

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers
Who uses vertex enumeration?

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers
- Software should be easy to install, run on standard work stations and ...
Who uses vertex enumeration?

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers
- Software should be easy to install, run on standard work stations and ...
- ... should run faster on better hardware!
Who uses vertex enumeration?

- Wide variety of users: scientists, engineers, economists, operations researchers ...
- ...who are not experts in polyhedral computation ...
- ... and not software engineers
- Software should be easy to install, run on standard work stations and ...
- ... should run faster on better hardware!
- Goal: parallelize lrs for multicore workstations using existing code
Ground states of a ternary fcc lattice model with nearest- and next-nearest-neighbor interactions

G. Ceder and G. D. Garbulsy
Department of Materials Science and Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

D. Avis
School of Computer Science, McGill University, Montreal, Quebec, Canada H3A 2A7

K. Fukuda
Graduate School of Systems Management, University of Tsukuba, Tokyo, 3-29-1 Otsuka, Bunkyo-ku, Tokyo 112, Japan
(Received 9 September 1993)

The possible ground states of a ternary fcc lattice model with nearest- and next-nearest-neighbor pair interactions are investigated by constructing an eight-dimensional configuration polytope and enumerating its vertices. Although a structure could not be constructed for most of the vertices, 31 ternary ground states are found, some of which correspond to structures that have been observed experimentally.
large problems. The drawback of the method is that many duplicates of the same vertex can be generated when degeneracy is present. While both methods successfully generated all vertices of the polytope, the double description method seems to be more appropriate for this computation because of the high degeneracy and moderate size of the inequality system. For larger systems, however, the reverse search method may become the only feasible algorithm for vertex enumeration.

III. RESULTS

The ground-state polytope we found is highly degenerate and consists of 4862 vertices in the eight-dimensional space spanned by the correlation functions. Some of the vertices found correspond to structures that can be transformed into each other by permutations of the A, B, and C species. If these are considered to be the same structure, the total number of distinct structures is
Case study: MIT problem

- Polytope: defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993, it took about 3 weeks to solve by cdd and 6 weeks by lrs (20MHz?)
- Goal: parallelize lrs for multicore workstations using existing code
- In 2012:
  - cddr+: cores=8, 368 secs
  - mplrs: cores=16, 496 secs
  - mplrs: cores=32, 99 secs

Table:

<table>
<thead>
<tr>
<th>Cores</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>368</td>
</tr>
<tr>
<td>16</td>
<td>496</td>
</tr>
<tr>
<td>32</td>
<td>99</td>
</tr>
</tbody>
</table>

- 32-core speedup of mplrs on 1993 mplrs: about 140,000 times!
Case study: MIT problem

- Polytope mit defined by 729 inequalities in 8 dimensions
Case study: MIT problem

- Polytope mit defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
Case study: MIT problem

- Polytope *mit* defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by *cdd* and 6 weeks by *lrs* (20MHz?)
Case study: MIT problem

- Polytope *mit* defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by *cdd* and 6 weeks by *lrs* (20MHz?)
- Goal: Goal: parallelize *lrs* for multicore workstations using existing code
Case study: MIT problem

- Polytope mit defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by cdd and 6 weeks by lrs (20MHz?)
- Goal: parallelize lrs for multicore workstations using existing code
- In 2012:

<table>
<thead>
<tr>
<th>cddr+</th>
<th>lrs</th>
<th>mplrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>secs</td>
<td>secs</td>
<td>cores=8 secs</td>
</tr>
<tr>
<td></td>
<td>secs</td>
<td>su 5.0 secs</td>
</tr>
<tr>
<td>368</td>
<td>496</td>
<td>99</td>
</tr>
</tbody>
</table>

Table: mai64: Opteron 6272, 2.1GHz, 64 cores, speedups(su) on lrs
Case study: MIT problem

- Polytope **mit** defined by 729 inequalities in 8 dimensions
- Output consists of 4862 vertices
- In 1993 it took about 3 weeks to solve by *cdd* and 6 weeks by *lrs* (20MHz?)
- Goal: **Goal: parallelize lrs for multicore workstations using existing code**
- In 2012:

<table>
<thead>
<tr>
<th></th>
<th>cddr+</th>
<th>lrs</th>
<th>mplrs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>secs</td>
<td>secs</td>
<td>cores=8</td>
</tr>
<tr>
<td>secs</td>
<td>368</td>
<td>496</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.0</td>
</tr>
<tr>
<td>su</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table:** *mai64:* Opteron 6272, 2.1GHz, 64 cores, speedups(su) on *lrs*

- 32-core speedup of *plrs* on 1993 *mplrs*: about 140,000 times! (processor=110 × 1300=software)
## More cores

<table>
<thead>
<tr>
<th>Name</th>
<th>lrs (mai20)</th>
<th>96 cores</th>
<th>128 cores</th>
<th>160 cores</th>
<th>192 cores</th>
<th>256 cores</th>
<th>312 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>c40</td>
<td>10002</td>
<td>329</td>
<td>247</td>
<td>203</td>
<td>179</td>
<td>134 (.44)</td>
<td>129 (.37)</td>
</tr>
<tr>
<td>perm10</td>
<td>2381</td>
<td>115</td>
<td>94</td>
<td>85</td>
<td>96</td>
<td>64 (.23)</td>
<td>61 (.20)</td>
</tr>
<tr>
<td>mit71</td>
<td>21920</td>
<td>686</td>
<td>516</td>
<td>412</td>
<td>350</td>
<td>231 (.60)</td>
<td>205 (.55)</td>
</tr>
<tr>
<td>bv7</td>
<td>9040</td>
<td>302</td>
<td>229</td>
<td>184</td>
<td>158</td>
<td>98 (.57)</td>
<td>88 (.52)</td>
</tr>
<tr>
<td>cp6</td>
<td>1774681</td>
<td>56700</td>
<td>43455</td>
<td>34457</td>
<td>28634</td>
<td>18657 (.72)</td>
<td>15995 (.69)</td>
</tr>
</tbody>
</table>

Table: efficiency = speedup/number of cores (mai cluster)
**Even more cores ...**

<table>
<thead>
<tr>
<th>Name</th>
<th>1 core</th>
<th>300 cores</th>
<th>mplrs 600 cores</th>
<th>900 cores</th>
<th>1200 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>c40</td>
<td>17755</td>
<td>89</td>
<td>49</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.66</td>
<td>.60</td>
<td>.46</td>
<td>.34</td>
</tr>
<tr>
<td>mit71</td>
<td>36198</td>
<td>147</td>
<td>80</td>
<td>63</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.82</td>
<td>.75</td>
<td>.64</td>
<td>.62</td>
</tr>
<tr>
<td>bv7</td>
<td>10594</td>
<td>48</td>
<td>27</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.73</td>
<td>.65</td>
<td>.44</td>
<td>.30</td>
</tr>
<tr>
<td>cp6</td>
<td>2400648</td>
<td>9640</td>
<td>4887</td>
<td>3278</td>
<td>2570</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>.83</td>
<td>.82</td>
<td>.81</td>
<td>.78</td>
</tr>
</tbody>
</table>

**Table:** *Tsubame2.5* at Tokyo Institute of Technology: secs/efficiency
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
- Typical Problems:
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
- Typical Problems:
  - Generate all triangulations on a given point set.
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
- Typical Problems:
  - Generate all triangulations on a given point set.
  - Generate all planar spanning trees on a given set of points.
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
- Typical Problems:
  - Generate all triangulations on a given point set.
  - Generate all planar spanning trees on a given set of points.
  - Generate all the cells or vertices of an arrangement of lines, planes, or hyperplanes.
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
- Typical Problems:
  - Generate all triangulations on a given point set.
  - Generate all planar spanning trees on a given set of points.
  - Generate all the cells or vertices of an arrangement of lines, planes, or hyperplanes.
  - Generate all vertices of a convex polyhedron
Reverse Search (A. & Fukuda, ’91)

- Space efficient technique to list unstructured discrete objects
- Typical Problems:
  - Generate all triangulations on a given point set.
  - Generate all planar spanning trees on a given set of points.
  - Generate all the cells or vertices of an arrangement of lines, planes, or hyperplanes.
  - Generate all vertices of a convex polyhedron
- Reverse search is defined by an adjacency oracle and a local search function
Reverse Search - Adjacency Oracle

- $V$ are the objects to be generated
Reverse Search - Adjacency Oracle

- $V$ are the objects to be generated
- Define graph $G = (V, E)$ by:

  - For every $v \in V$, $i = 1, 2, \ldots, \Delta$ ($\Delta$ is the maximum degree)
    
    $$\text{Adj}(v, i) = \{v' \mid vv' \in E\}$$
    
    - Otherwise

  - For every edge $vv'$ in $G$, there is a unique $i$ such that $v' = \text{Adj}(v, i)$.

  - "Similar" objects are joined by an edge.

  - Maximum degree $\Delta$ should be as small as possible.
Reverse Search - Adjacency Oracle

- \( V \) are the objects to be generated
- Define graph \( G = (V, E) \) by:
- For every \( v \in V, i = 1, 2, .., \Delta \) (maximum degree)

\[
\text{Adj}(v, i) = \begin{cases} 
  v' & \text{where } vv' \in E \\
  \emptyset & \text{otherwise}
\end{cases}
\]
Reverse Search - Adjacency Oracle

- $V$ are the objects to be generated
- Define graph $G = (V, E)$ by:
- For every $v \in V \ i = 1, 2, \ldots, \Delta$ (maximum degree)

$$Adj(v, i) = \begin{cases} v' & \text{where } vv' \in E \\ \emptyset & \text{otherwise} \end{cases}$$

- For every edge $vv'$ in $G$ there is a unique $i$ such that $v' = Adj(v, i)$. 
Reverse Search - Adjacency Oracle

- $V$ are the objects to be generated
- Define graph $G = (V, E)$ by:
  - For every $v \in V$ $i = 1, 2, \ldots, \Delta$ (maximum degree)
    $$\text{Adj}(v, i) = \begin{cases} v' & \text{where } vv' \in E \\ \emptyset & \text{otherwise} \end{cases}$$
  - For every edge $vv'$ in $G$ there is a unique $i$ such that $v' = \text{Adj}(v, i)$.
  - "Similar" objects are joined by an edge
Reverse Search - Adjacency Oracle

- $V$ are the objects to be generated
- Define graph $G = (V, E)$ by:
  - For every $v \in V$ $i = 1, 2, \ldots, \Delta$ (maximum degree)
    $$\text{Adj}(v, i) = \begin{cases} 
    v' & \text{where } vv' \in E \\
    \emptyset & \text{otherwise}
  \end{cases}$$
  - For every edge $vv'$ in $G$ there is a unique $i$ such that $v' = \text{Adj}(v, i)$.
- "Similar" objects are joined by an edge
- Maximum degree $\Delta$ should be as small as possible
Reverse Search - Local Search

• $G = (V, E)$ is the given graph
Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
Reverse Search - Local Search

• $G = (V, E)$ is the given graph
• $v^* \in V$ is a target vertex
• $f : V \mapsto V$ is a local search function s.t.:
Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \leftrightarrow V$ is a local search function s.t.:
  - $f(v^*) = v^*$
Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \mapsto V$ is a local search function s.t.:
  - $f(v^*) = v^*$
  - Iterating $f$ on any $v$ leads to $v^*$
Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \mapsto V$ is a local search function s.t.:
  - $f(v^*) = v^*$
  - Iterating $f$ on any $v$ leads to $v^*$
  - i.e. $f(f(\ldots(f(v))\ldots)) = v^*$
Reverse Search - Local Search

- \( G = (V, E) \) is the given graph
- \( v^* \in V \) is a target vertex
- \( f : V \mapsto V \) is a local search function s.t.:
  - \( f(v^*) = v^* \)
  - Iterating \( f \) on any \( v \) leads to \( v^* \)
  - I.e. \( f(f(f(..(f(v))..)) = v^* \)
- \( f \) defines a spanning tree on \( G \) rooted at \( v^* \)
Reverse Search - Local Search

- $G = (V, E)$ is the given graph
- $v^* \in V$ is a target vertex
- $f : V \leftrightarrow V$ is a local search function s.t.:
  - $f(v^*) = v^*$
  - Iterating $f$ on any $v$ leads to $v^*$
  - I.e. $f(f(f...(f(v))..)) = v^*$
- $f$ defines a spanning tree on $G$ rooted at $v^*$
- Reverse search generates this tree starting at $v^*$
Example - Problem

Problem:
Generate permutations of \( \{1, 2, \ldots, n\} \)

Input:
\( n = 4 \)

Output:

\[
(1, 2, 3, 4) \quad (1, 2, 4, 3) \quad (1, 3, 2, 4) \quad (1, 3, 4, 2) \quad (1, 4, 2, 3) \quad (1, 4, 3, 3) \\
(2, 1, 3, 4) \quad (2, 1, 4, 3) \quad (2, 3, 1, 4) \quad (2, 3, 4, 1) \quad (2, 4, 1, 3) \quad (2, 4, 3, 1) \\
(3, 1, 2, 4) \quad (3, 1, 4, 2) \quad (3, 2, 1, 4) \quad (3, 2, 4, 1) \quad (3, 4, 1, 2) \quad (3, 4, 2, 1) \\
(4, 1, 2, 3) \quad (4, 1, 3, 2) \quad (4, 2, 1, 3) \quad (4, 2, 3, 1) \quad (4, 3, 1, 2) \quad (4, 3, 2, 1)
\]
Example - Adjacency Oracle

\{\pi_1, \pi_2, ..., \pi_n\} is a permutation of \{1, 2, ..., n\}

\text{Adj}(\pi, i) = (\pi_1, \pi_2, ..., \pi_{i-1}, \pi_{i+1}, \pi_i, ... \pi_n) \text{ for } i = 1, 2, ..., n - 1.

Note: \(\Delta = n - 1\)
Example - Local Search

Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$

Target: $(1, 2, \ldots, n)$

$$f(\pi) = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_{i+1}, \pi_i, \ldots, \pi_n)$$

where $i$ is the smallest index for which $\pi_i > \pi_{i+1}$. 
Example - Reverse Search Tree
Reverse Search - Pseudocode

Algorithm 1 reverseSearch($v^*$, $\Delta$, Adj, $f$)

repeat
  $v \leftarrow v^*$
  $j \leftarrow 0$
  while $j < \Delta$ do
    $j \leftarrow j + 1$
    if $f(\text{Adj}(v, j)) = v$ then
      $v \leftarrow \text{Adj}(v, j)$
      forward step
      print $v$
      $j \leftarrow 0$
    end if
  end while
  if $v \neq v^*$ then
    $(v, j) \leftarrow f(v)$
    backtrack step
  end if
until $v = v^*$ and $j = \Delta$
Reverse search for vertex enumeration-I

- \( G = (V, E) \) is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle
- Simplex method gives a path from any vertex to the optimum
- lrs is a C implementation available on-line
Reverse search for vertex enumeration-I

- $G = (V, E)$ is defined by the vertices and edges of the polytope
Reverse search for vertex enumeration-I

- $G = (V, E)$ is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle
Reverse search for vertex enumeration-I

• $G = (V, E)$ is defined by the vertices and edges of the polytope
• Pivoting between vertices defines the adjacency oracle
• Simplex method gives a path from any vertex to the optimum vertex
Reverse search for vertex enumeration-I

- $G = (V, E)$ is defined by the vertices and edges of the polytope
- Pivoting between vertices defines the adjacency oracle
- Simplex method gives a path from any vertex to the optimum vertex
- lrs is a C implementation available on-line
Reverse search for vertex enumeration-II


(a) The “simplex tree” induced by the objective \((-\sum x_i)\).  
(b) The corresponding reverse search tree.
Reverse Search: features for parallelization

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards
- Subtrees may be enumerated independently without communication
- Subtree size may be estimated by Hall-Knuth estimator
Reverse Search: features for parallelization

- Objects generated are not stored in a database: no collisions
Reverse Search: features for parallelization

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards
Reverse Search: features for parallelization

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards
- Subtrees may be enumerated independently without communication
Reverse Search: features for parallelization

- Objects generated are not stored in a database: no collisions
- Each vertex is reported once and may be discarded afterwards
- Subtrees may be enumerated independently without communication
- Subtree size may be estimated by Hall-Knuth estimator
Extended Reverse Search

Extension to allow:

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root

Additional parameters:

- $\text{maxd}$ is the depth at which forward steps are terminated.
- $\text{mind}$ is the depth at which backtrack steps are terminated.
- $d$ is the depth of subtree root $v$. 
Extended Reverse Search

Extension to allow:

- all subtrees to be listed at some fixed depth
Extended Reverse Search

Extension to allow:

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
Extended Reverse Search

Extension to allow:

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
- Additional parameters:
Extended Reverse Search

Extension to allow:

• all subtrees to be listed at some fixed depth
• a subtree to be enumerated from its given root
• Additional parameters:
  • $maxd$ is the depth at which forward steps are terminated.
Extended Reverse Search

Extension to allow:

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
- Additional parameters:
  - $maxd$ is the depth at which forward steps are terminated.
  - $mind$ is the depth at which backtrack steps are terminated.
Extended Reverse Search

Extension to allow:

- all subtrees to be listed at some fixed depth
- a subtree to be enumerated from its given root
- Additional parameters:
  - $maxd$ is the depth at which forward steps are terminated.
  - $mind$ is the depth at which backtrack steps are terminated.
  - $d$ is the depth of subtree root $v^*$. 
Extended Reverse Search - Pseudocode

**Algorithm 2** extendedReverseSearch($v^*$, $\Delta$, $Adj$, $f$, $d$, $maxd$, $mind$)

repeat
  $v \leftarrow v^*$ $j \leftarrow 0$
  while $j < \Delta$ and $d < maxd$ do
    $j \leftarrow j + 1$
    if $f(Adj(v, j)) = v$ then
      $v \leftarrow Adj(v, j)$
      print $v$
      $j \leftarrow 0$
      $d \leftarrow d + 1$
    end if
  end while
  if $v \neq v^*$ then
    $(v, j) \leftarrow f(v)$
    $d \leftarrow d - 1$
  end if
until ($d = mind$ or $v = v^*$) and $j = \Delta$
Parallelization design parameters
Parallelization design parameters

• Users are from many disciplines and are not software engineers!
Parallelization design parameters

- Users are from many disciplines and are not software engineers!
- No special setup, extra library installation, or change of usage for users
Parallelization design parameters

- Users are from many disciplines and are not software engineers!
- No special setup, extra library installation, or change of usage for users
- Use available cores on user machine ‘automatically’
Parallelization design parameters

- Users are from many disciplines and are not software engineers!
- No special setup, extra library installation, or change of usage for users
- Use available cores on user machine ’automatically’
- Reuse existing lrs code (8,000+ lines!)
Naive Parallel Reverse Search: 3 phases

- **Phase 1: (single processor)**
  - Generate the reverse search tree $T$ down to a fixed depth $init_{\text{depth}}$.
  - Redirect output nodes and store in list $L$.
Naive Parallel Reverse Search: 3 phases

- **Phase 1: (single processor)**
  - Generate the reverse search tree $T$ down to a fixed depth $init\_depth$.
  - Redirect output nodes and store in list $L$.
- **Phase 2: (full parallelization)**
  - Schedule threads from $L$ using subtree enumeration feature.
  - Use parameter $max\_threads$ to limit number of parallel threads.
  - Direct output to shared output stream.
Naive Parallel Reverse Search: 3 phases

- **Phase 1**: (single processor)
  - Generate the reverse search tree $T$ down to a fixed depth $init_depth$.
  - Redirect output nodes and store in list $L$.

- **Phase 2**: (full parallelization)
  - Schedule threads from $L$ using subtree enumeration feature.
  - Use parameter $max_threads$ to limit number of parallel threads.
  - Direct output to shared output stream.

- **Phase 3**: (partial parallelization)
  - Wait until all children threads terminate.
Parallel Reverse Search - Pseudocode

Algorithm 3 parallelReverseSearch($v^*$, $\Delta$, $Adj$, $f$, $id$, $mt$)

\begin{verbatim}
num_threads ← 0
redirect output to a list $L$
extendedReverseSearch($v^*$, $\Delta$, $Adj$, $f$, $0$, $id$, $0$)
remove all $v \in L$ with $\text{depth}(v) < id$ and output $v$
while $L \neq \emptyset$ do
  if $\text{num_threads} < mt$ then
    remove any $v \in L$
    $\text{num_threads} ← \text{num_threads} + 1$
    extendedReverseSearch($v$, $\Delta$, $Adj$, $f$, $\text{depth}(v)$, $\infty$, $\text{depth}(v)$)
  end if
end while
while $\text{num_threads} > 0$ do
  wait for termination signal
  if $L \neq \emptyset$ then
    wait until a termination signal is received
    extendedReverseSearch($v$, $\Delta$, $Adj$, $f$, $\text{depth}(v)$, $\infty$, $\text{depth}(v)$)
  else
    $\text{num_threads} ← \text{num_threads} − 1$
  end if
end while
\end{verbatim}
A portable parallel implementation of *lrs* derived from the parallel reverse search algorithm.

**Architecture:**

- Light C++ wrapper around *lrs*.
  - Leverage *lrs*'s restart feature.
  - Use portable g++ compiler.
- Multi-producer and single consumer.
  - Producer threads traverse subtrees of the reverse search tree, appending nodes to a lock-free queue.
  - Consumer thread removes nodes from shared queue and concatenates to unified location.
- Leverage open source Boost library for atomic features.
  - Ensures portability, maintainability and strong performance.

**plrs (Implemented by Gary Roumanis)**
3 Phases: CPU utilization

Figure: Input file: mit, $id = 6$, cores=12
Estimates at depth 2: mit
Initial depth variation: mit

Figure: $id = 3$, $L = 127$, 124 secs

Figure: $id = 4$, $L = 284$, 105 secs

Figure: $id = 6$, $L = 1213$, 105 secs

Figure: $id = 10$, $L = 7985$, 125 secs
**plrs: limitations**

- **Algorithm analysis:**
  - No parallelization in Phase 1.
  - Complete parallelization in Phase 2.
  - Parallelization drops monotonically in Phase 3.

Please come back for part 2!
plrs: limitations

- **Algorithm analysis:**
  - No parallelization in Phase 1.
  - Complete parallelization in Phase 2.
  - Parallelization drops monotonically in Phase 3.

- **This leads to the following issues:**
**plrs: limitations**

- **Algorithm analysis:**
  - No parallelization in Phase 1.
  - Complete parallelization in Phase 2.
  - Parallelization drops monotonically in Phase 3.

- **This leads to the following issues:**
  - Success depends on balance of the reverse search tree.

These problems were solved in mplrs.
Algorithm analysis:
- No parallelization in Phase 1.
- Complete parallelization in Phase 2.
- Parallelization drops monotonically in Phase 3.

This leads to the following issues:
- Success depends on balance of the reverse search tree.
- Conflicting issues in setting `init_depth`. 

plrs: limitations
**plrs: limitations**

- **Algorithm analysis:**
  - No parallelization in Phase 1.
  - Complete parallelization in Phase 2.
  - Parallelization drops monotonically in Phase 3.

- **This leads to the following issues:**
  - Success depends on balance of the reverse search tree.
  - Conflicting issues in setting `init_depth`.
  - These problems were solved in mplrs.
**plrs: limitations**

- **Algorithm analysis:**
  - No parallelization in Phase 1.
  - Complete parallelization in Phase 2.
  - Parallelization drops monotonically in Phase 3.

- **This leads to the following issues:**
  - Success depends on balance of the reverse search tree.
  - Conflicting issues in setting `init_depth`.
  - These problems were solved in mplrs.
  - Please come back for part 2!