Efficient Solvers for Packing Puzzles

Tom Verhoef

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica

© 2001, Tom Verhoef (TUE)

Topics

- Packing Puzzles:
  Concrete → Abstract

- Solvers for Packing Puzzles:
  Data in backtrack program → Program for puzzle processor

- Results

© 2001, Tom Verhoef (TUE)
Simple Packing Puzzle

Pack the set of **pieces** in the **box**, without overlapping:

- **Box** (left): $2 \times 3$ unit squares
- **Pieces** (right): A, B, C

Aspects and Embeddings

- **Aspect**: Cell in box, or piece kind
- **Embedding**: Placement of piece in box

Can be encoded by set of aspects.
Aspects and Embeddings for Simple Puzzle

Aspects: \{0, 1, 2, 3, 4, 5, A, B, C\}

Embeddings:

Eliminating Symmetries by Restricting Embeddings

Restrict piece C to one orientation:

© 2001, Tom Verhoeff (TUE)  Puzzle Processor–5  Puzzle Processor–6
Abstract Puzzle

Abstract Puzzle: Pair \((A, E)\) with

- \(A\): Set of aspects
- \(E\): Set of embeddings with \(E \subseteq \mathcal{P}(A) \land E \neq \emptyset \land \emptyset \not\in E\)

Captures **topology**, ignores **(geo)metric** information

© 2001, Tom Verhoeff (TUE)

Abstract Puzzle Solution

Solution: Subset \(S\) of \(E\) that partitions \(A\):

Each aspect in \(A\) is covered **exactly once** by an embedding in \(S\)

Embeddings \(\{1, 10, 15\}\) solve the Simple Puzzle:

© 2001, Tom Verhoeff (TUE)
Deciding whether an abstract puzzle has a solution is an NP-complete problem.

Number of solutions can be exponential in size of puzzle.
Recursive Solver: Generalized Specification

Partial solution: Subset $p$ of $E$ such that

Each aspect in $A$ is covered at most once by an embedding in $p$

```plaintext
proc Solve ( p: subset of E )
{ pre: p is partial solution
  post: Solution(s) called once for each solution $s$ extending $p$
}
```

Use as: `Solve ( Ø )`

© 2001, Tom Verhoef (TUE)

Recursive Solver: Implementation

```plaintext
proc Solve ( p: subset of E )
{ vf: number of uncovered aspects: #(A − ∪p) }
[] if ∪p = A → { p is solution } Solution(p)
[] ∪p ≠ A → { there are uncovered aspects }
[] var a: A; e: E;
  a :∈ A − ∪p { a is not covered }
  { cover a in all possible ways }
; for e ∈ E with a ∈ e ∧ e ∩ ∪p = Ø do
  { p ∪ { e } is partial solution }
    Solve ( p ∪ { e } )
  od
[]
fi
```

© 2001, Tom Verhoef (TUE)
Choosing Next Aspect to Be Covered

Refine assignment \( a \in \bigcup \mathcal{P} - \mathcal{A} \), picking uncovered aspect

- **Static choice** (computed at compile-time):
  - Go through aspects in fixed order
  - How to choose order?
  - Least-Growing Footprint

- **Dynamic choice** (computed at run-time):
  - Zero-fit Cut-Off (ZCO)
  - Least-Fit First (LFF)

---

Zero-Fit Cut-Off in 6x10 Pentominoes

© 2001, Tom Verhoeff (TUE)
Least-Fit First in $6 \times 10$ Pentominoes

Hollow Pyramid Puzzle: Koos Verhoeff (198?)
## All Fit Counts for Empty $6 \times 10$ Pentominoes

<table>
<thead>
<tr>
<th>37</th>
<th>78</th>
<th>103</th>
<th>111</th>
<th>112</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>166</td>
<td>219</td>
<td>234</td>
<td>235</td>
</tr>
<tr>
<td>102</td>
<td>218</td>
<td>282</td>
<td>297</td>
<td>298</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th>304</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>248</td>
<td>128</td>
</tr>
<tr>
<td>V</td>
<td>128</td>
<td>248</td>
</tr>
<tr>
<td>L</td>
<td>248</td>
<td>128</td>
</tr>
<tr>
<td>U</td>
<td>152</td>
<td>56</td>
</tr>
<tr>
<td>F</td>
<td>256</td>
<td>32</td>
</tr>
</tbody>
</table>

© 2001, Tom Verhoeff (TUE)

## Aspect-71 Fit Counts for Empty $6 \times 10$ Pentominoes

‘Footprint’ for aspect 71 (piece X):

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P</th>
<th></th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
<td>Z</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>F</td>
<td>32</td>
<td>X</td>
</tr>
</tbody>
</table>

**Fanout:** 32  
**Total area:** 57

© 2001, Tom Verhoeff (TUE)
Aspect-0 Fit Counts for Empty $6 \times 10$ Pentominoes

‘Footprint’ for aspect 0 (upper lefthand corner cell):

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>23</td>
<td>13</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>14</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>6</td>
<td>2</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>4</td>
<td>4</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>3</td>
<td>4</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>6</td>
<td>2</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>4</td>
<td>2</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Fanout: 37     Total area: 26

© 2001, Tom Verhoeff (TUE)

Recursive Solver (Transforms 1, 2): Specification

Eliminate expression $A - \bigcup p$: introduce parameter $q = A - \bigcup p$

Eliminate parameters $p, q$: replace by global variables $p, q$

**var** $p$: subset of $E$; $q$: subset of $A$;

{ inv: $p$ is partial solution, $q = A - \bigcup p$ (uncovered aspects) }

**proc** Solve2 { **glob**: $p, q$ }

{ pre: true

  post: Solution($s$) called once for each solution $s$ extending $p$

  $p = \tilde{p} \land q = \tilde{q}$ ($p, q$ unchanged)
}

Use as: $p, q := \emptyset, A$ ; Solve2

© 2001, Tom Verhoeff (TUE)
Recursive Solver (Transforms 1, 2): Implementation

```
proc Solve2 { glob: p, q }
  [[ if q = ∅ → { p is solution } Solution(p) ]]
  [[ q ≠ ∅ → { there are uncovered aspects } ]]
  [[ var a: A; e: E; ]
    a ∈ q { a is not covered }
    { cover a in all possible ways }
  ; for e ∈ E with a ∈ e ∧ e ⊆ q do
    { p ∪ { e } is partial solution }
    p, q := p ∪ { e }, q - e
    ; Solve2 { acts on p, q }
    ; p, q := p - { e }, q ∪ e
  od
  ]]
fi
]]
```

© 2001, Tom Verhoeff (TUE)  

Recursive Solver (Transform 3): Specification

Refine choice $a :∈ q$, by introducing

```
total order ≤ on $A$, and parameter $a$ with $a ≤ \min q$
```

**var** $p$: subset of $E$; $q$: subset of $A$;
{ **inv**: $p$ is partial solution, $q = A - \cup p$ (uncovered aspects) } 

```
proc Solve3 { a: A ∪ {∞} } { glob: p, q }
  { **pre**: $a ≤ \min q$ }
  **post**: Solution($s$) called once for each solution $s$ extending $p$
  $p = \tilde{p} \land q = \tilde{q}$ ($p, q$ unchanged)
}
```

Use as: $p, q := ∅, A ; Solve2 (\min A )$

© 2001, Tom Verhoeff (TUE)
Recursive Solver (Transform 3): Implementation

```plaintext
proc Solve3 (a: A ∪ {∞}) { glob: p, q }
| if a = ∞ → { q = ∅, p is solution } Solution(p) |
| a ≠ ∞ ∧ a ∉ q → { succ(a) ≤ min q } Solve3(succ(a)) |
| a ∈ q → { a = min q, a is not covered } |
| var e: E; |
| { cover a in all possible ways } |
| for e ∈ E with a ∈ e ∧ e ⊆ q do |
| { p ∪ {e} is partial solution } |
| p, q := p ∪ {e}, q - e |
| ; Solve3 (succ(a)) |
| ; p, q := p - {e}, q ∪ e |
| od |
| fi |
| ]|
```

Further Transformations

4. Instantiate $Solve3(a)$ for each $a ∈ A ∪ \{∞\}$

5. Partition $E$ into $E_a = \{e | \min e = a\}$ with $a ∈ A$

6. Unroll the for-loops over $E_a$

7. Eliminate $p$

8. Expand each embedding $e ∈ E$ into its aspects

9. Exploit overlap among embeddings
Partitioning Example for Simple Puzzle

Transformed Program for Simple Puzzle: Solve_1

```
if ( q[1] ) { q[1] = 0;  
  if ( q[A] ) { q[A] = 0; /* embedding 1 = \{1,A\} placed */
    Solve_4 ( );
    q[A] = 1; }
  if ( q[B] ) { q[B] = 0;
    if ( q[2] ) { q[2] = 0; /* embedding 8 = \{1,2,B\} placed */
      Solve_4 ( );
      q[2] = 1; }
    if ( q[4] ) { q[4] = 0; /* embedding 9 = \{1,4,B\} placed */
      Solve_4 ( ); /* can be optimized to Solve_2 */
      q[4] = 1; }
    q[B] = 1; }
  q[1] = 1;
return; }
Solve_4 ( ); /* optimizable to "fall through" */
```
### Instruction Set of Puzzle Processor

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands and Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td><em>aspect</em> to test-and-cover: if (q[a]) { q[a] = 0; }&lt;br&gt;rel. address to jump to if aspect not free: ... }</td>
</tr>
<tr>
<td>SOLVE</td>
<td>rel. address of routine to call</td>
</tr>
<tr>
<td></td>
<td>push current address on stack</td>
</tr>
<tr>
<td>MF</td>
<td><em>aspect</em> to make free: q[a] = 1;</td>
</tr>
<tr>
<td>RETURN</td>
<td>pop address from stack and make current</td>
</tr>
<tr>
<td>SOLUTION</td>
<td>process solution encoded on stack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opcode</th>
<th>aspect</th>
<th>rel. address</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 bit</td>
<td>7 bit</td>
<td>14 bit</td>
</tr>
</tbody>
</table>

© 2001, Tom Verhoeff (TUE)

#### Puzzle-Processor Program for Simple Puzzle

```plaintext
Solve_1: IF 1, Solve_4
  IF A, L1_0 ; embedding 1 = {1,A} placed
  CALL Solve_4
  MF A
L1_0: IF B, L1_1
  IF 2, L1_2 ; embedding 8 = {1,2,B} placed
  CALL Solve_4
  MF 2
L1_2: IF 4, L1_3 ; embedding 9 = {1,4,B} placed
  CALL Solve_2
  MF 4
L1_3: MF B
L1_1: MF 1
RETURN

Solve_4: ...
```

© 2001, Tom Verhoeff (TUE)
Packing Puzzles in Benchmark

- **Pentominoes** (12 pieces of 5 squares/cubes each)
  
  2D box: 6 \times 10;  
  3D box: 3 \times 4 \times 5

  Variants: 25 Y in $5 \times 5 \times 5$, 25 N in $5 \times 5 \times 5$

- **Hollow Pyramid** (25 pieces of 4 spheres each)

- **Meteor** (10 pieces of 5 hexagons each)

© 2001, Tom Verhoeff (TUE)  

Meteor Puzzle: Christopher Monckton (1999)
### Benchmark: Aspects and Embeddings

<table>
<thead>
<tr>
<th>Puzzle</th>
<th>Aspects</th>
<th>Embeddings</th>
<th>Total Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Full</td>
<td>Restr.</td>
</tr>
<tr>
<td>Simple Puzzle</td>
<td>9</td>
<td>21</td>
<td>15</td>
</tr>
<tr>
<td>Meteor</td>
<td>60</td>
<td>2596</td>
<td>2447</td>
</tr>
<tr>
<td>6 x 10 Pentominoes</td>
<td>72</td>
<td>2056</td>
<td>1828</td>
</tr>
<tr>
<td>3 x 4 x 5 Pentominoes</td>
<td>72</td>
<td>2440</td>
<td>2062</td>
</tr>
<tr>
<td>25-Y Pentominoes</td>
<td>125</td>
<td>960</td>
<td>891</td>
</tr>
<tr>
<td>25-N Pentominoes</td>
<td>125</td>
<td>960</td>
<td>891</td>
</tr>
<tr>
<td>Hollow Pyramid</td>
<td>125</td>
<td>5632</td>
<td>4850</td>
</tr>
</tbody>
</table>

Total size = \( \sum_{e : e \in E} \#e \)

© 2001, Tom Verhoef (TUE)  Puzzle Processor–33

### Benchmark: Execution Times

<table>
<thead>
<tr>
<th>Puzzle</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Full</td>
</tr>
<tr>
<td>Simple Puzzle</td>
<td>–</td>
</tr>
<tr>
<td>Meteor</td>
<td>0.8</td>
</tr>
<tr>
<td>6 x 10 Pentominoes</td>
<td>7.9</td>
</tr>
<tr>
<td>3 x 4 x 5 Pentominoes</td>
<td>1220</td>
</tr>
<tr>
<td>25-Y Pentominoes</td>
<td>7426</td>
</tr>
<tr>
<td>25-N Pentominoes</td>
<td>2701</td>
</tr>
<tr>
<td>Hollow Pyramid</td>
<td>?</td>
</tr>
</tbody>
</table>

Times in seconds on Pentium III, 533EB

LGF = Least-Growing Footprint (others: Short-Side First)

© 2001, Tom Verhoef (TUE)  Puzzle Processor–34
Concluding Remarks

- Simple transformations yield efficient (but longer) program
- Programs involve only five operations on one boolean array
- Special-purpose puzzle processor feasible for faster execution
- Transformations are typical in compilers for embedded software
- Formal methods are indispensible
- Plenty of opportunity for further research . . .

© 2001, Tom Verhoeff (TUE)  Puzzle Processor–35

Acknowledgments

- Koos Verhoeff ('58): genes and memes
- Jan de Ruiter ('93): first solution for hollow pyramid puzzle
- Bart Tonneijk ('94): comparison of ZCO, LFF
- Erik van der Tol ('99): VLSI design for puzzle processor
- Zoltán Bálint ('00): implementation of footprint method

© 2001, Tom Verhoeff (TUE)  Puzzle Processor–36