# A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization

Sriraman Tallam  Neelam Gupta

The University of Arizona

The University of ARIZONA. Tucson Arizona

# Problem Statement

Given

- $T = \{t_1, t_2, \ldots, t_n\}$ test cases
- $R = \{r_1, r_2, \ldots, r_m\}$ testing requirements
- Testing requirements exercised by each $t_i$ (i = 1..n)

Find

- Minimum cardinality subset of T that exercises all the requirements in R exercised by test cases in T.

(NP-Complete Problem - reduction from Set Cover)

# Classical Greedy Heuristic for Set Cover

**[V. Chvatal - 1979]**

**Based on the number of requirements covered by a test case.**

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ |
|---|---|---|---|---|---|---|
| $t_1$ | X | X | X | | | |
| $t_2$ | X | | | X | | |
| $t_3$ | | X | | | X | |
| $t_4$ | | | X | | | X |
| $t_5$ | | | | | X | |

- Pick test case $t_i$ that covers most requirements.
- Throw out requirements covered by $t_i$.
- Repeat until all requirements covered.

**Minimized suite {$t_1$, $t_2$, $t_3$, $t_4$}**

**Optimal size suite {$t_2$, $t_3$, $t_4$}**

# HGS Greedy Heuristic

**[Harrold, Gupta, & Soffa - 1993]**

**Based on the number of test cases covering a requirement.**

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ |
|---|---|---|---|---|---|
| $t_1$ | X | X | | | |
| $t_2$ | X | | X | | X |
| $t_3$ | | X | X | X | |
| $t_4$ | | | X | X | |
| $t_5$ | | | | X | |
| $t_6$ | | | | | X |
| $t_7$ | | | | | X |
| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ |

**Minimized suite {$t_1$, $t_2$, $t_3$}**

**Optimal size suite {$t_2$, $t_3$}**

- Select test cases that occur in $T_i$'s of cardinality 1. Mark all $T_i$'s containing these test cases.
- Repeatedly select test case that occurs in the maximum number of $T_i$'s of cardinality 2. Mark all $T_i$'s containing these test cases.
- Repeat the process for $T_i$'s of cardinality 3, 4, …. MAX.
- In case of a tie among test cases, while considering $T_i$'s of cardinality m, test case that occurs in maximum number of unmarked $T_i$'s of cardinality m+1 is chosen.

The University of Arizona. Tucson Arizona

# Using Implications Among Requirements

- [Agarwal - 1994]

  Uses the notion of dominators and superblocks to derive *coverage implications* among the basic blocks with the goal of reducing coverage requirements for testing a program.

- [Marre and Bertolino - 2003]

  Exploits *entitiy subsumption* and use spanning trees to determine reduced set of coverage entities such that coverage of reduced set implies the coverage of unreduced set.

# Concept Analysis and Test Suite Minimization

- Test cases as *objects* and requirements as their *attributes*.
- Coverage for each test case is the *relation* between a object and its attributes.

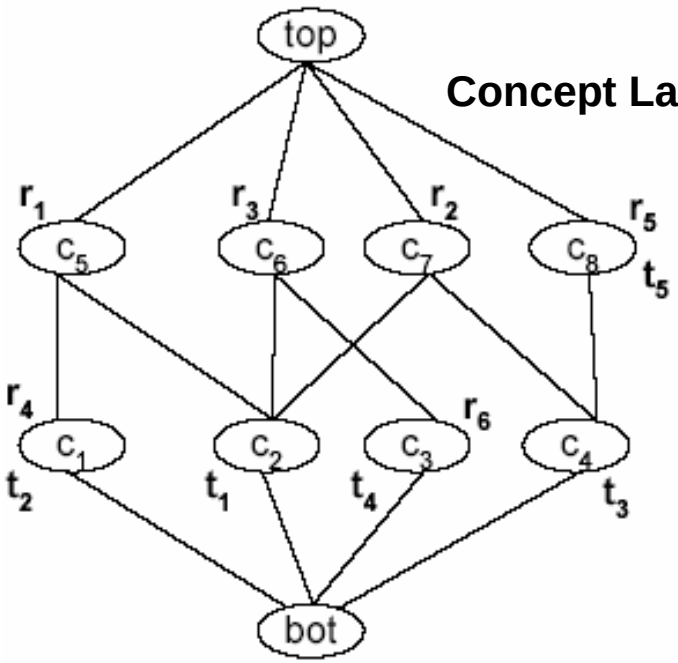
**Context table**

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ |
|---|---|---|---|---|---|---|
| $t_1$ | X | X | X | | | |
| $t_2$ | X | | | X | | |
| $t_3$ | | X | | | X | |
| $t_4$ | | | X | | | X |
| $t_5$ | | | | | X | |

**Concepts**

| Concept | (Objects,Attributes) |
|---|---|
| TOP | $(\{t_1, t_2, t_3, t_4, t_5\}, \{\})$ |
| $c_1$ | $(\{t_2\}, \{r_1, r_4\})$ |
| $c_2$ | $(\{t_1\}, \{r_1, r_2, r_3\})$ |
| $c_3$ | $(\{t_4\}, \{r_3, r_6\})$ |
| $c_4$ | $(\{t_3\}, \{r_2, r_5\})$ |
| $c_5$ | $(\{t_1, t_2\}, \{r_1\})$ |
| $c_6$ | $(\{t_1, t_4\}, \{r_3\})$ |
| $c_7$ | $(\{t_1, t_3\}, \{r_2\})$ |
| $c_8$ | $(\{t_3, t_5\}, \{r_5\})$ |
| BOT | $(\{\}, \{r_1, r_2, r_3, r_4, r_5, r_6\})$ |

**Concept Lattice**




$t_3 \Rightarrow t_5$

$r_6 \Rightarrow r_3$ and $r_4 \Rightarrow r_1$

# Reduced Context Table, Concepts and Lattice

- Applying *object reduction*: $t_3 \Rightarrow t_5$.
- Applying *attribute reductions*: $r_6 \Rightarrow r_3$ & $r_4 \Rightarrow r_1$.

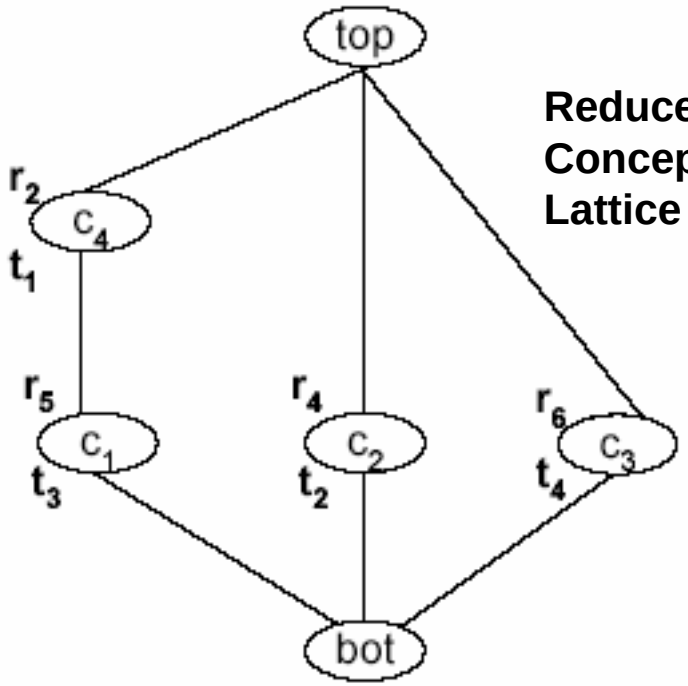
**Reduced Context Table**

| | $r_2$ | $r_4$ | $r_5$ | $r_6$ |
|---|---|---|---|---|
| $t_1$ | X | | | |
| $t_2$ | | X | | |
| $t_3$ | X | | X | |
| $t_4$ | | | | X |

**Concepts**

| Concept | (Objects,Attributes) |
|---|---|
| TOP | $(\{t_1, t_2, t_3, t_4\}, \{\})$ |
| $c_1$ | $(\{t_3\}, \{r_2, r_5\})$ |
| $c_2$ | $(\{t_2\}, \{r_4\})$ |
| $c_3$ | $(\{t_4\}, \{r_6\})$ |
| $c_4$ | $(\{t_1, t_3\}, \{r_2\})$ |
| BOT | $(\{\}, \{r_2, r_4, r_5, r_6\})$ |

**Reduced Concept Lattice**




$t_3 \Rightarrow t_1$

# Reduced Context Table, Concepts and Lattice

- Applying *object reduction*: $t_3 \Rightarrow t_1$.

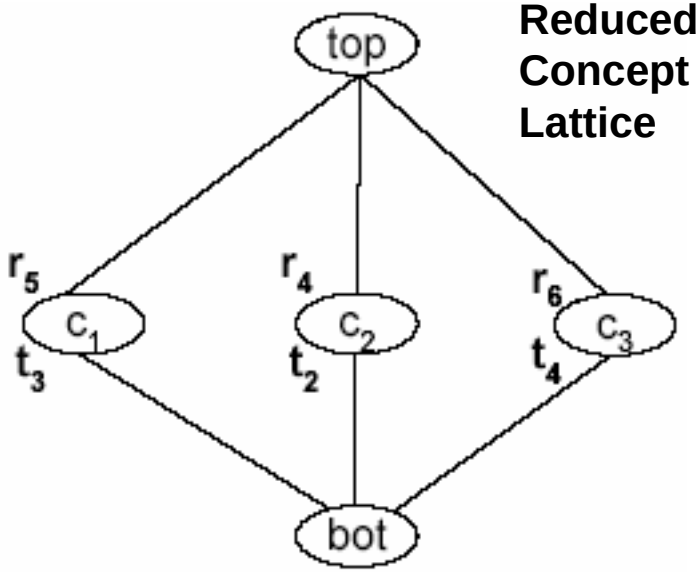
**Reduced Context Table**

| | $r_2$ | $r_4$ | $r_5$ | $r_6$ |
|---|---|---|---|---|
| $t_2$ | | X | | |
| $t_3$ | X | | X | |
| $t_4$ | | | | X |

**Concepts**

| Concept | (Objects,Attributes) |
|---|---|
| TOP | $(\{t_2, t_3, t_4\}, \{\})$ |
| $c_1$ | $(\{t_3\}, \{r_2, r_5\})$ |
| $c_2$ | $(\{t_2\}, \{r_4\})$ |
| $c_3$ | $(\{t_4\}, \{r_6\})$ |
| BOT | $(\{\}, \{r_2, r_4, r_5, r_6\})$ |

**Reduced Concept Lattice**




**Owner reductions select {$t_2$, $t_3$, $t_4$} as the minimized suite which is of optimal size.**

# Selecting Test Cases from Strongest Concepts

[Sampath, Mihaylov, Soutter, & Pollock - 2004]

- each web session as an object, URLs used in session as attributes
- select one test case from each next-to-bottom concept.

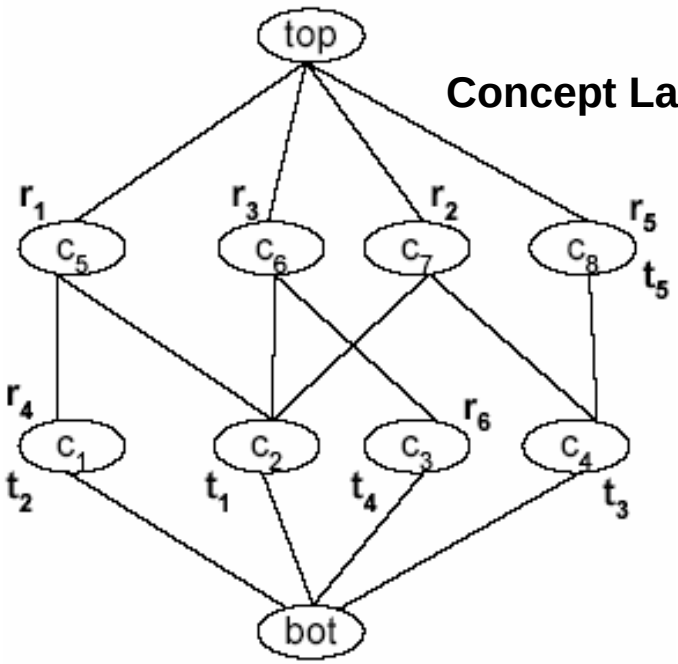
**Context table**

| | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ |
|---|---|---|---|---|---|---|
| $t_1$ | X | X | X | | | |
| $t_2$ | X | | | X | | |
| $t_3$ | | X | | | X | |
| $t_4$ | | | X | | | X |
| $t_5$ | | | | | X | |

**Concepts**

| Concept | (Objects,Attributes) |
|---|---|
| TOP | $(\{t_1, t_2, t_3, t_4, t_5\}, \{\})$ |
| $c_1$ | $(\{t_2\}, \{r_1, r_4\})$ |
| $c_2$ | $(\{t_1\}, \{r_1, r_2, r_3\})$ |
| $c_3$ | $(\{t_4\}, \{r_3, r_6\})$ |
| $c_4$ | $(\{t_3\}, \{r_2, r_5\})$ |
| $c_5$ | $(\{t_1, t_2\}, \{r_1\})$ |
| $c_6$ | $(\{t_1, t_4\}, \{r_3\})$ |
| $c_7$ | $(\{t_1, t_3\}, \{r_2\})$ |
| $c_8$ | $(\{t_3, t_5\}, \{r_5\})$ |
| BOT | $(\{\}, \{r_1, r_2, r_3, r_4, r_5, r_6\})$ |

**Concept Lattice**




**Minimized suite $\{t_1, t_2, t_3, t_4\}$**

**$t_1$ is redundant**

# Our Delayed-Greedy Algorithm

Input: Context table for given test suite T

Output: Test cases in minimized suite $T_{min}$

While (Context Table != empty) do

  While (heuristic not needed) and (Context table != empty) do

    Remove rows $o_j$ for *object implications* $o_i \Rightarrow o_j$

    Remove columns $r_j$ for *attribute implications* $r_i \Rightarrow r_j$

    Add test cases corresponding to *owner reductions* to $T_{min}$ and update Context Table.

  Endwhile

  If (Context Table != empty) Then Pick test case using greedy heuristic, add it to $T_{min}$ and update the Context Table Endif

Endwhile

If (greedy heuristic never used) Then $T_{min}$ is of optimal size Endif

Return($T_{min}$)

# Experiments

| Prog. | loc. | *Avg. size of un-minimized suite* | | *Total No. of requirements* | |
|---|---|---|---|---|---|
| | | Branch Cov. | Def-use Cov. | Branches | Def-use pairs |
| space | 6218 | 533 | 539 | 1356 | 5179 |
| tcas | 138 | 20 | 21 | 41 | 51 |
| print tokens | 402 | 64 | 66 | 127 | 275 |
| print tokens2 | 483 | 77 | 79 | 154 | 235 |
| schedule | 299 | 46 | 46 | 84 | 148 |
| replace | 516 | 83 | 108 | 155 | 759 |
| totinfo | 346 | 53 | 53 | 83 | 287 |

# DelGreedy vs. Greedy, HGS, and SMSP

**Average Size Of Minimized Suites**

| Program | Branch Coverage |
|---|---|
| space | 123 |
| tcas | 4 |
| print-tokens | 6 |
| print-tokens2 | 4 |
| schedule | 2 |
| replace | 9 |
| totinfo | 2 |

DelGreedy computed same or smaller size suites for all programs

**Number of times**

**( $|T_{min}|$ by Algo. - $|T_{min}|$ by DelGreedy) = 0,1, 2, 3…**

| Prog. | Algo. | *frequency of (size of* **Tmin by Algo.** *- size of* **Tmin by DelGreedy** *)* Branch Coverage Suites | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | > 9 |
| space | Greedy | 0 | 4 | 10 | 20 | 22 | 22 | 12 | 8 | 2 | - | - |
| | HGS | 4 | 19 | 22 | 18 | 18 | 10 | 5 | 3 | 1 | - | - |
| | SMSP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 |
| tcas | Greedy | 57 | 41 | 2 | - | - | - | - | - | - | - | - |
| | HGS | 58 | 36 | 6 | - | - | - | - | - | - | - | - |
| | SMSP | 0 | 0 | 0 | 0 | 5 | 6 | 11 | 10 | 13 | 18 | 37 |
| print tokens | Greedy | 82 | 17 | 1 | - | - | - | - | - | - | - | - |
| | HGS | 43 | 36 | 15 | 4 | 2 | - | - | - | - | - | - |
| | SMSP | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 | 2 | 2 | 91 |
| print tokens2 | Greedy | 86 | 14 | - | - | - | - | - | - | - | - | - |
| | HGS | 49 | 44 | 17 | 0 | - | - | - | - | - | - | - |
| | SMSP | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 94 |
| schedule | Greedy | 100 | 0 | - | - | - | - | - | - | - | - | - |
| | HGS | 37 | 48 | 14 | 1 | - | - | - | - | - | - | - |
| | SMSP | 99 | 0 | 1 | - | - | - | - | - | - | - | - |
| replace | Greedy | 49 | 45 | 4 | 2 | - | - | - | - | - | - | - |
| | HGS | 33 | 37 | 27 | 3 | - | - | - | - | - | - | - |
| | SMSP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 99 |
| totinfo | Greedy | 84 | 16 | - | - | - | - | - | - | - | - | - |
| | HGS | 27 | 47 | 20 | 5 | 1 | - | - | - | - | - | - |
| | SMSP | 1 | 2 | 8 | 12 | 20 | 12 | 17 | 12 | 9 | 4 | 3 |

# Number of Optimal Size Suites

| Prog. | Algo. | *Branch Coverage Suites* | | | |
|---|---|---|---|---|---|
| | | NO | OPT | UN | Time (msec) |
| space | DelGreedy | - | 92 | 8 | 259.876 |
| | Greedy | 100 | 0 | 0 | 266.012 |
| | HGS | 96 | 4 | 0 | 280.213 |
| | SMSP | 100 | 0 | 0 | - |
| tcas | DelGreedy | - | 68 | 32 | 4.294 |
| | Greedy | 43 | 37 | 20 | 3.876 |
| | HGS | 42 | 39 | 19 | 4.031 |
| | SMSP | 100 | 0 | 0 | - |
| print tokens | DelGreedy | - | 71 | 29 | 6.295 |
| | Greedy | 18 | 62 | 20 | 6.163 |
| | HGS | 57 | 35 | 8 | 5.967 |
| | SMSP | 100 | 0 | 0 | - |
| print tokens2 | DelGreedy | - | 84 | 16 | 7.051 |
| | Greedy | 14 | 70 | 16 | 6.820 |
| | HGS | 51 | 29 | 20 | 7.128 |
| | SMSP | 100 | 0 | 0 | - |
| schedule | DelGreedy | - | 99 | 1 | 5.446 |
| | Greedy | 0 | 99 | 1 | 5.212 |
| | HGS | 63 | 36 | 1 | 5.863 |
| | SMSP | 1 | 99 | 0 | - |
| replace | DelGreedy | - | 53 | 47 | 8.510 |
| | Greedy | 51 | 25 | 24 | 7.512 |
| | HGS | 67 | 17 | 16 | 7.763 |
| | SMSP | 100 | 0 | 0 | - |
| totinfo | DelGreedy | - | 46 | 54 | 5.483 |
| | Greedy | 16 | 32 | 52 | 5.318 |
| | HGS | 73 | 11 | 16 | 5.565 |
| | SMSP | 99 | 1 | 0 | - |

DelGreedy computed same or more number of optimal size solutions than other algorithms

Time performance of DelGreedy was comparable to other algorithms.

The University of Arizona. Tucson Arizona

# DelGreedy vs. Variants of DelGreedy

**Number of times**

**( $|T_{min}|$ by Algo. - $|T_{min}|$ by DelGreedy) = 0,1, 2, 3…**

| Prog. | Algo. | *frequency of ( size of* Tmin by Algo. *- size of* Tmin by DelGreedy *)* Branch Coverage Suites | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | >10 |
| space | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 79 | 19 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 99 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 71 | 27 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 4 | 9 | 25 | 34 | 17 | 4 | 6 | 1 | 0 | 0 | 0 | 0 |
| | Attr. | 79 | 19 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| tcas | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 97 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 92 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 86 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 67 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Attr. | 97 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| print tokens | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 95 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 99 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 92 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 88 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Attr. | 95 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| print tokens2 | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 96 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# DelGreedy vs. Variants of DelGreedy

**Number of times**

**( $|T_{min}|$ by Algo. - $|T_{min}|$ by DelGreedy) = 0,1, 2, 3…**

| Prog. | Algo. | *frequency of ( size of* **Tmin by Algo.** *- size of* **Tmin by DelGreedy** *)* Branch Coverage Suites | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | >10 |
| schedule | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| replace | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 95 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 94 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 92 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 61 | 38 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Attr. | 95 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| totinfo | Obj. + Attr. | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Attr. | 97 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. + Obj. | 91 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Own. | 88 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Obj. | 88 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Attr. | 97 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# DelGreedy vs. Variants of DelGreedy

Observation: Obj.+Attr. always gives same size solutions as DelGreedy because owner reductions appear as attribute reductions.

Question: Why use owner reductions at all?

Answer: Owner reductions reduce the size of the table sooner resulting in better overall time performance of DelGreedy in comparison to Obj.+Attr.

For example, in *space* DelGreedy took 662 milliseconds while Obj.+Attr. took 5516 milliseconds.

# Conclusions

- A new greedy heuristic called Delayed-Greedy that is designed to obtain same or more reduction in the test suite size as compared to Classical Greedy heuristic.
- In our experiments, Delayed-Greedy also always produced same or smaller size minimized suites than HGS and SMSP.
- In our experiments, time performance of Delayed-Greedy was comparable to other algorithms.
- Delayed-Greedy computed larger number of optimal size solutions as compared to other algorithms. Moreover, it was able to identify that the solution was optimal in a large number of cases.