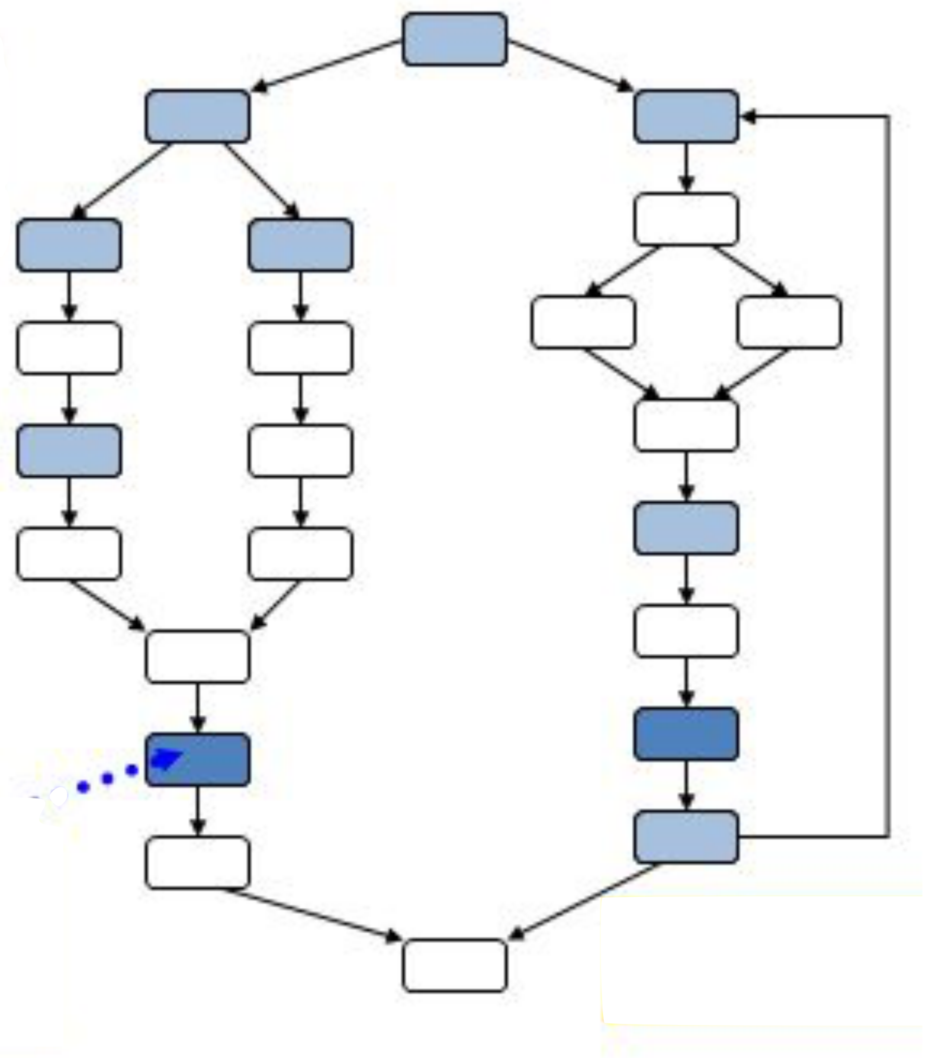
Hyperheuristic Observation Based Slicing of Guava

Seongmin Lee and Shin Yoo

Korea Advanced Institute of Science and Technology
COINSE Lab

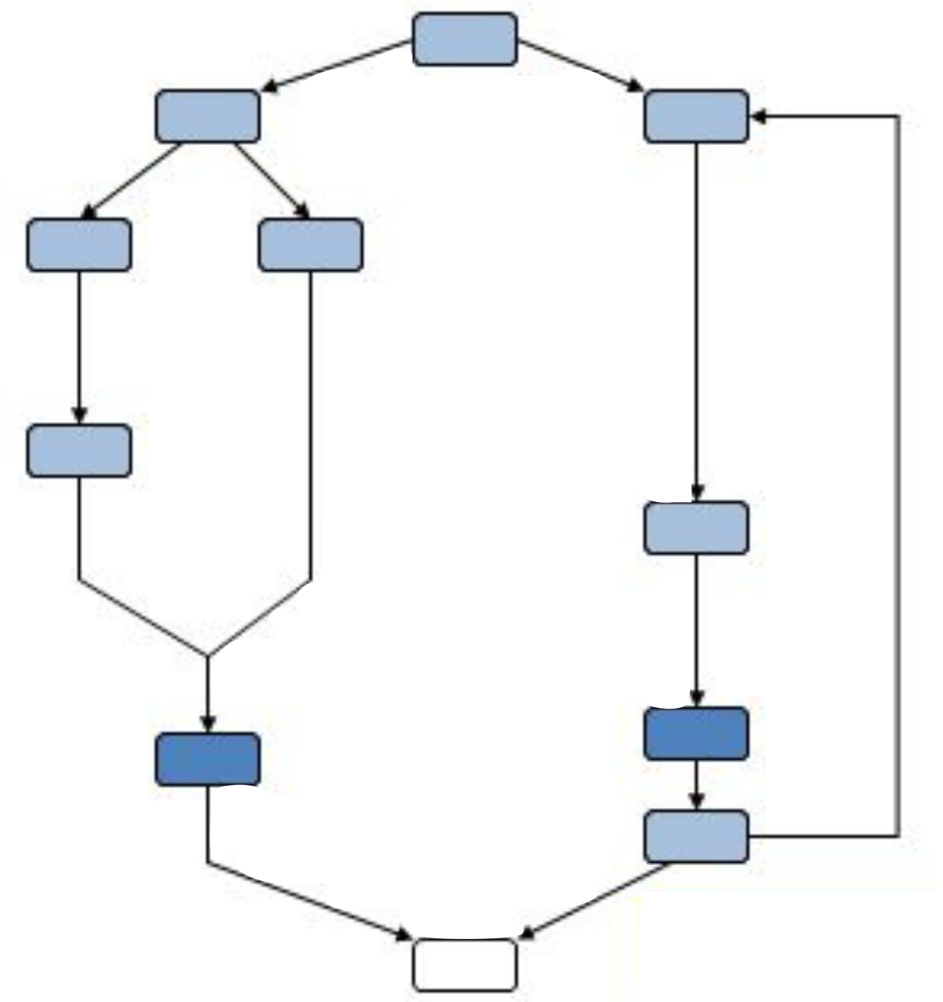
Program Slicing

- Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.
- Specific behavior: Slicing Criterion $< i , V >$ (i : line number, V : variable name)
- Testing, Debugging, Maintenance, etc.



Program Slicing

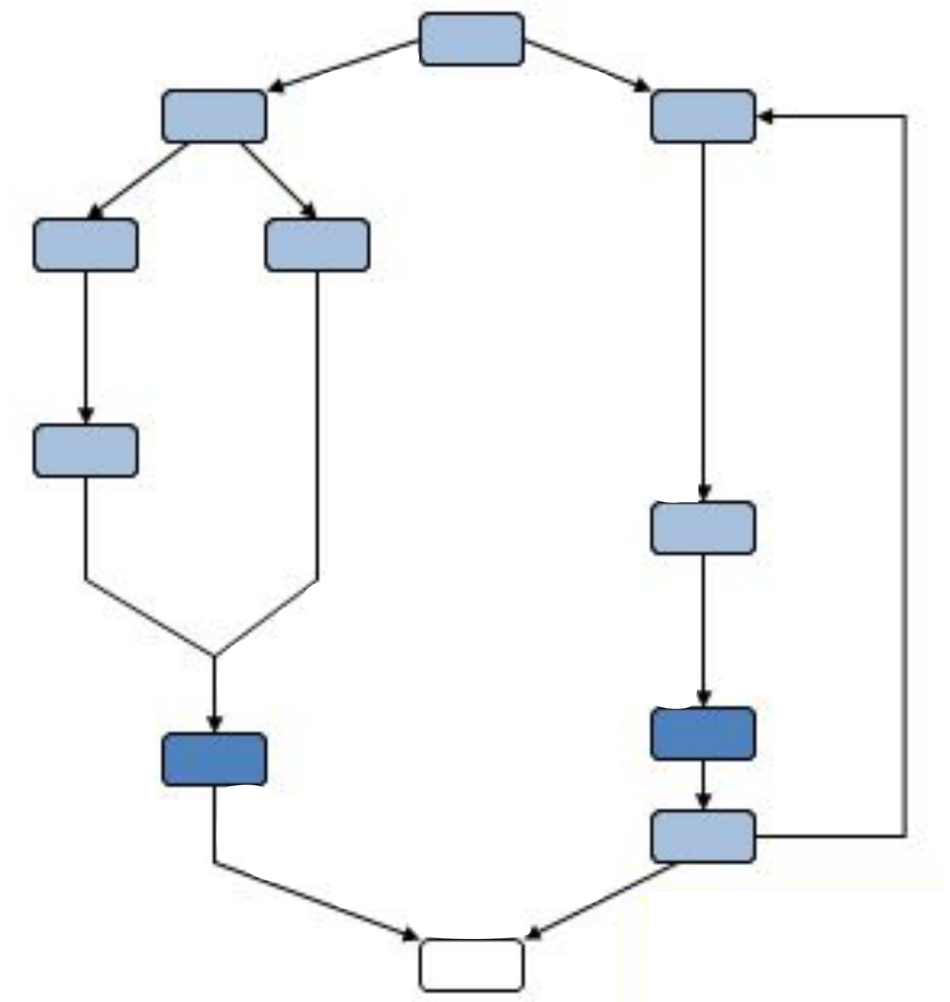
- Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.
- Specific behavior: Slicing Criterion $< i , V >$ (i : line number, V : variable name)
- Testing, Debugging, Maintenance, etc.



Program Slicing

- Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.
- Specific behavior: Slicing Criterion $< i , V >$ (i : line number, V : variable name)
- Testing, Debugging, Maintenance, etc.

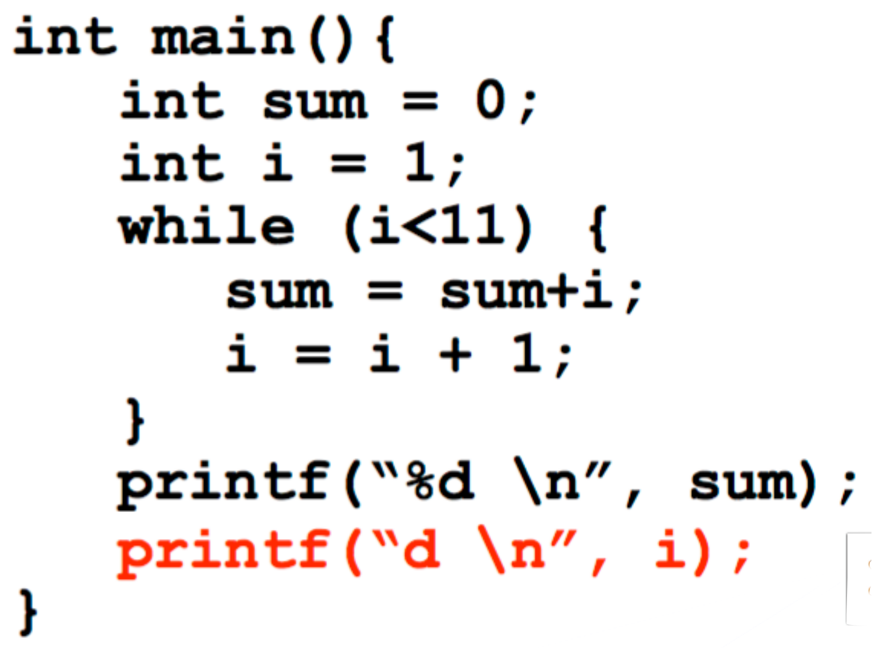
- Limitations:
  - scalability of static analysis
  - lack of supports on multi-lingual systems.



Observation Based Slicing (ORBS)

- Purely dynamic & Language Independent
- Makes a series of deletions of code lines, which
    1) leaves the code (still) compilable, and
    2) preserves the trajectory of the slicing criterion.
- Approximate the program dependence via observations of test executions.

Observation-Based Slicing (ORBS)



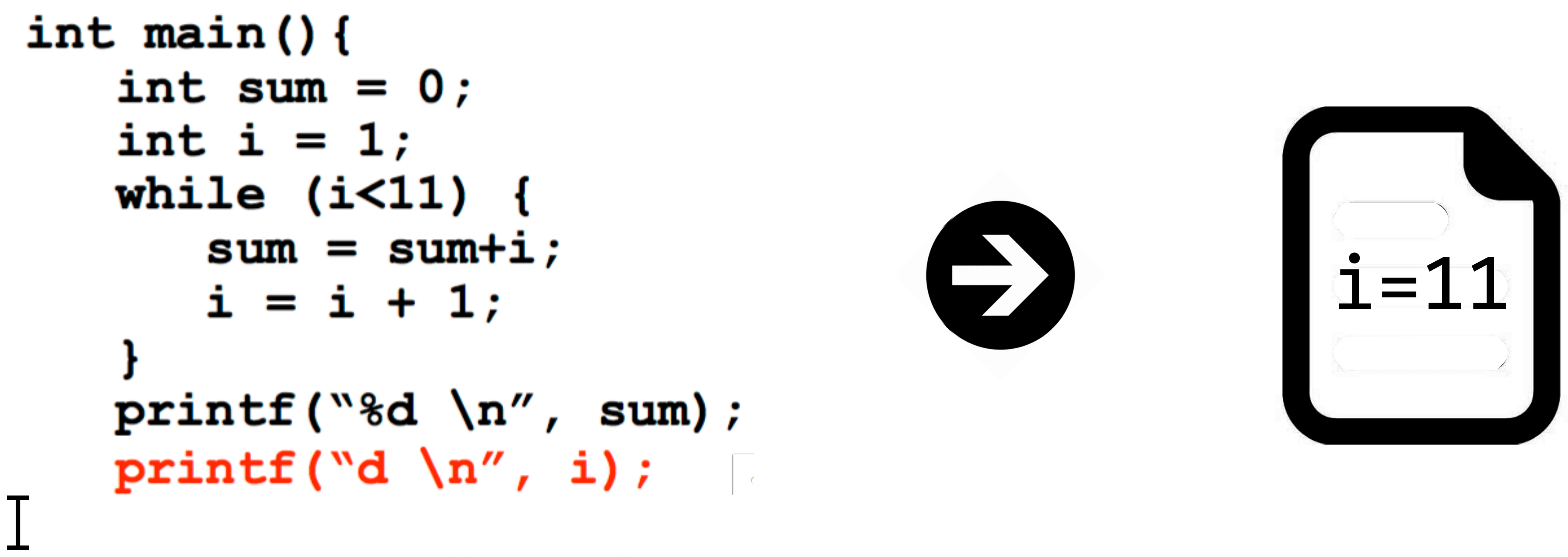
```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```

Observation-Based Slicing (ORBS)

```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```

i=11

Observation-Based Slicing (ORBS)

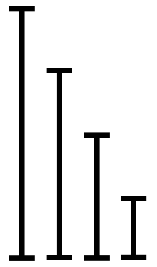


```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
```

```
i=11
```

Observation-Based Slicing (ORBS)

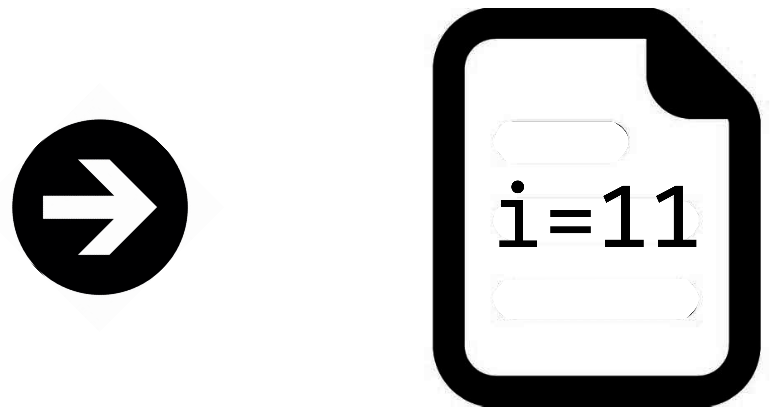
```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
```



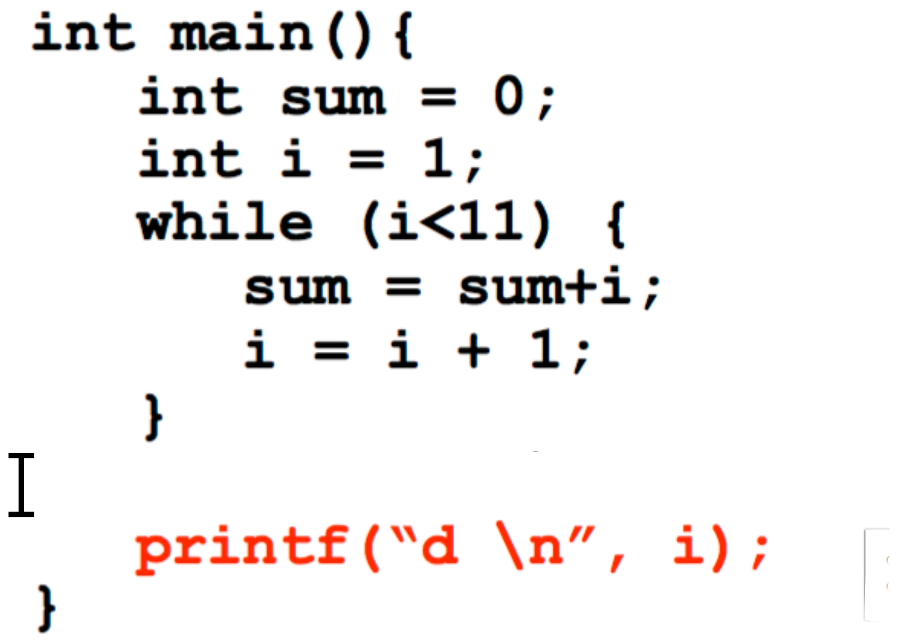
i=11

Observation-Based Slicing (ORBS)

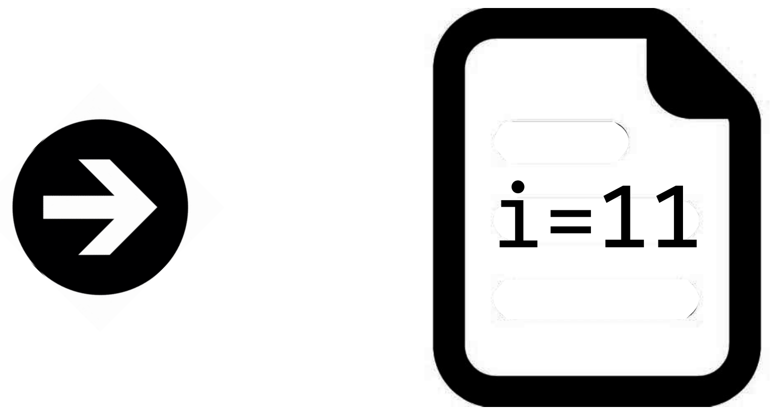
```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
}
```



Observation-Based Slicing (ORBS)



```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }

    printf("d \n", i);
}
```



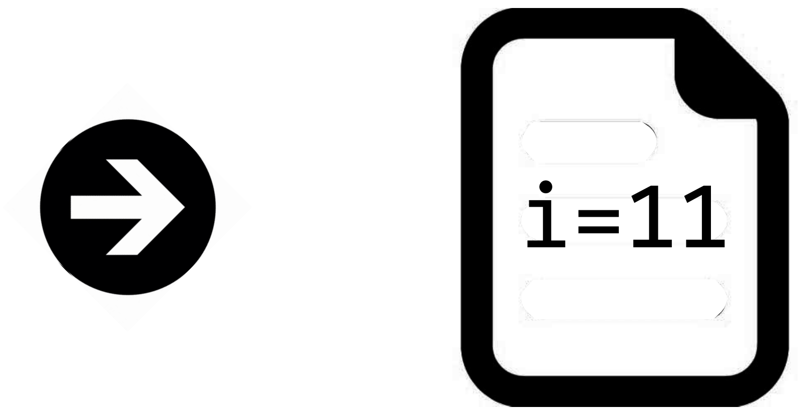
Observation-Based Slicing (ORBS)

```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }

    printf("d \n", i);
}
```

```
i=11
```

Observation-Based Slicing (ORBS)

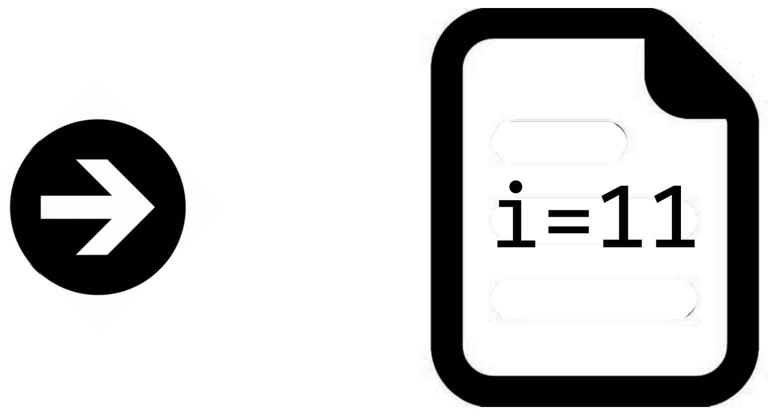
```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }

    printf("d \n", i);
}
```



Window-Deletion

Observation-Based Slicing (ORBS)

```
int main(){

    int i = 1;
    while (i<11) {

        i = i + 1;
    }

    printf("d \n", i);
}
```



Window-Deletion

Observation-Based Slicing (ORBS)

- Purely dynamic & Language Independent
- Able to slice programs on which
    - static slicers are guaranteed to err,

      [3] *ORBS and the Limits of Static Slicing*, SCAM15
    - have highly unconventional semantics.

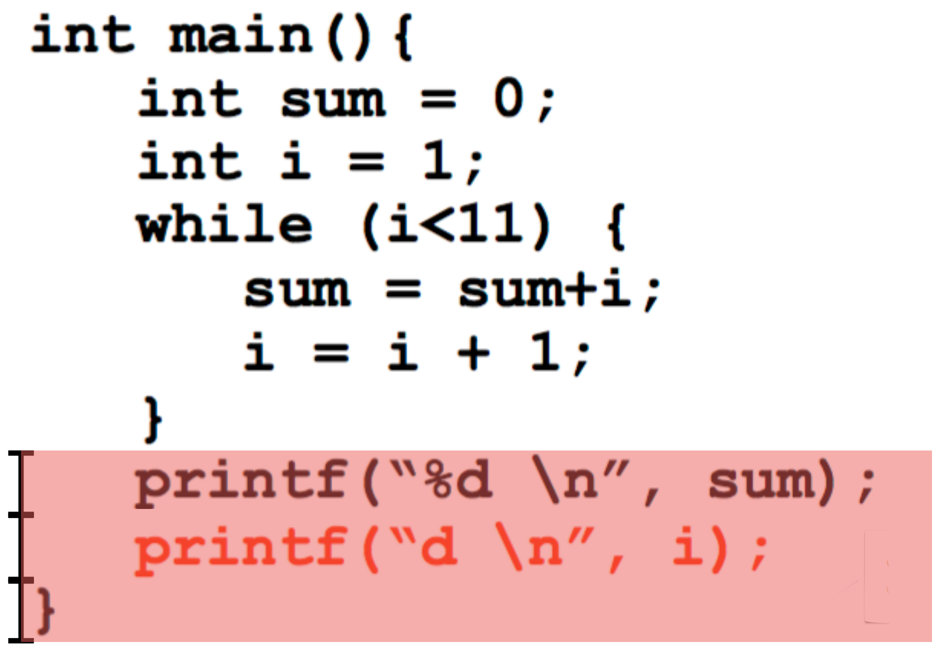
      [9] *Observational slicing based on visual semantics*, JSS17

Limitations of ORBS

- Scalability

Limitations of ORBS

- Scalability



```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```

Limitations of ORBS

- Scalability
  - Takes around 7200 s to delete 220 lines.
    $\Rightarrow$ **0.03 del/s**
    $\Rightarrow$ **32.7 s/del**

(‘`escape`’ package on Guava)

```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```

Scalability

Efficiency

$$\text{Efficiency} = \text{Number of Deleted Lines} / \text{Time spent}$$

Efficiency

=

Number of Deleted Lines

/

Deletion Attempt

$$\uparrow \text{Efficiency} = \frac{\text{Number of Deleted Lines}}{\text{Deletion Attempt}}$$

$$\uparrow \text{Efficiency} = \frac{\uparrow \text{Number of Deleted Lines}}{\text{Deletion Attempt}}$$

Deletion based on Lexical Similarity

Deletion based on Lexical Similarity

“ *Delete all lines of code that are related to a word* ‘log’*!* ”

Deletion based on Lexical Similarity

“ *Delete all lines of code that are related to a word* ‘log’*!* ”

Dependence Approximation

Spatial
Neighborhood

Deletion based on Lexical Similarity

“ Delete all lines of code that are related to a word ‘log’*! ”*

Dependence Approximation

Spatial Neighborhood → Lexical Neighborhood

Deletion based on Lexical Similarity

- Vector Space Model
  - Traditional method for calculating distances between text documents and a query.
- Latent Dirichlet Allocation
  - Probabilistic model that describes which topics are present in a given document.
- Consider each code lines as a document.
- Attempts to delete code lines whose similarity is above certain threshold.

Deletion based on Lexical Similarity

- Vector Space Model ➡ **VSM-Deletion**
  - Traditional method for calculating distances between text documents and a query.
- Latent Dirichlet Allocation ➡ **LDA-Deletion**
  - Probabilistic model that describes which topics are present in a given document.
- Consider each code lines as a document.
- Attempts to delete code lines whose similarity is above certain threshold.

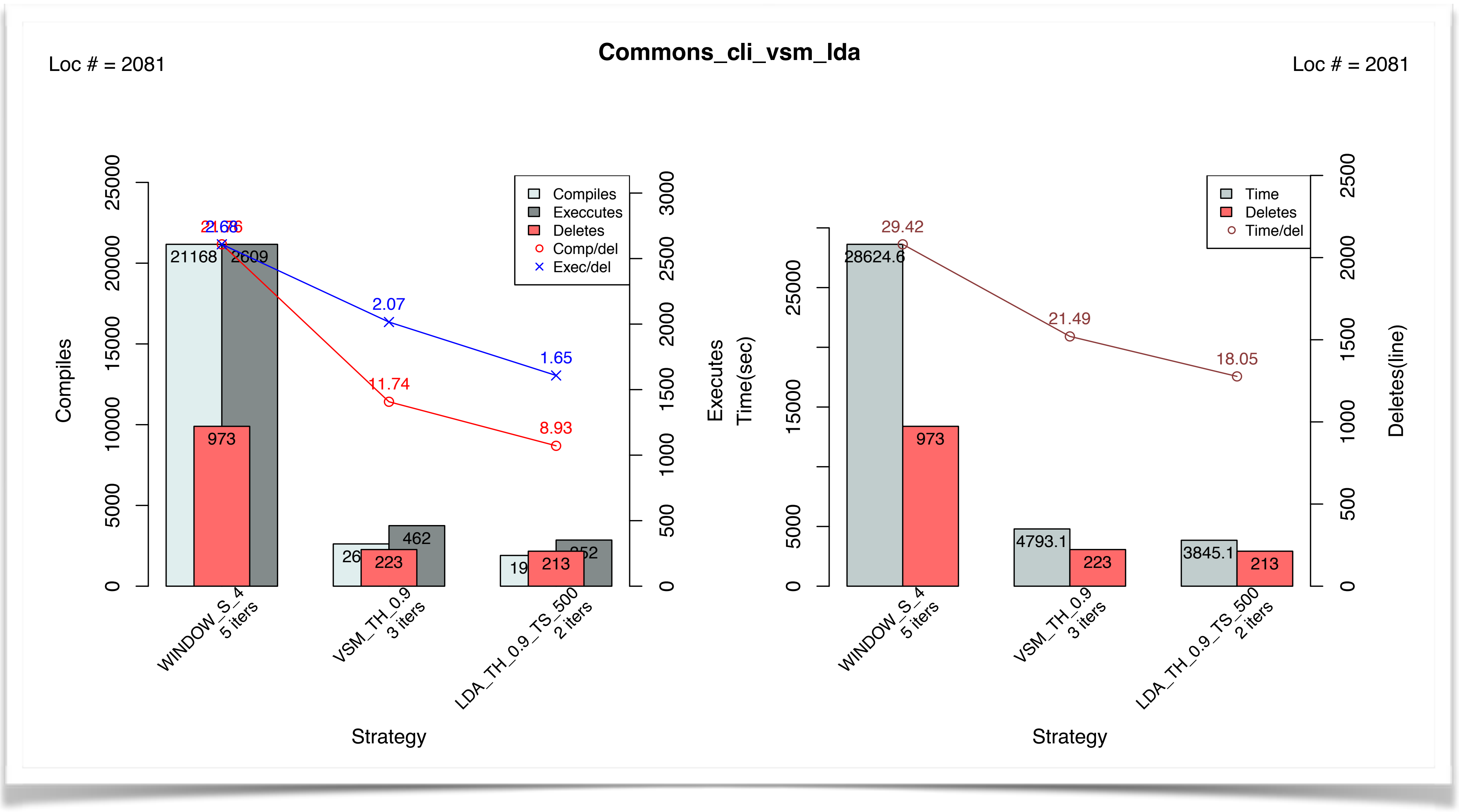
Deletion based on Lexical Similarity

- Vector Space Model ➡ **VSM-Deletion**
  - Traditional method for calculating distances between text documents and a query.
- Latent Dirichlet Allocation ➡ **LDA-Deletion**
  - Probabilistic model that describes which topics are present in a given document.

$\Rightarrow$ Line Similarity based ORBS (LS-ORBS)

[7] *Using source code lexical similarity to improve efficiency of observation based slicing*

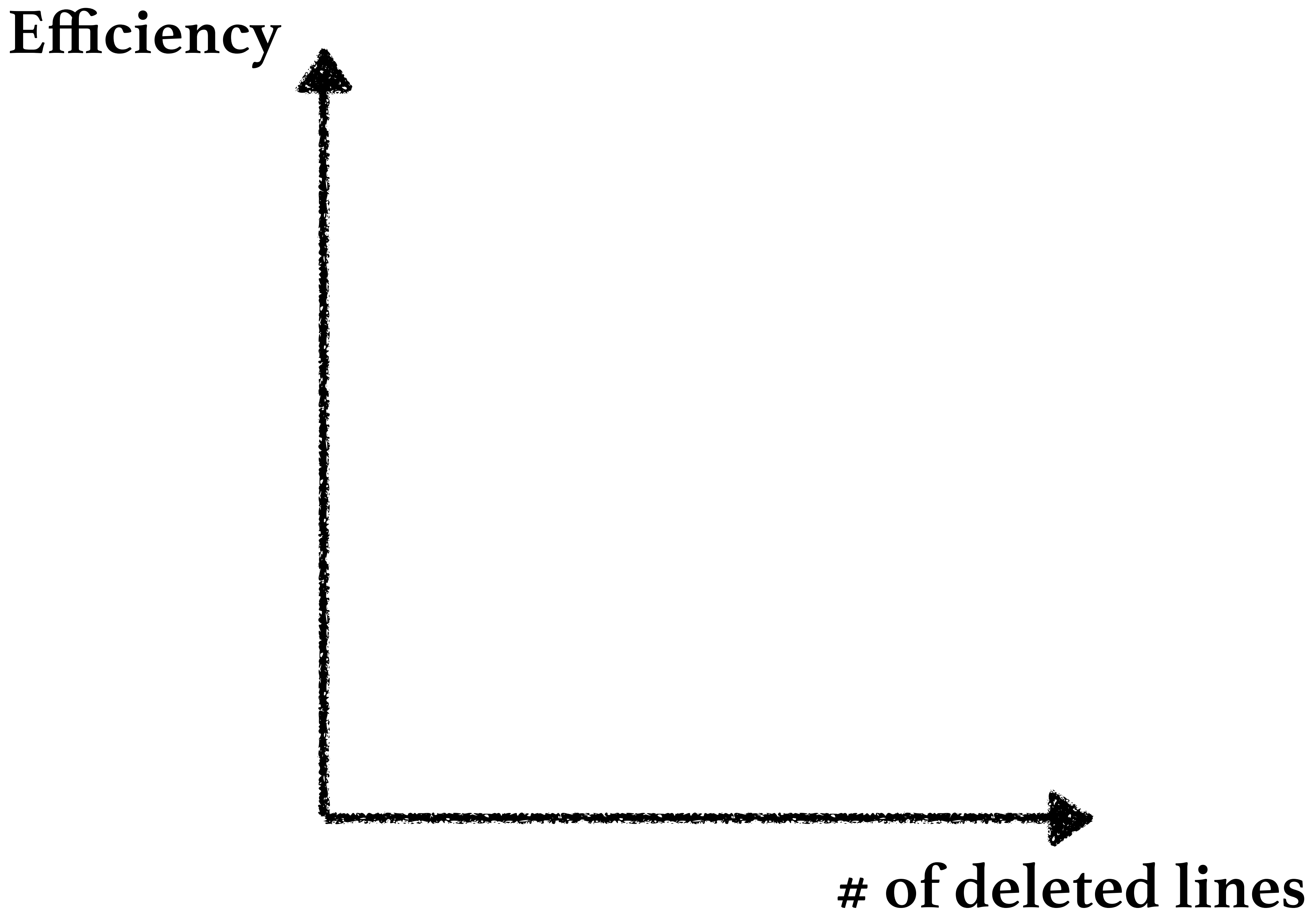
Deletion based on Lexical Similarity



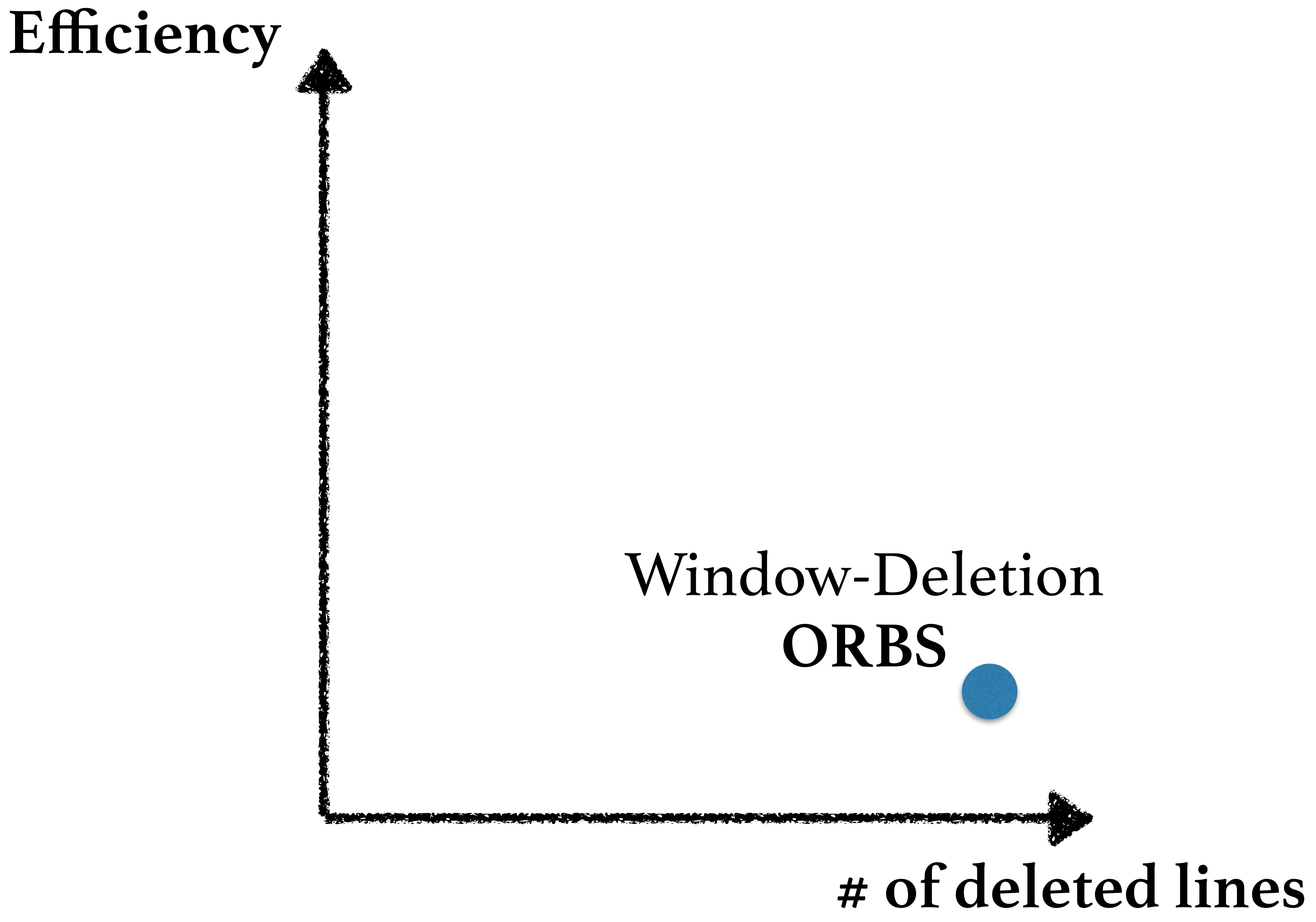
53.3% less compilations, **34.3%** less executions, **39.3%** less time

per 1 deleted lines.

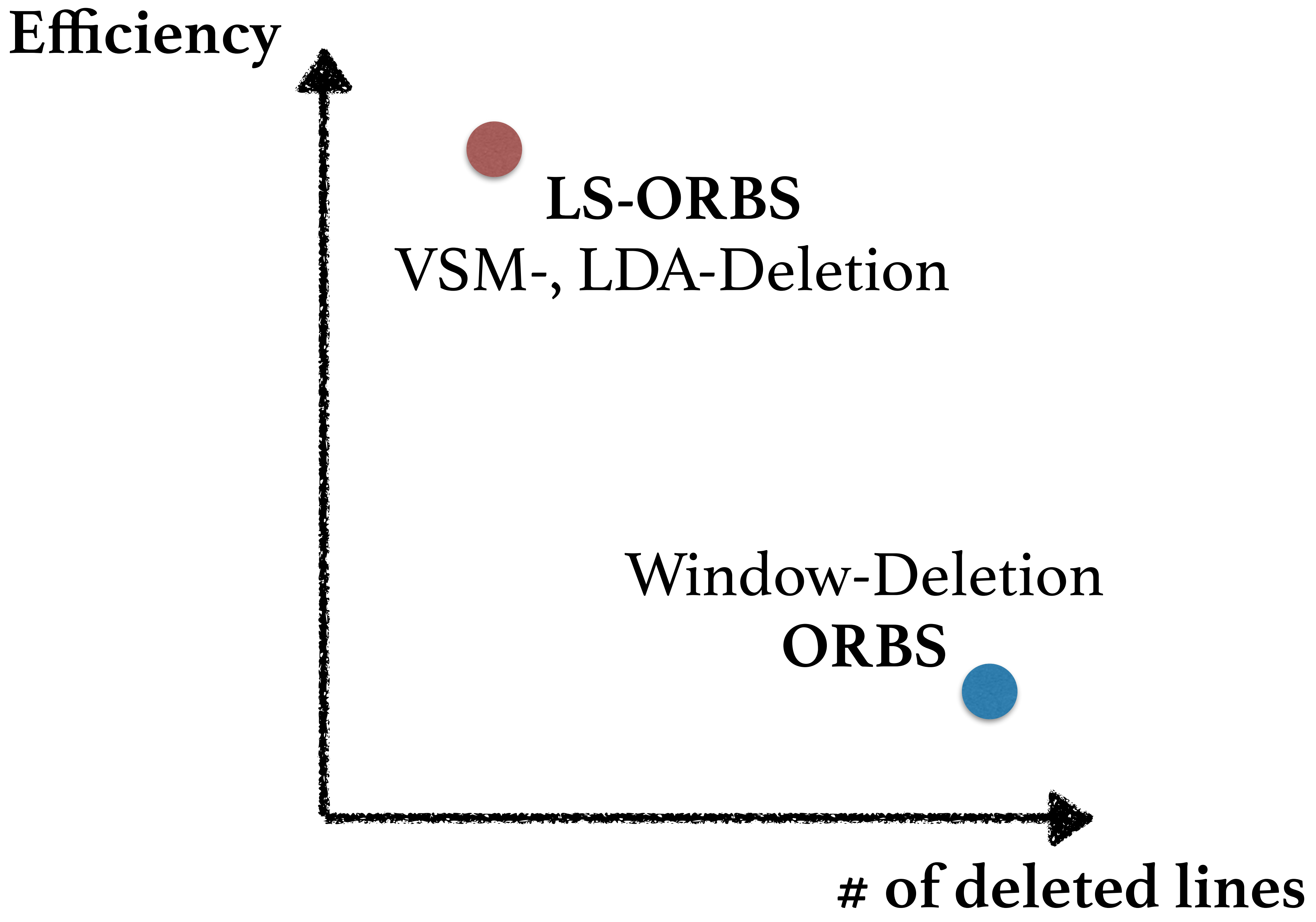
Compare Strategies



Compare Strategies



Compare Strategies



Compare Strategies

● LS-ORBS

● ORBS

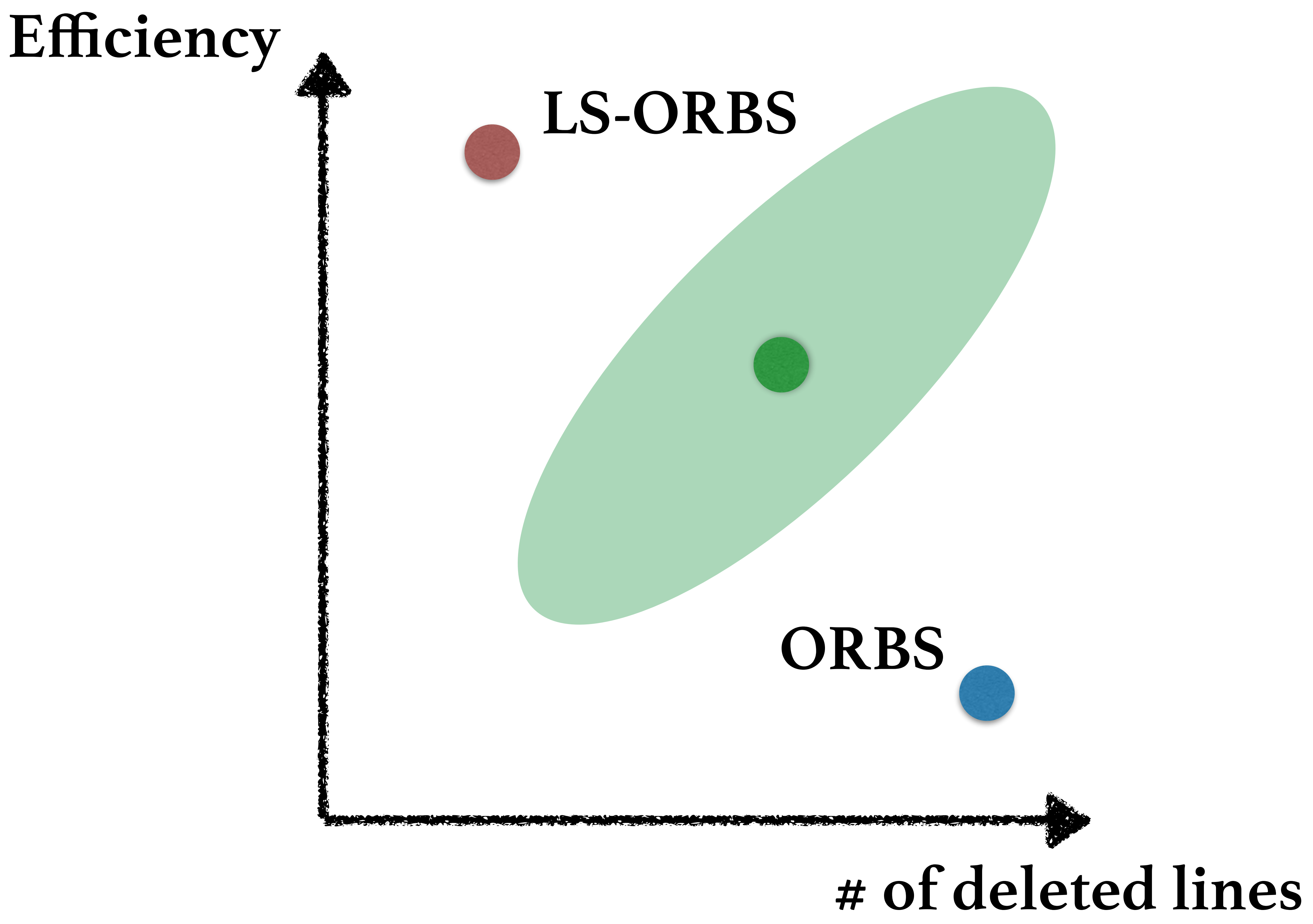
Compare Strategies

VSM-, LDA-Deletion
+
Window-Deletion

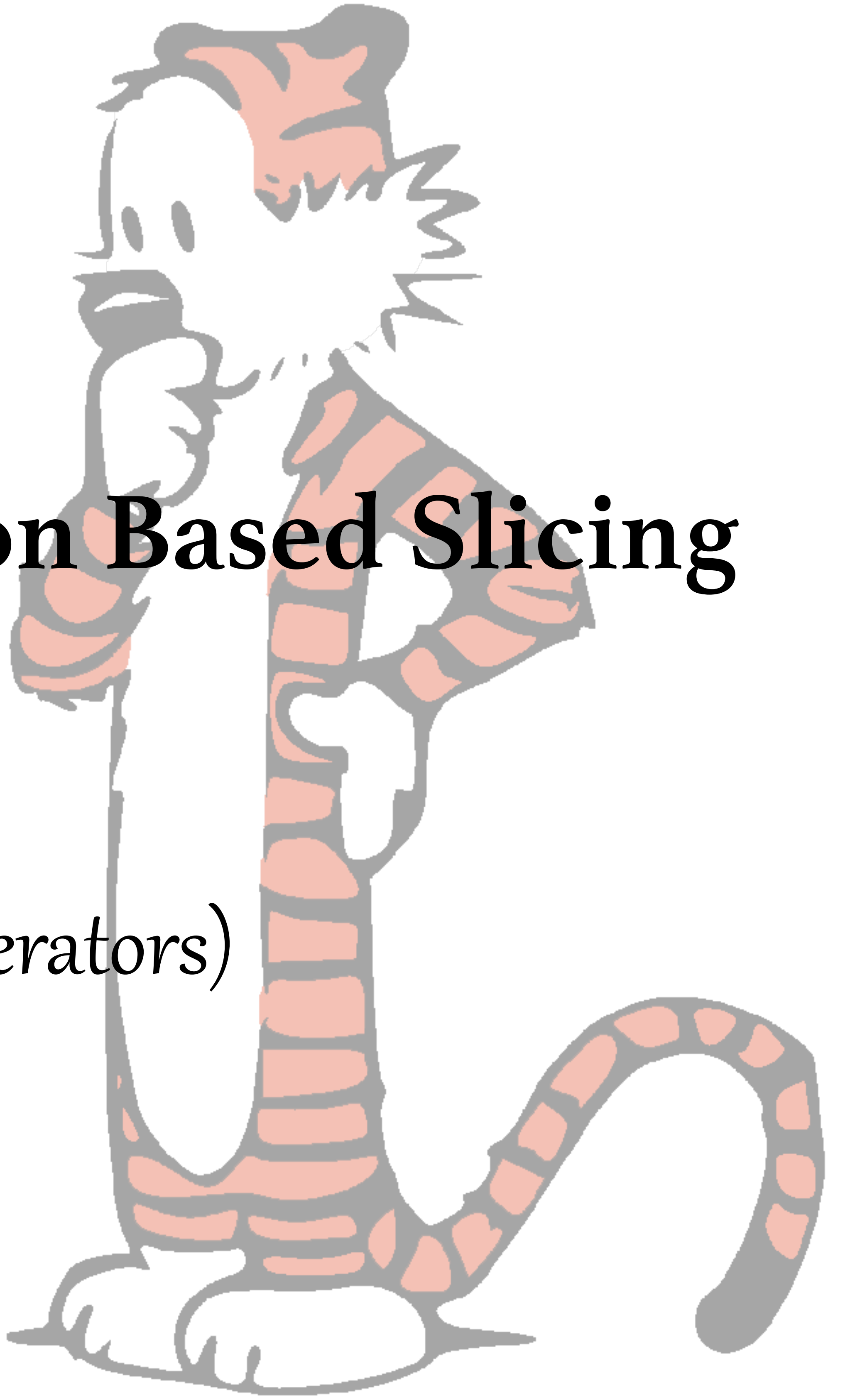
● ● ●

LS-ORBS **ORBS**

Compare Strategies



Q. How to select the operator among various kind of deletion operators ?



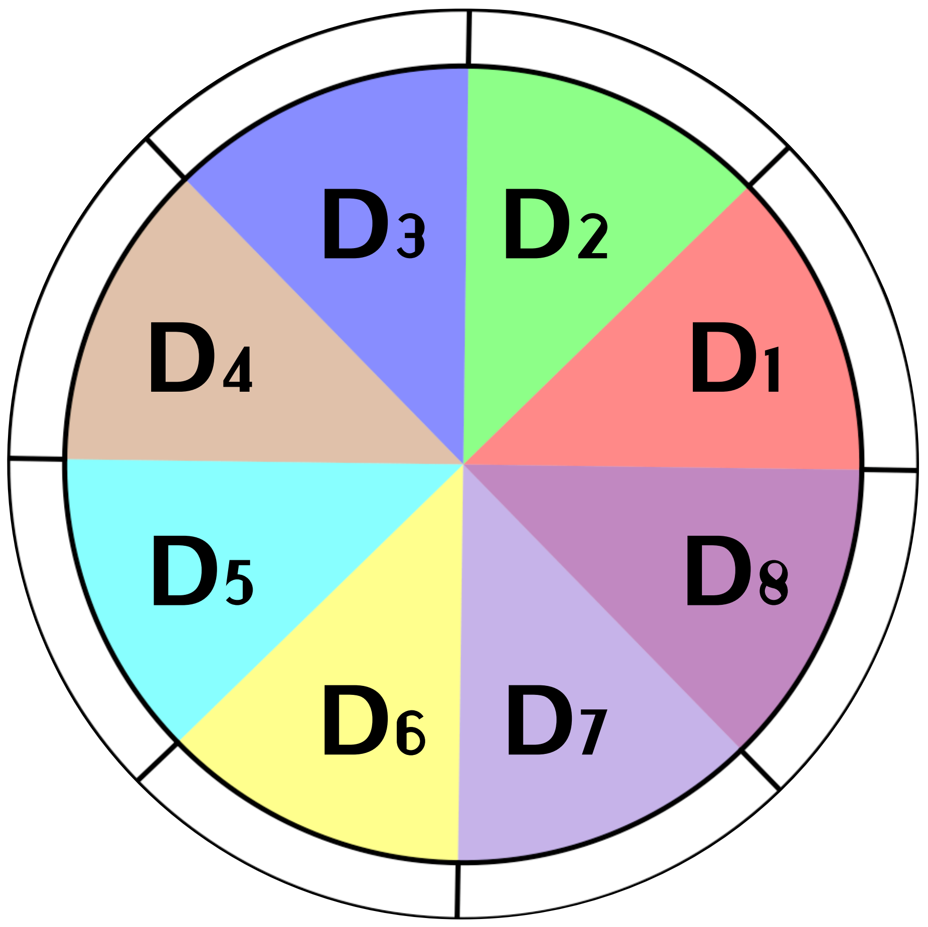
Hyperheuristic Observation Based Slicing

(HOBBES)

(On selecting deletion operators)

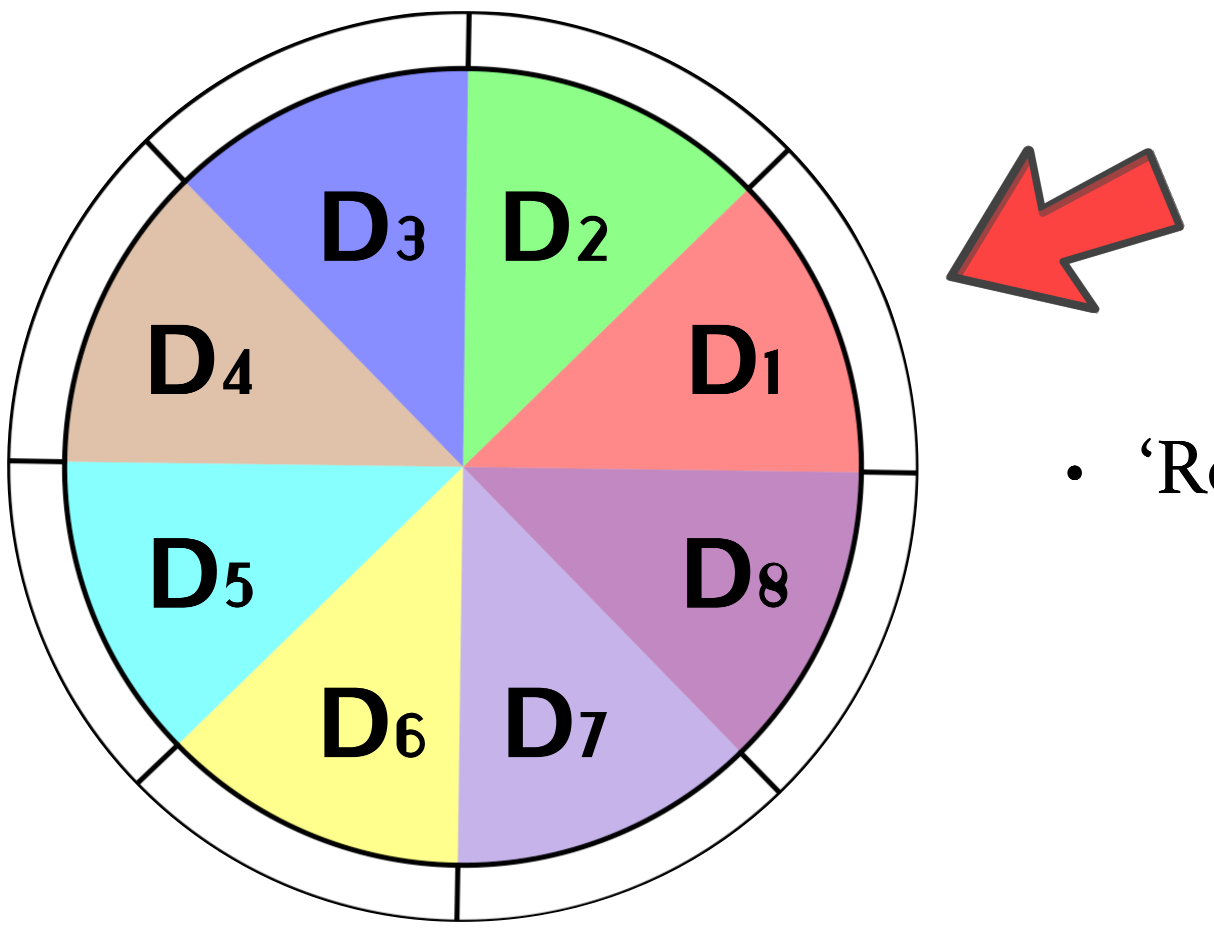
HOBBES Algorithm

HOBBES Algorithm



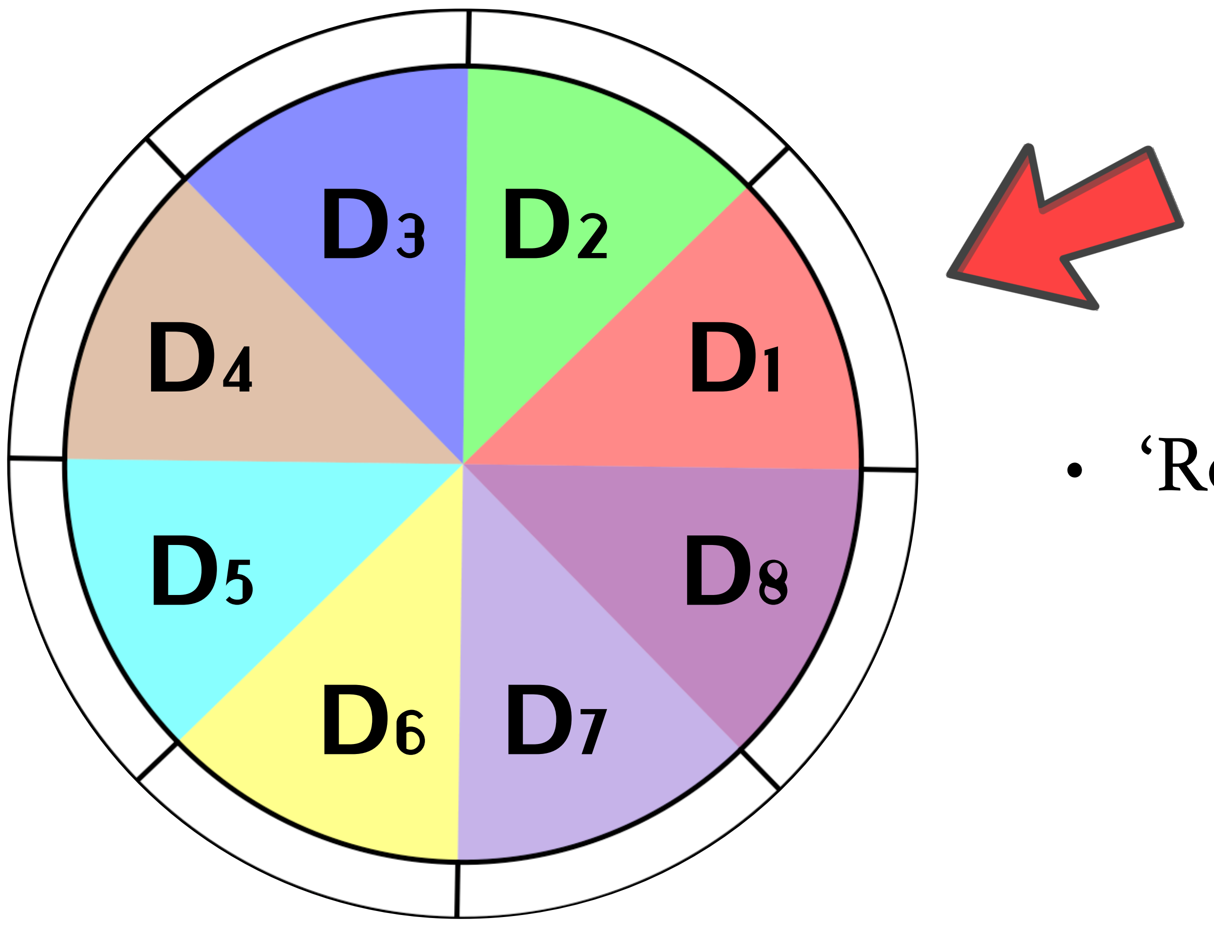
- Initialize selection probability of deletion operators with uniform distribution

HOBBES Algorithm



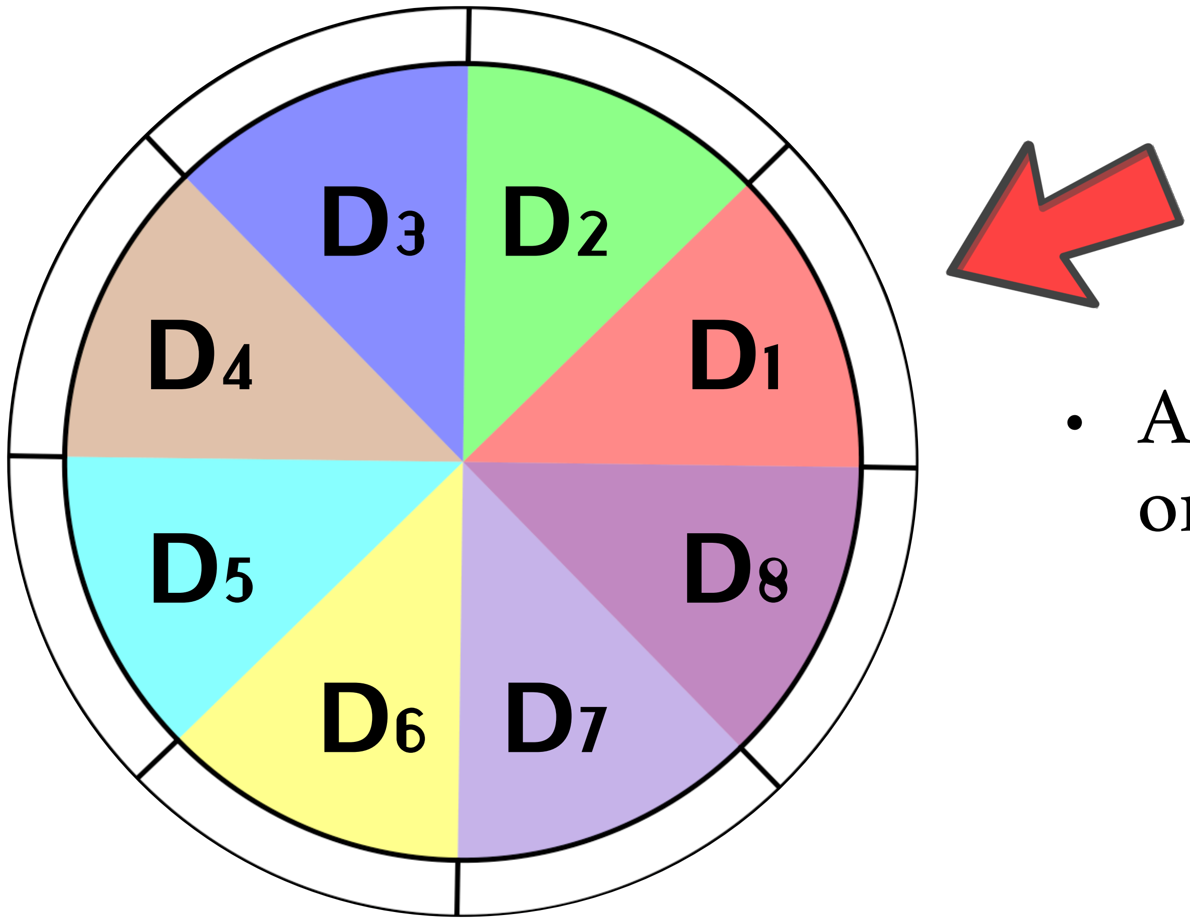
- 'Roulette Wheel Selection'

HOBBES Algorithm



- ‘Roulette Wheel Selection’

HOBBES Algorithm



- Apply selected deletion operator on source code.

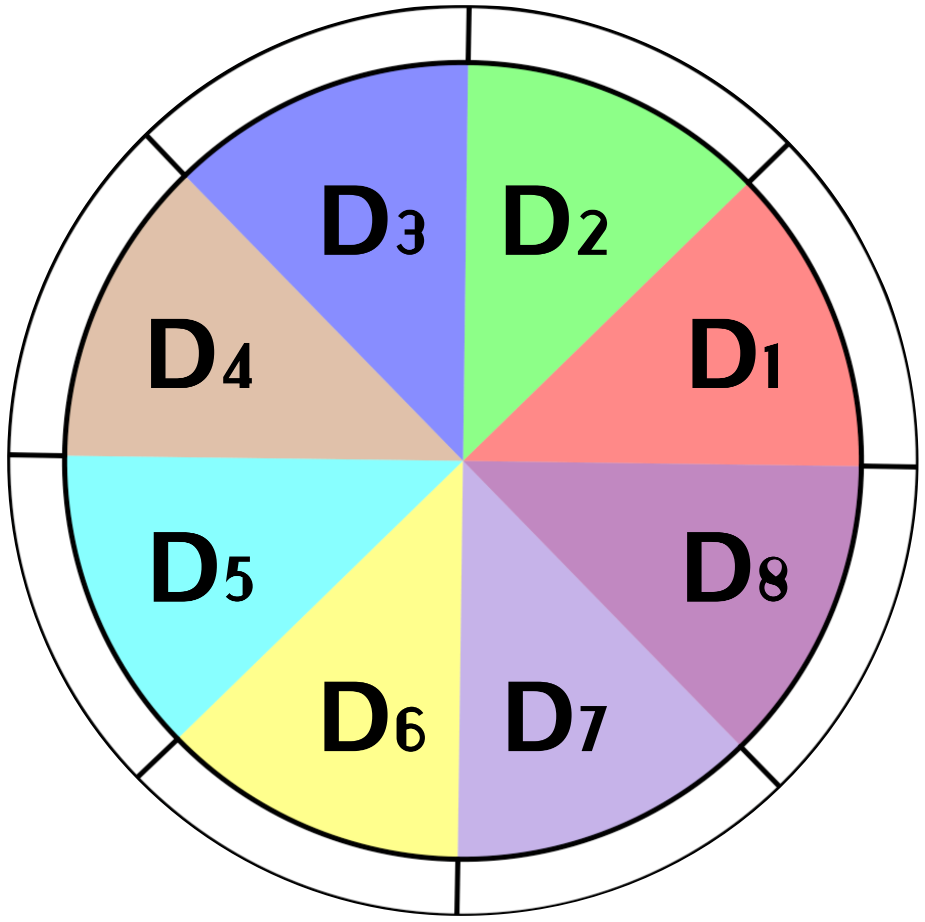
HOBBES Algorithm

$$newP(\text{D}\textsc{k}) = \begin{cases} \omega_{comp} \cdot P(\text{D}\textsc{k}) & \text{when compile fails} \\ \omega_{exec} \cdot P(\text{D}\textsc{k}) & \text{when compile suceeds, trajectory changes} \\ (1 + \log_{10} l) \cdot P(\text{D}\textsc{k}) & \text{otherwise} \end{cases}$$

Probability update formula 'UPDATE'

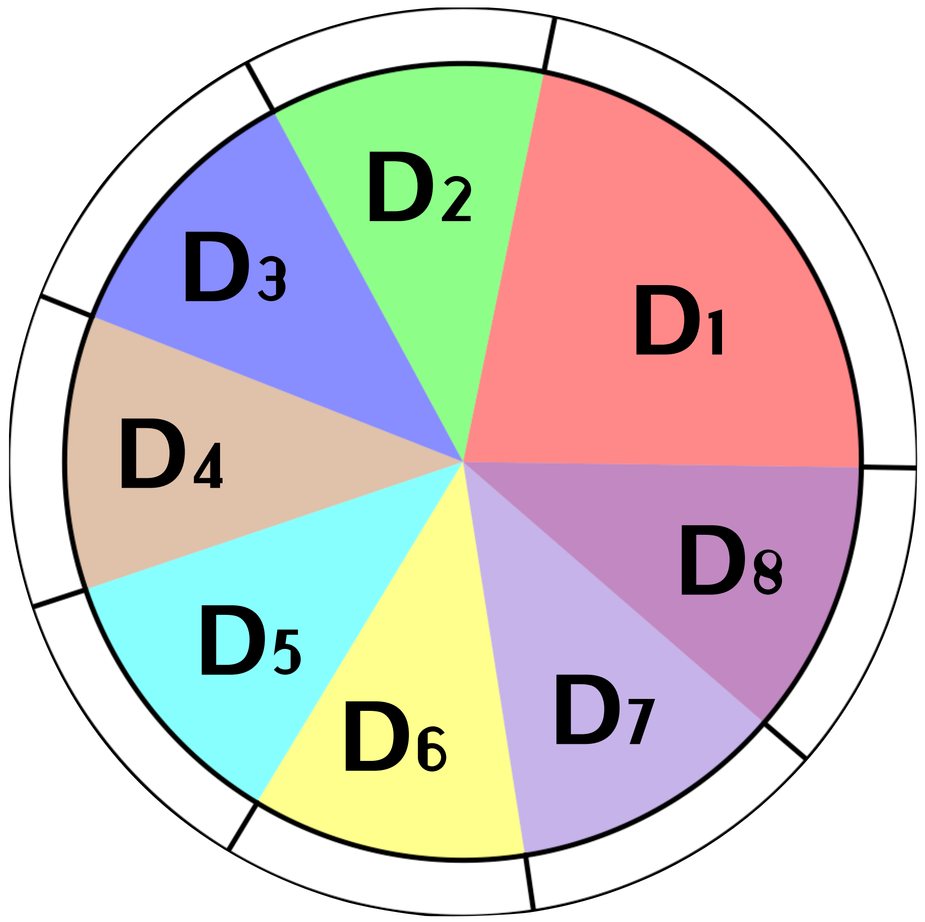
ω: *penalty value* ($\omega \in [0,1]$), l: *# of deleted lines*

HOBBES Algorithm



- Update the probability.

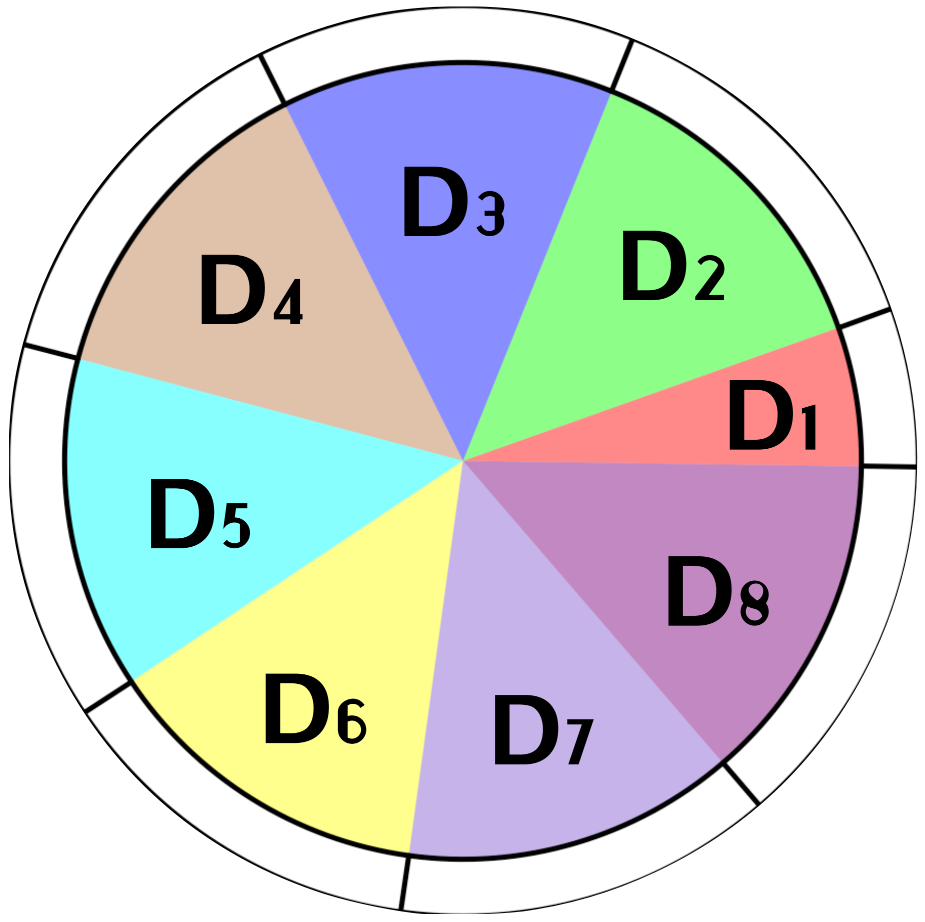
HOBBES Algorithm



Success to delete

- Update the probability.

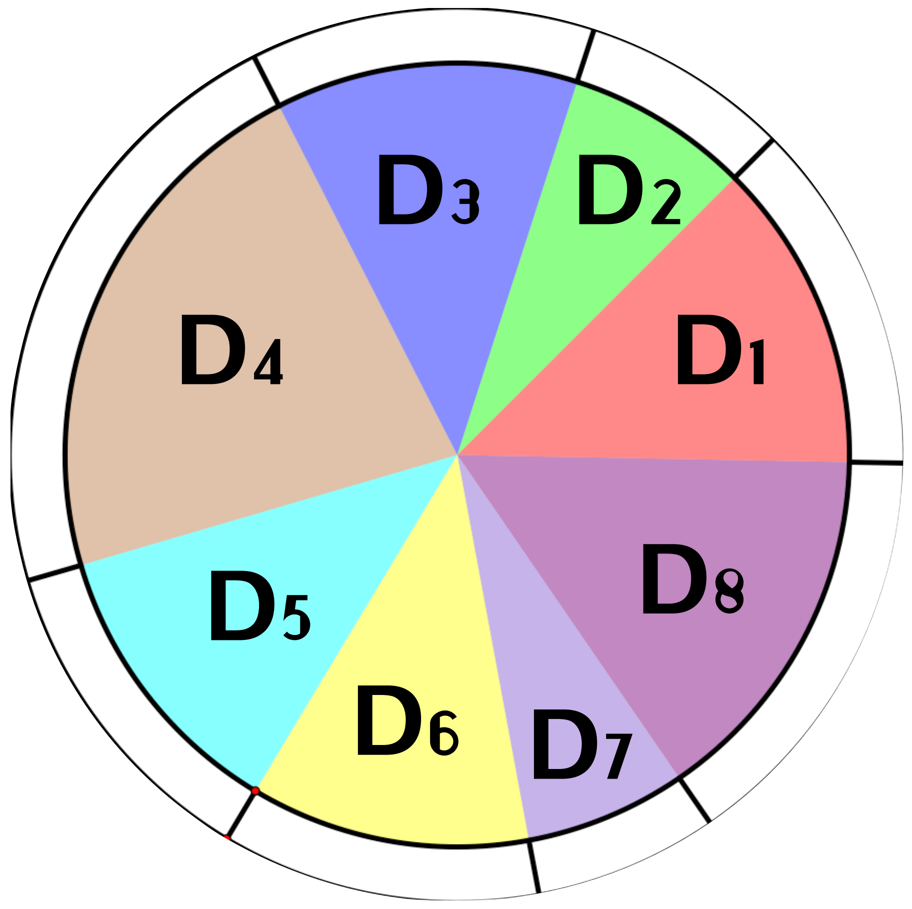
HOBBES Algorithm



Compilation error / Trajectory Change

- Update the probability.

HOBBES Algorithm



- Update the probability.

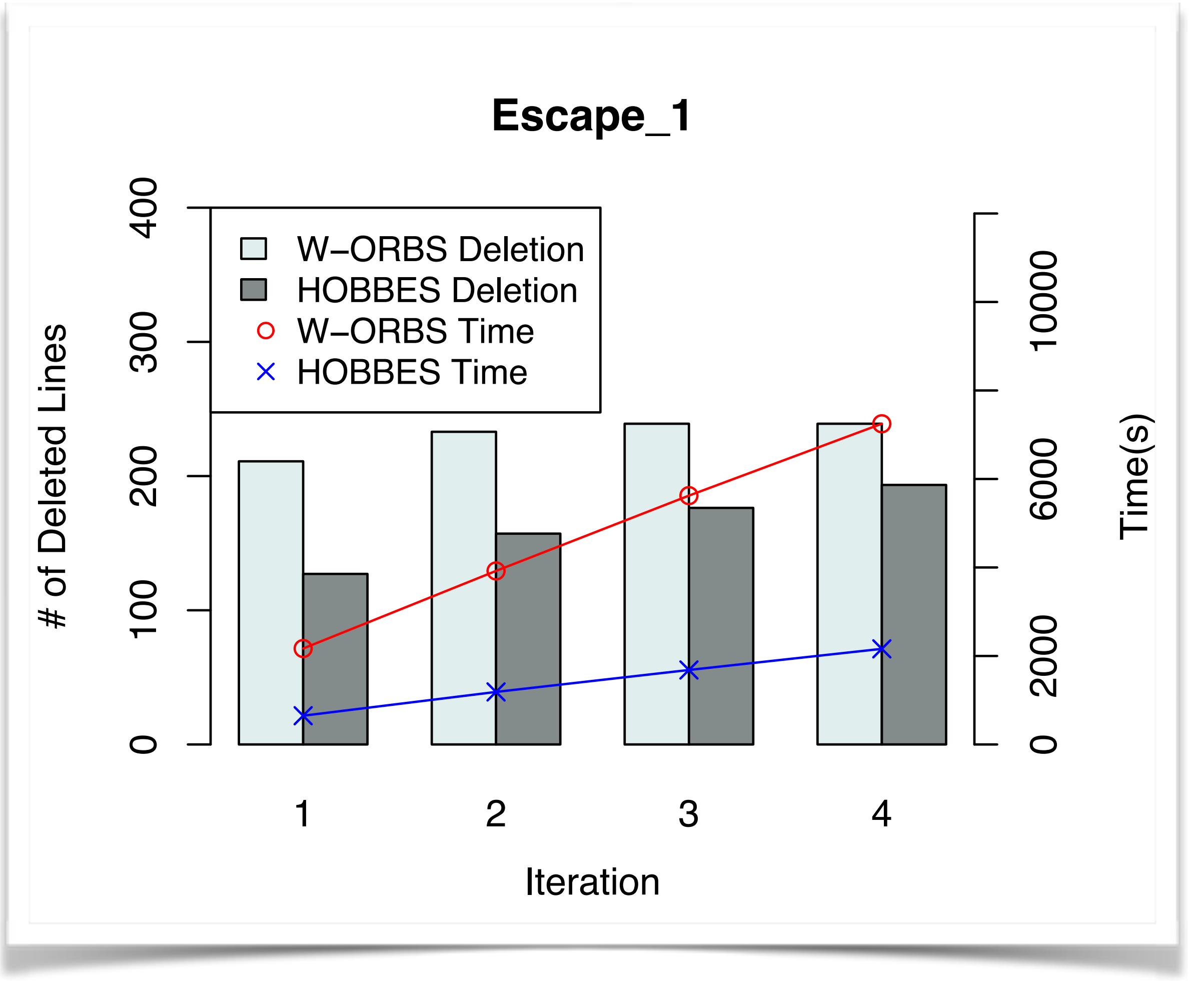
HOBBES - Configuration

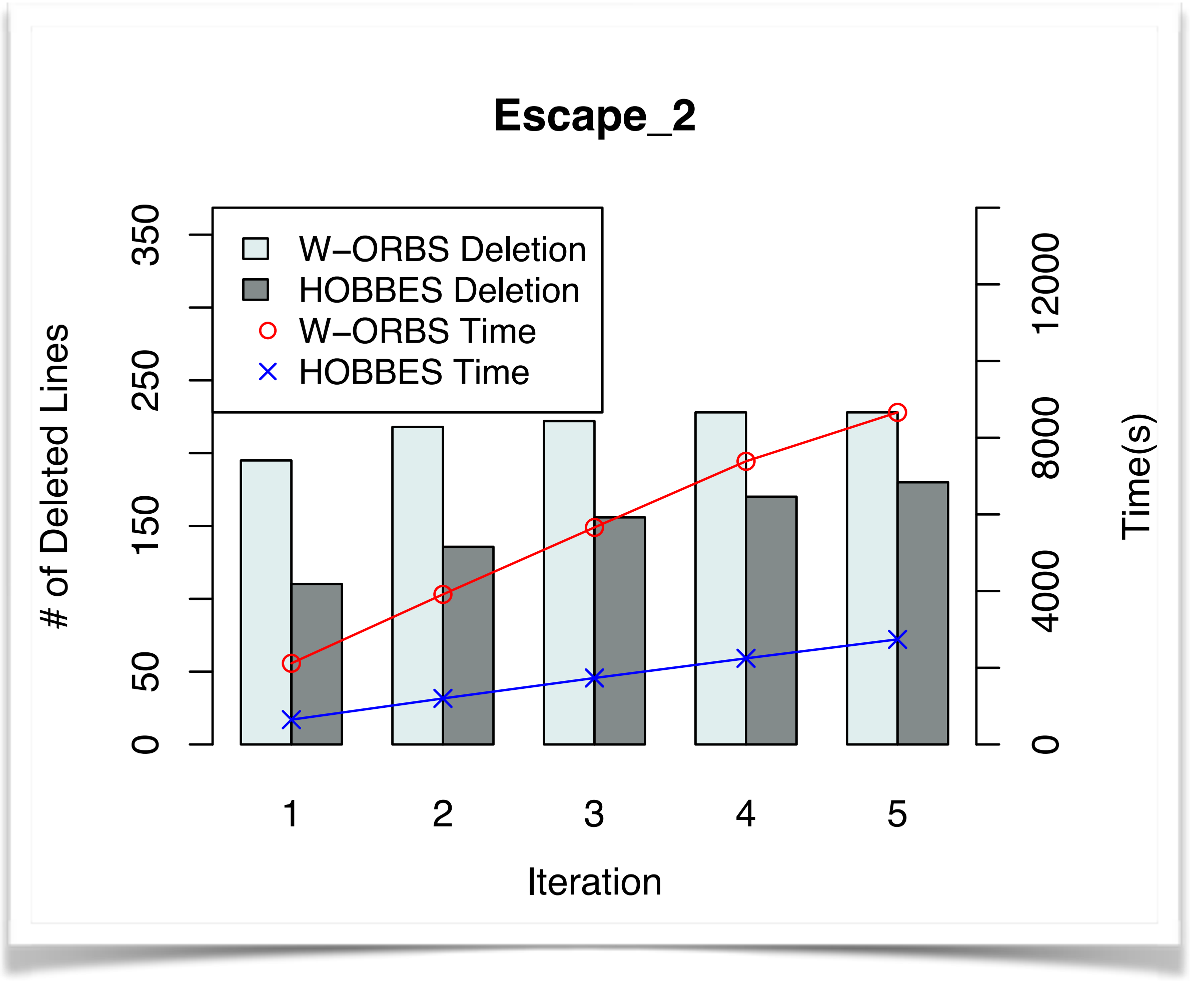
- Studied Deletion Operators
  - Window-Deletion of size 1, 2, 3, 4.
  - VSM-, LDA-Deletion of threshold 0.6, 0.7, 0.8, 0.9.
- Subject: Guava library
  - 2 slice criteria for each of subpackage ‘`escape`’ and ‘`net`’.
- Machine
  - Intel Core i7-6700K running Ubuntu 14.04.5 LTS.

HOBBES - Results

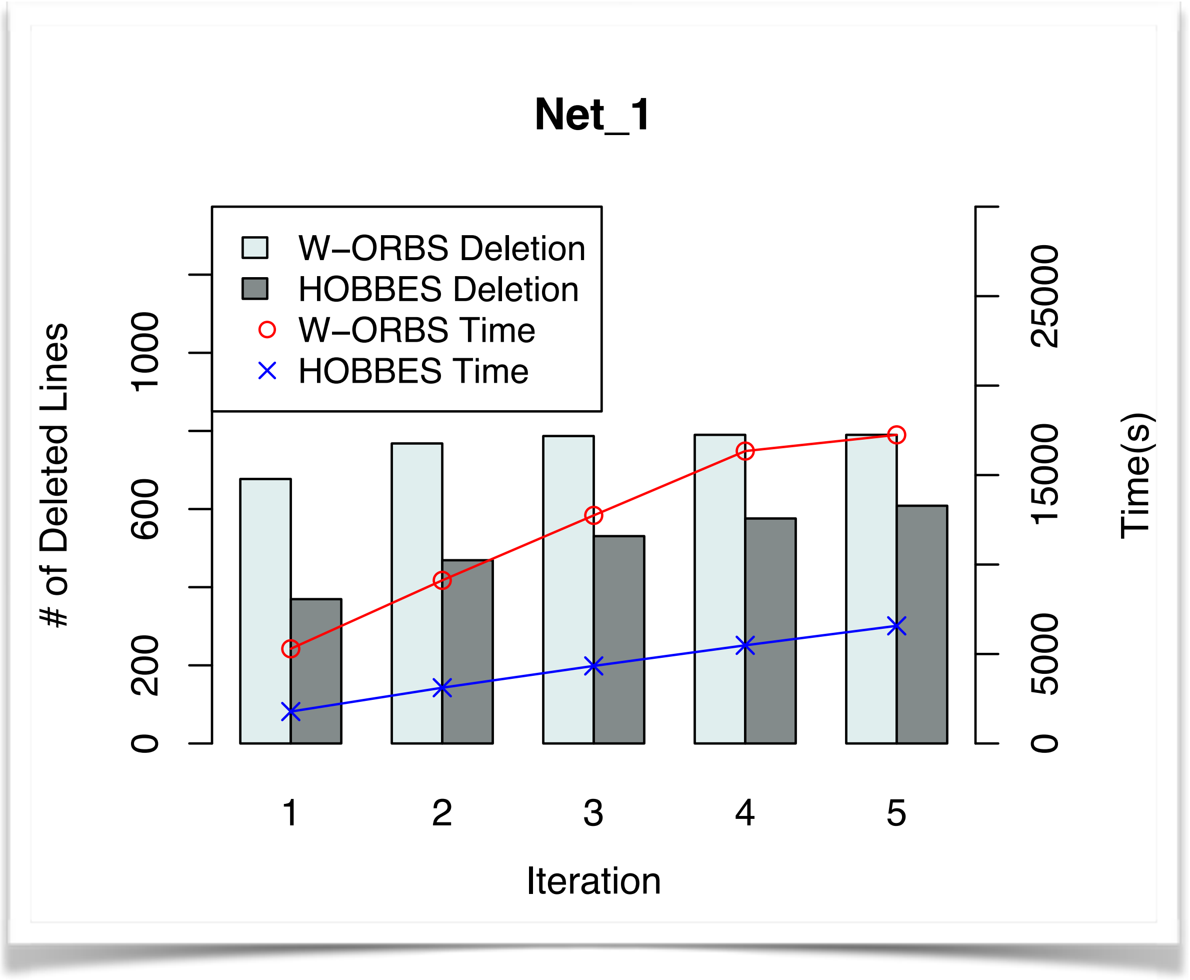
		Iter1			Iter2			Iter3			Iter4			Iter5		
Subject	Strategy	C	E	D/T	C	E	D/T	C	E	D/T	C	E	D/T	C	E	D/T
escape1	HOBBES	502	66	0.20	926	104	0.13	1321	135	0.11	1699	165	0.09	2060	192	0.09
	W-ORBS	1711	183	0.10	3137	267	0.06	4523	342	0.04	5840	415	0.03	NA	NA	NA
escape2	HOBBES	1332	214	0.21	2424	309	0.15	3430	388	0.12	4384	455	0.11	5289	516	0.09
	W-ORBS	4179	655	0.13	7383	922	0.08	10436	1159	0.06	13460	1390	0.05	14116	1558	0.05
net1	HOBBES	513	70	0.17	955	114	0.11	1374	154	0.09	1771	189	0.08	2154	224	0.07
	W-ORBS	1759	189	0.09	3251	280	0.06	4707	364	0.04	6141	448	0.03	7174	517	0.03
net2	HOBBES	1341	222	0.20	2444	324	0.14	3460	402	0.11	4425	473	0.10	5346	536	0.09
	W-ORBS	4332	667	0.11	7781	963	0.07	11077	1237	0.05	14337	1504	0.04	14993	1672	0.04

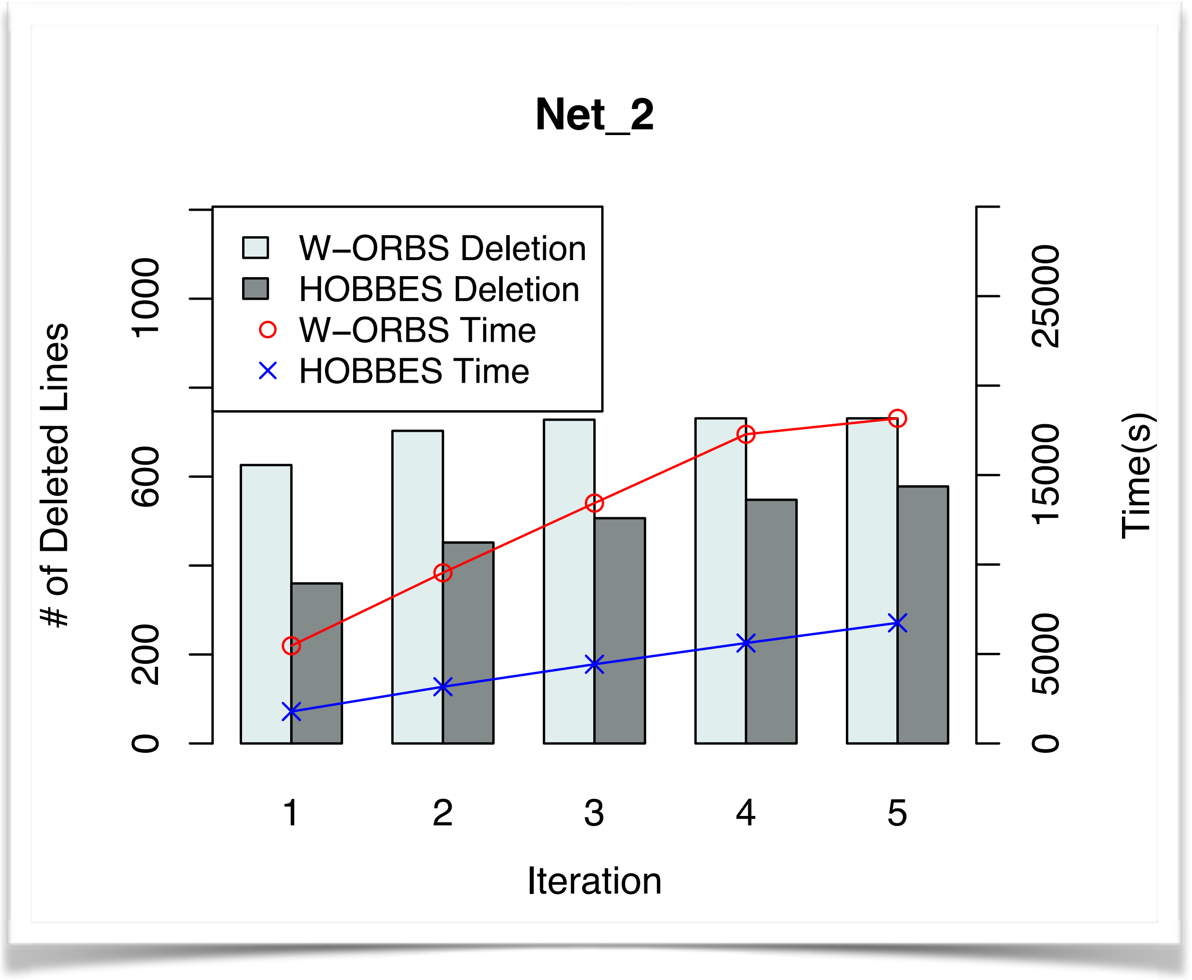
HOBBES - Results



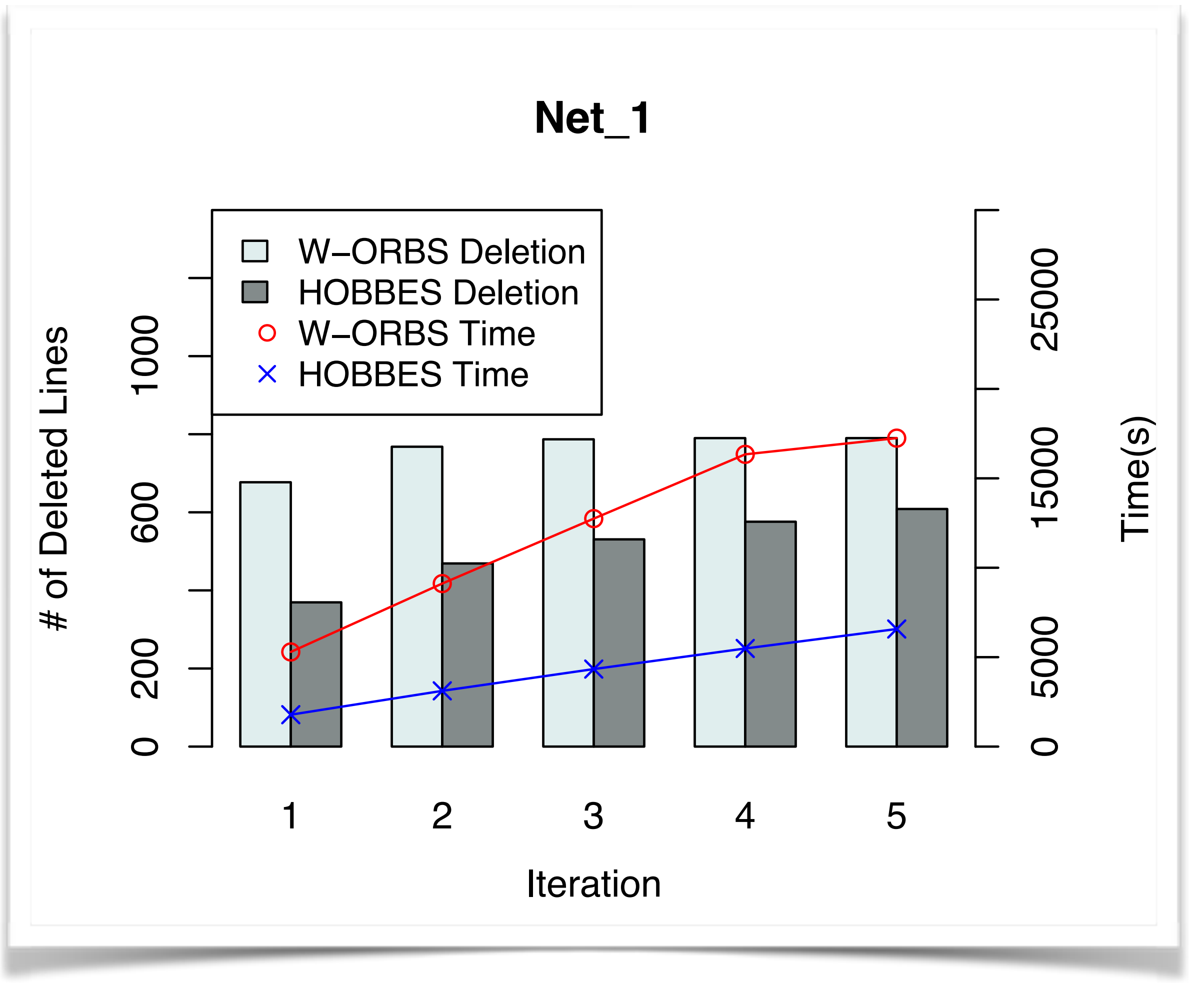


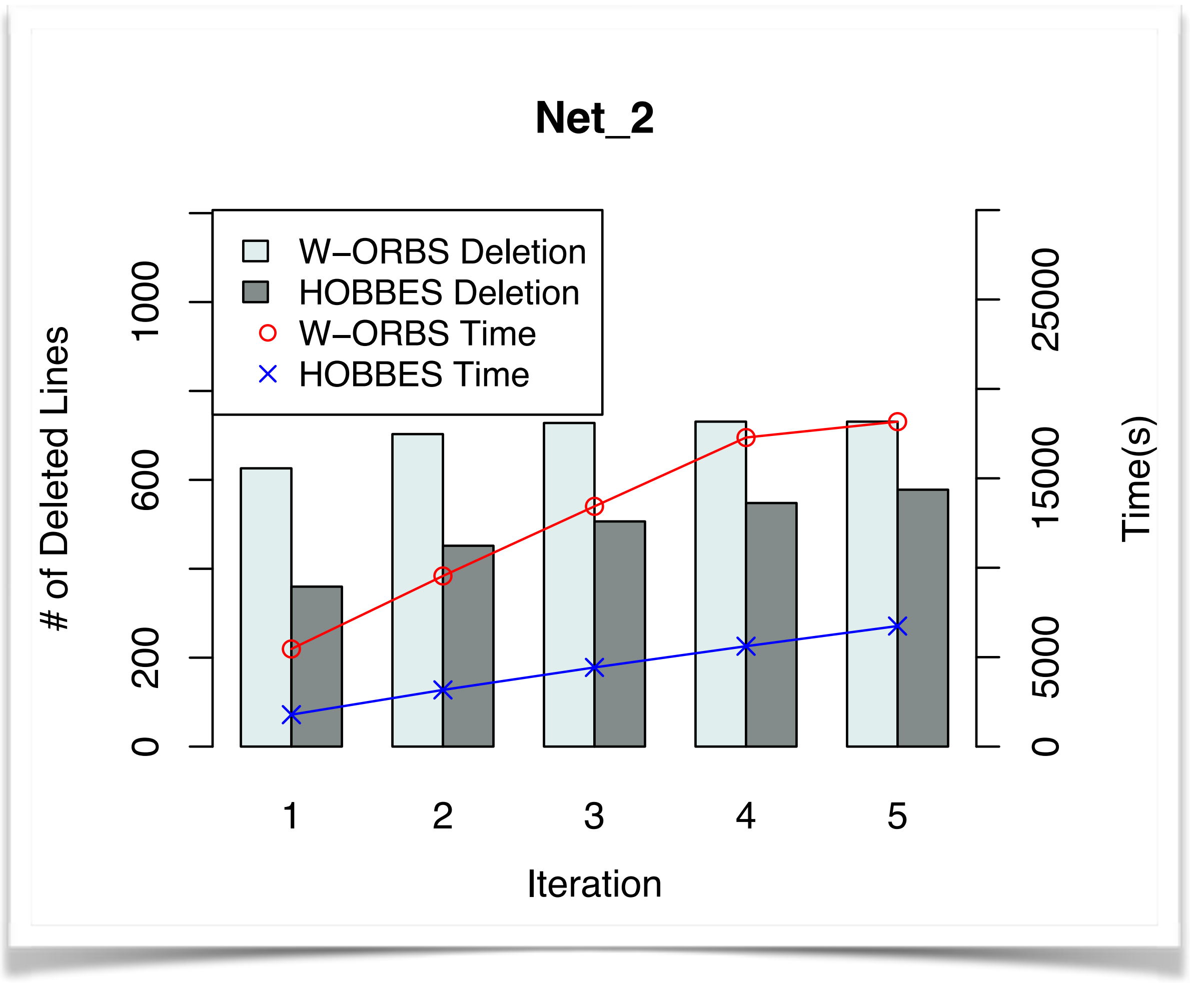
HOBBES - Results





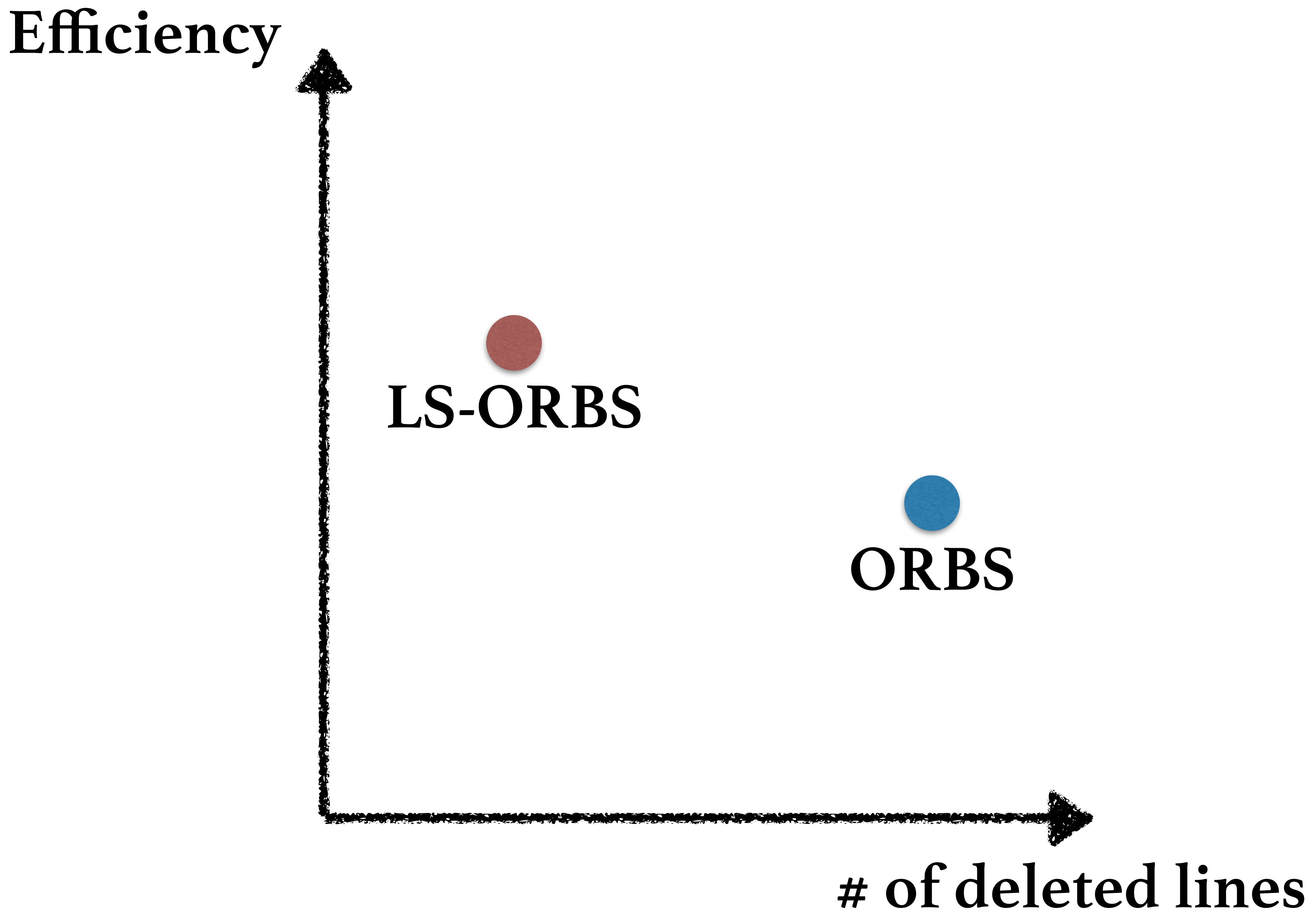
HOBBES - Results



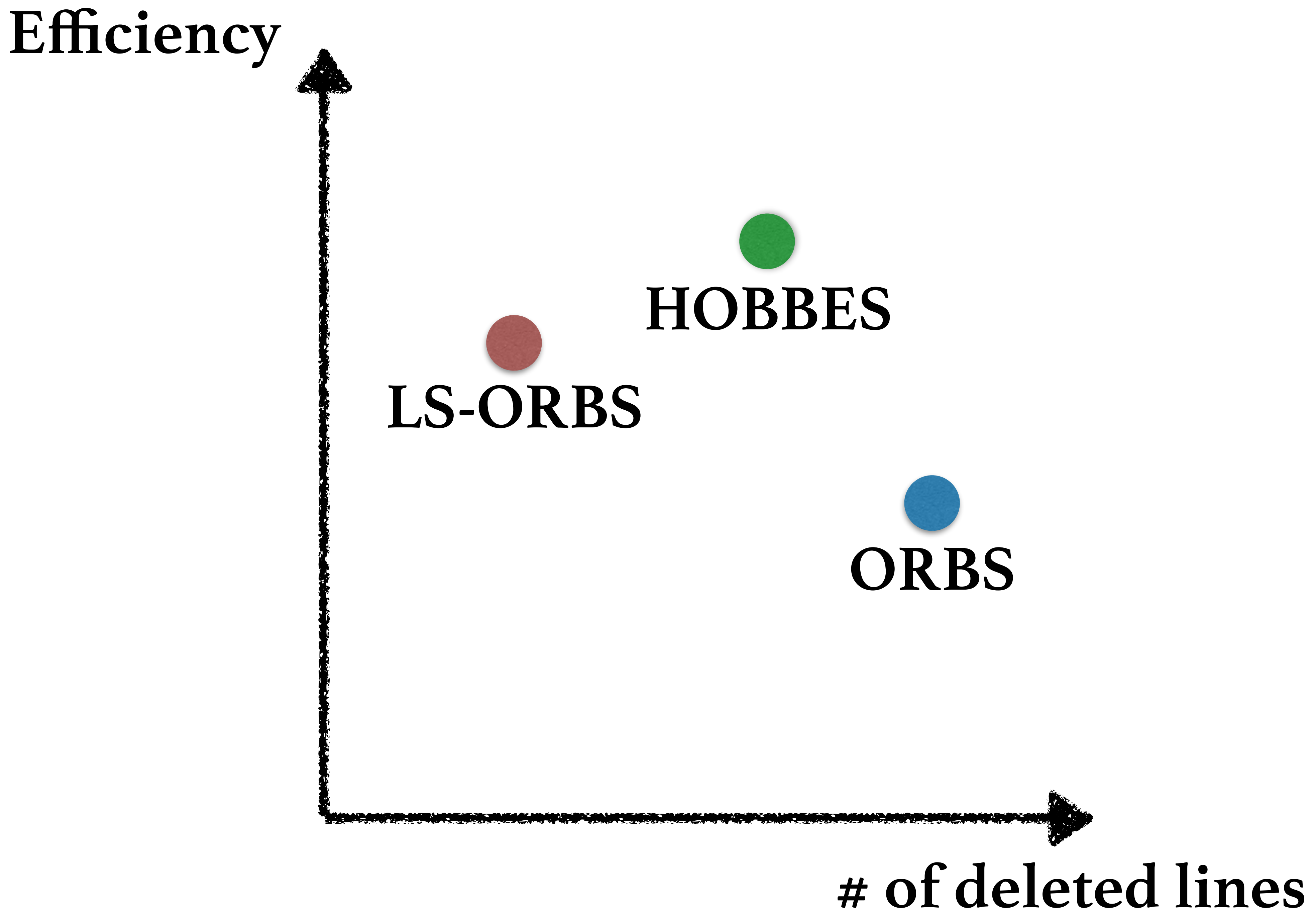


- HOBBES can delete about **71%** of the number of lines that ORBS deletes.
- However, HOBBES only takes about **30%** of the time spent by ORBS.

Again, Compare Strategies



Again, Compare Strategies



Future Work

- Investigate non-iterative application of deletions.
- Apply more sophisticated lexical analysis.
  - For example, token normalization

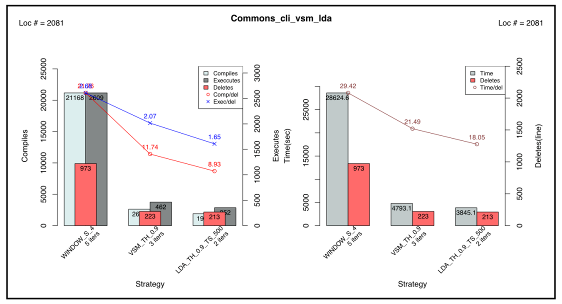
[“`open_file`”] → ["`open`", “`file`”]

Limitations of ORBS

- Scalability
  - Takes around 7200 s to delete 220 lines.
    ⇒ 0.03 del/s
    ⇒ 32.7 s/del

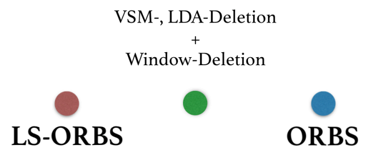
```
int main(){
    int sum = 0;
    int i = 1;
    while (i<11) {
        sum = sum+i;
        i = i + 1;
    }
    printf("%d \n", sum);
    printf("d \n", i);
}
```

Deletion based on Lexical Similarity



53.3% less compilations, **34.3%** less executions, **39.3%** less time per 1 deleted lines.

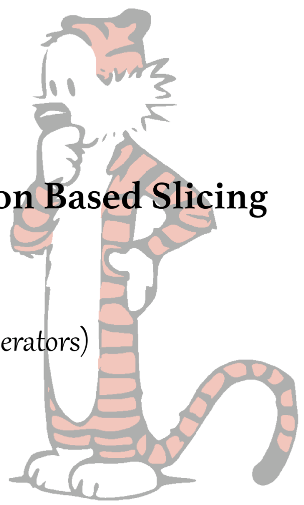
Compare Strategies



Hyperheuristic Observation Based Slicing

(HOBBES)

(On selecting deletion operators)

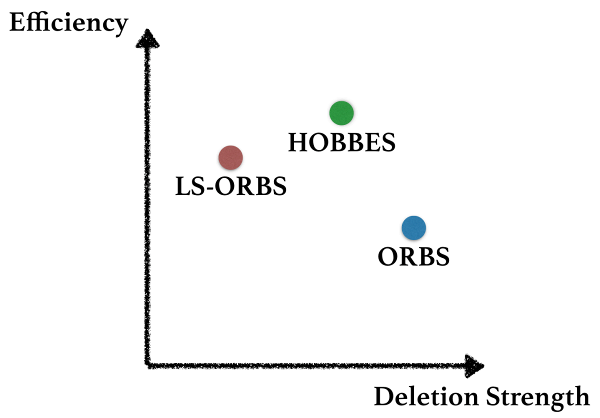


HOBBES - Results

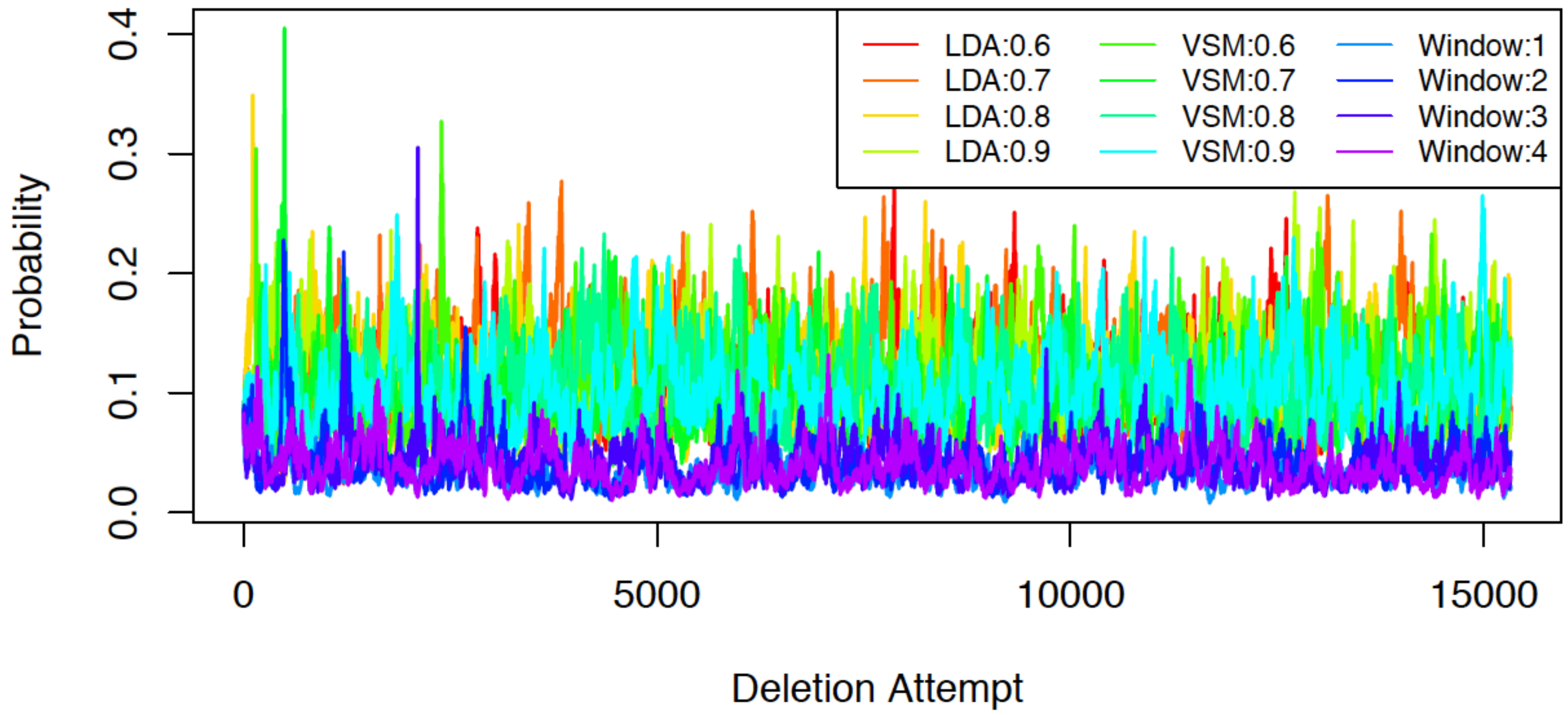
Subject	Strategy	Iter1			Iter2			Iter3			Iter4			Iter5		
		C	E	D/T	C	E	D/T	C	E	D/T	C	E	D/T	C	E	D/T
escape1	HOBBES	502	66	0.20	926	104	0.13	1321	135	0.11	1699	165	0.09	2060	192	0.09
	W-ORBS	1711	183	0.10	3137	267	0.06	4523	342	0.04	5840	415	0.03	NA	NA	NA
escape2	HOBBES	1332	214	0.21	2424	309	0.15	3430	388	0.12	4384	455	0.11	5289	516	0.09
	W-ORBS	4179	655	0.13	7383	922	0.08	10436	1159	0.06	13460	1390	0.05	14116	1558	0.05
net1	HOBBES	513	70	0.17	955	114	0.11	1374	154	0.09	1771	189	0.08	2154	224	0.07
	W-ORBS	1759	189	0.09	3251	280	0.06	4707	364	0.04	6141	448	0.03	7174	517	0.03
net2	HOBBES	1341	222	0.20	2444	324	0.14	3460	402	0.11	4425	473	0.10	5346	536	0.09
	W-ORBS	4332	667	0.11	7781	963	0.07	11077	1237	0.05	14337	1504	0.04	14993	1672	0.04

- HOBBES can delete about **71%** of the number of lines that ORBS deletes.
- However, HOBBES only takes about **30%** of the time spent by ORBS.

Again, Compare Strategies



How the selection probability of deletion operators changed?


LDA:0.6
LDA:0.7
LDA:0.8
LDA:0.9
VSM:0.6
VSM:0.7
VSM:0.8
VSM:0.9
Window:1
Window:2
Window:3
Window:4
Probability
Deletion Attempt