Optimizing Constraint Solving to Better Support Symbolic Execution

Ikpeme Erete and Alessandro Orso
School of Computer Science – College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, and MSR

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.         // do something
11.     else
12.       // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches:

Symbolic state:

Path condition (PC):

DSE:

Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13. return
14. }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches:

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0$

Path condition (PC):

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.         // do something
11.     else
12.       // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches:

Symbolic state:

$a=a_0$, $b=b_0$, $c=c_0$, $d=d_0$

Path condition (PC):

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.         // do something
11.     else
12.       // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T,

Symbolic state:

$a=a_0$, $b=b_0$, $c=c_0$, $d=d_0$

Path condition (PC):

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.         // do something
11.     else
12.       // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0$

Path condition (PC):

$(c_0 > a_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T,

Symbolic state:

$a=a_0$, $b=b_0$, $c=c_0$, $d=d_0$

Path condition (PC):

$(c_0 > a_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.        // do something
06.     else if (a < e)
07.        if (b < c)
08.           // do something
09.        else
10.           // do something
11.     else
12.        // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5)$

DSE:

Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13. return
14. }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5)$

DSE:

Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13. return
14. }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10)$

DSE:

Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13. return
14. }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T,

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.        // do something
06.     else if (a < e)
07.        if (b < c)
08.           // do something
09.        else
10.           // do something
11.     else
12.        // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10)$

DSE:

Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13. return
14. }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13. return
14. }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DSE:

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DSE:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10)$

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DSE:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

Background: Dynamic Symbolic Execution

```
01.  foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14.  }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:

$a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10$

Path condition (PC):

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DSE:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 >= d_0 + 10)$

$(c_0 > b_0) \wedge (b_0 > 5)$

$(c_0 <= b_0)$

Symbolic Execution

Symbolic Execution

Program

Symbolic Execution

Program

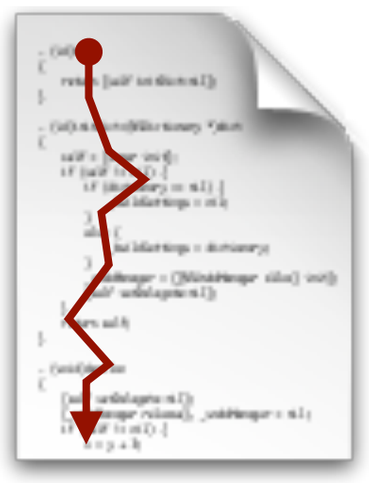
Symbolic executor

Symbolic Execution

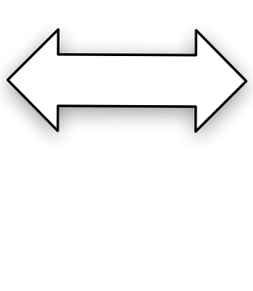
Program

Symbolic executor

Symbolic Execution

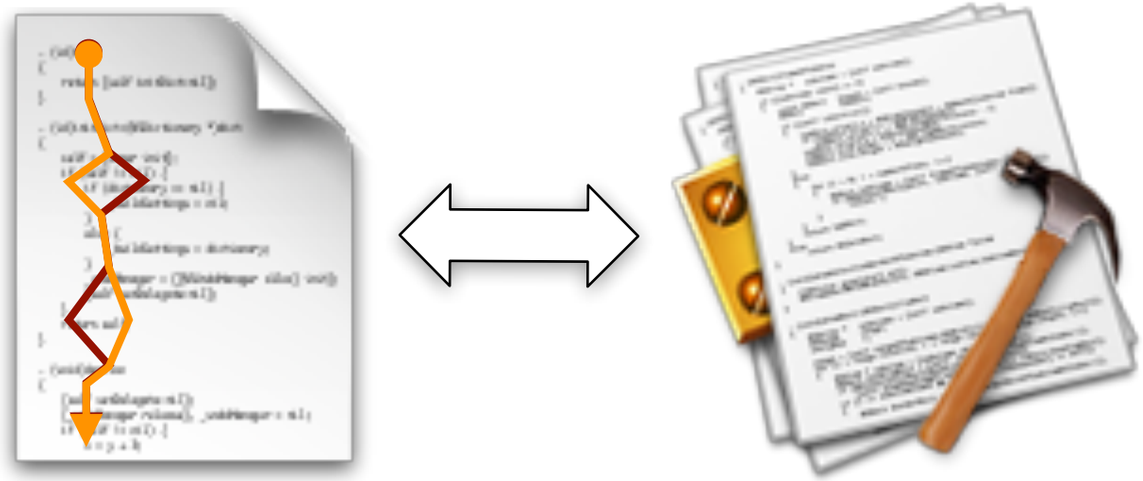


Program



Symbolic executor

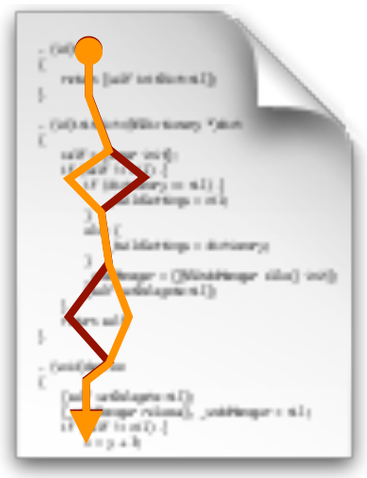
Symbolic Execution



Program

Symbolic executor

Symbolic Execution



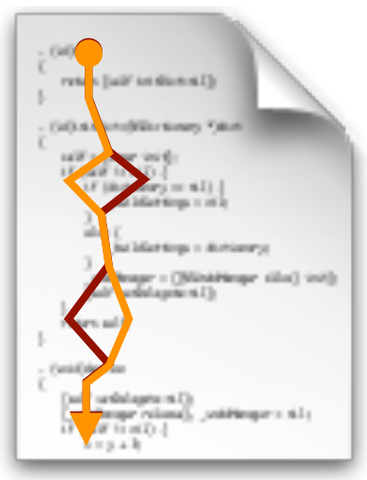
Program

PC

$c_1 \wedge \ldots \wedge c_n$

Symbolic executor

Symbolic Execution



Program

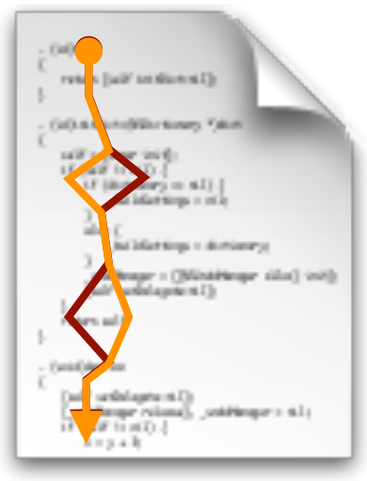
PC
$c_1 \wedge \ldots \wedge c_n$

Symbolic executor

PC
$c_1 \wedge \ldots \wedge c_n$

SMT solver

Symbolic Execution



Program

PC

$c_1 \wedge \ldots \wedge c_n$

Symbolic executor

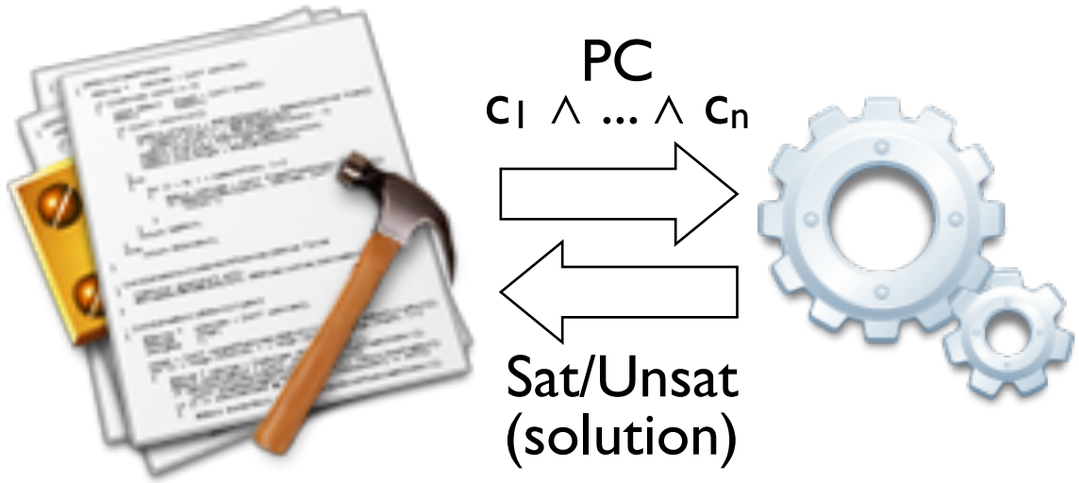
PC

$c_1 \wedge \ldots \wedge c_n$

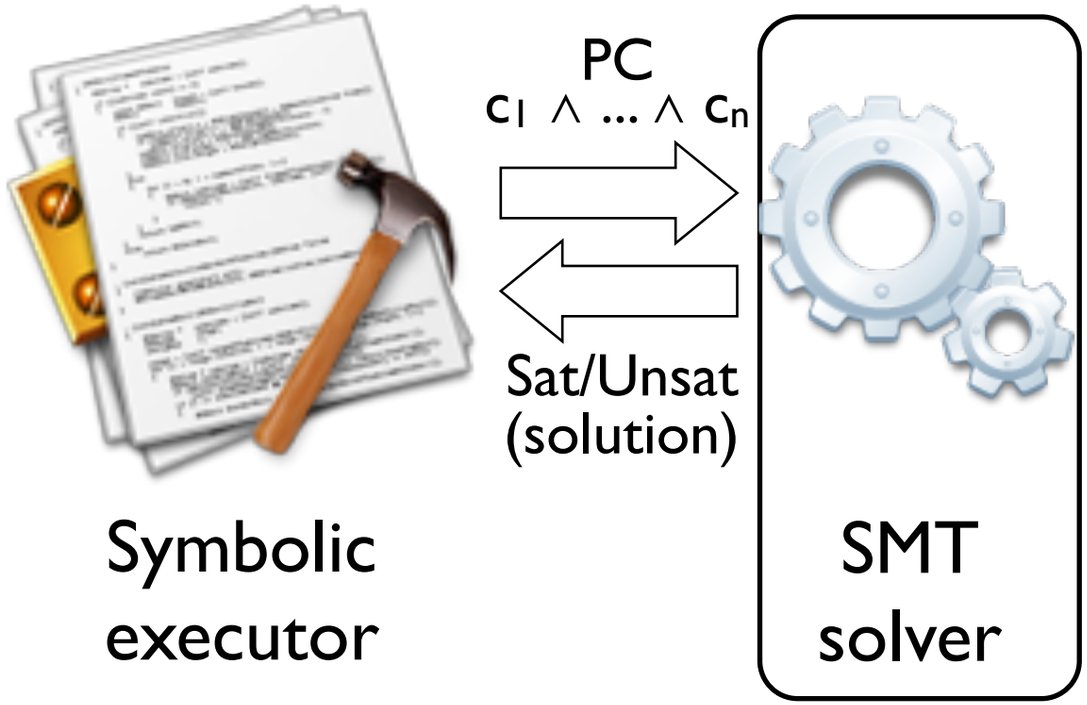
Sat/Unsat (solution)

SMT solver

Symbolic Execution and SMT Solving



Symbolic Execution and SMT Solving



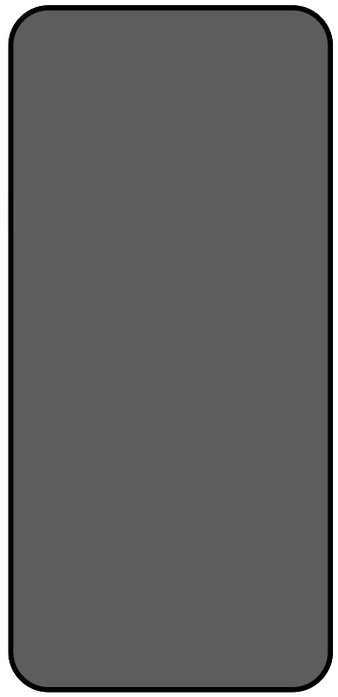
Symbolic Execution and SMT Solving

Symbolic executor

PC

$c_1 \wedge \ldots \wedge c_n$

Sat/Unsat (solution)



What Are We Missing?

- **Context information** (e.g., existence of previous solutions for similar PCs)
- **Domain knowledge** (e.g., programs' specific properties)

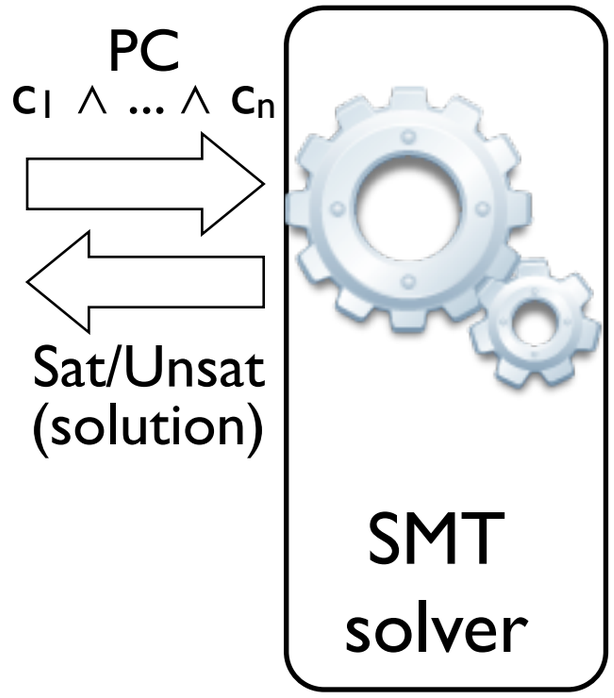
State of the Art

- Some techniques present initial solutions (domain-based constraint optimizations)
- But:
  - What is the effectiveness of these techniques?
  - What other techniques could be used?
  - Would symbolic execution actually benefit from these techniques?

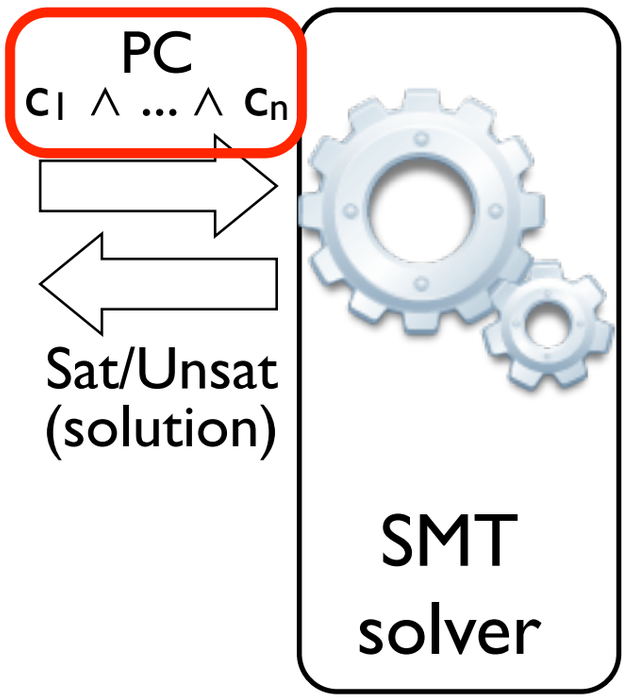
Our Goal

- Initial investigation of these questions by
  - proposing a novel constraint optimization technique for dynamic symbolic execution: DomainReduce
  - performing an empirical evaluation to assess new and existing optimizations empirically

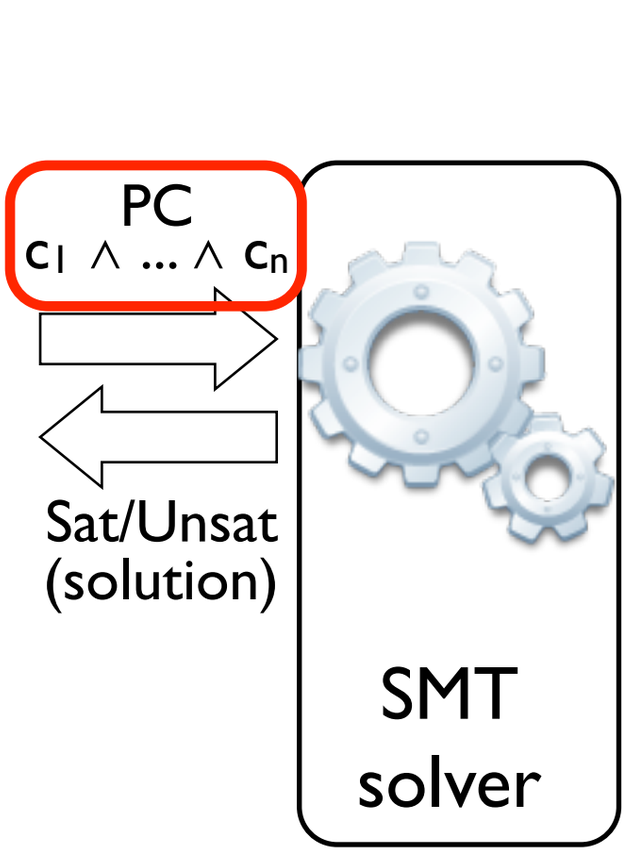
DomainReduce: Intuitive View

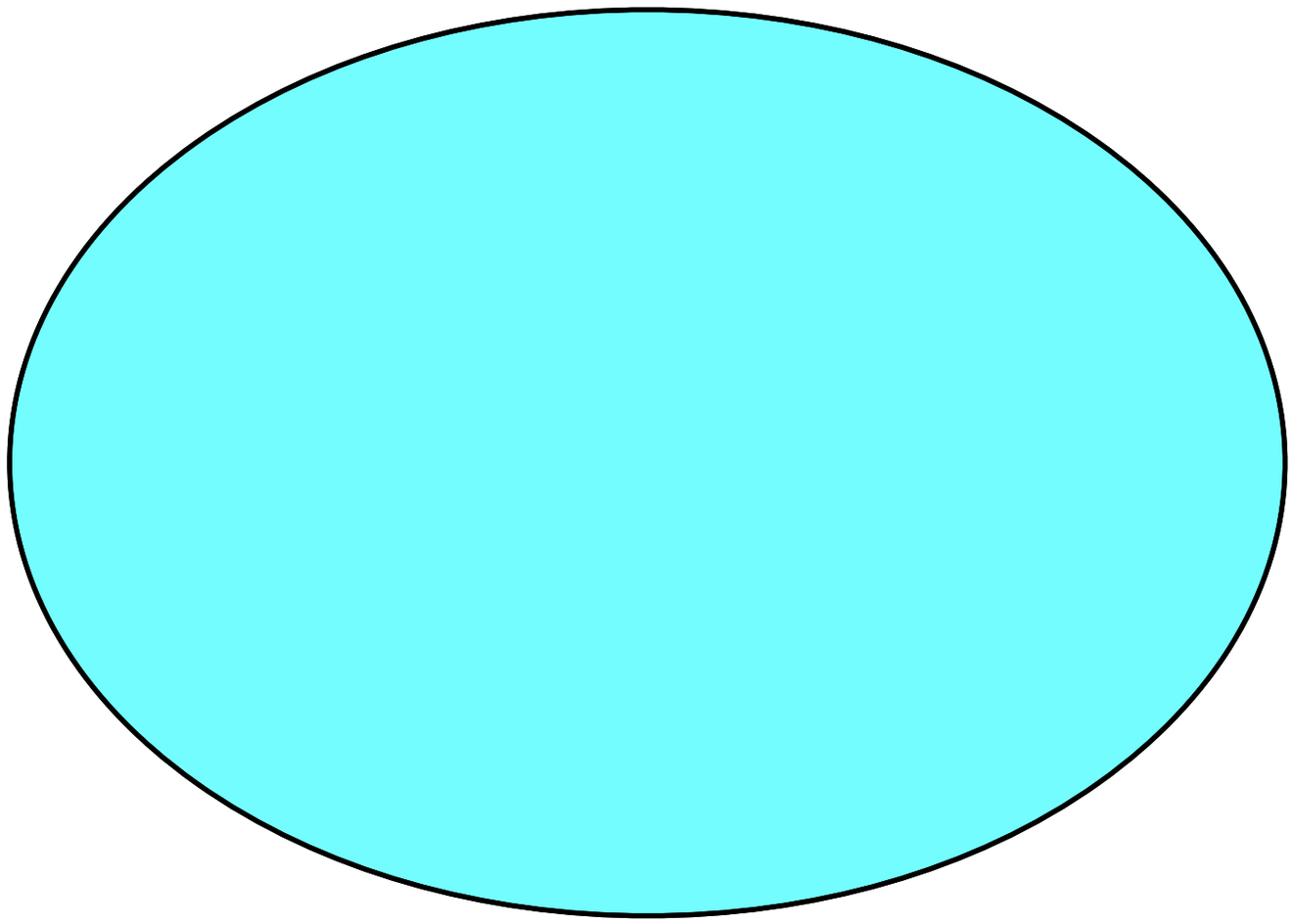


DomainReduce: Intuitive View



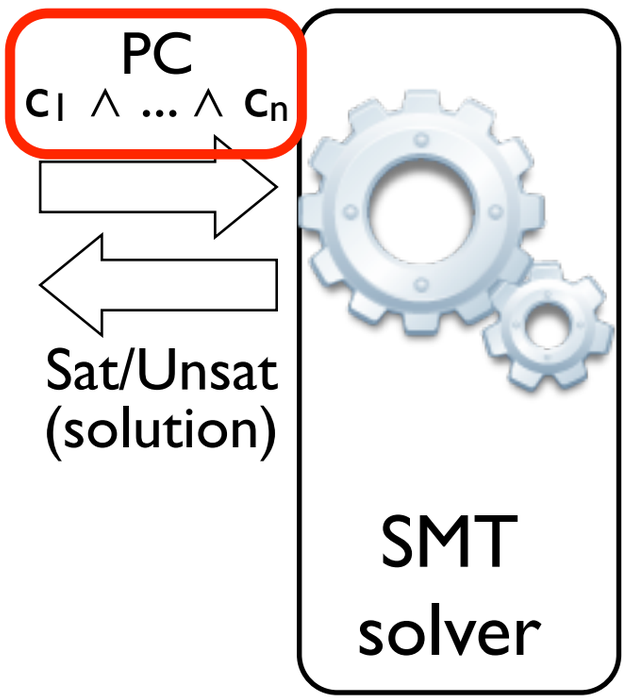
DomainReduce: Intuitive View

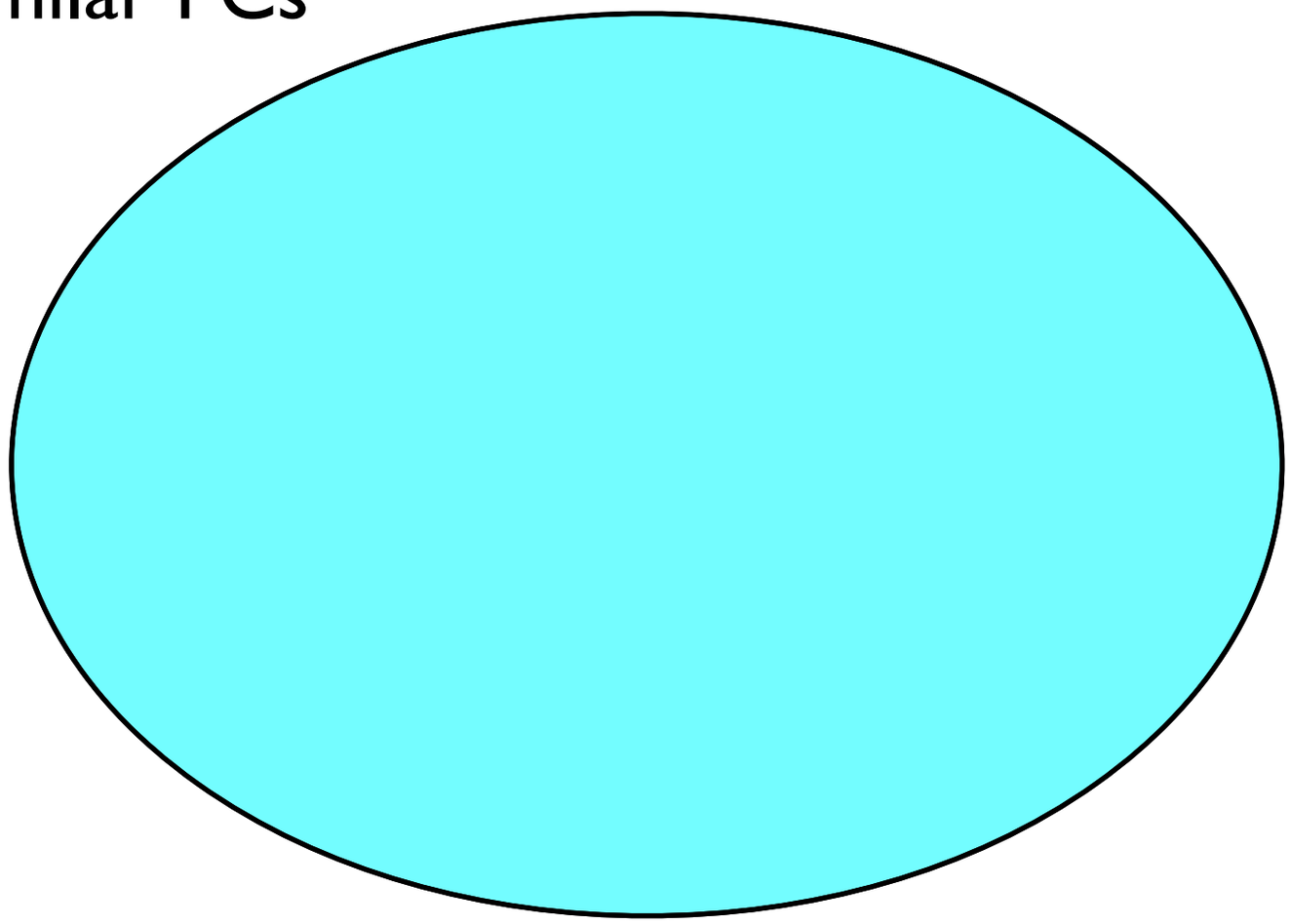




DomainReduce: Intuitive View

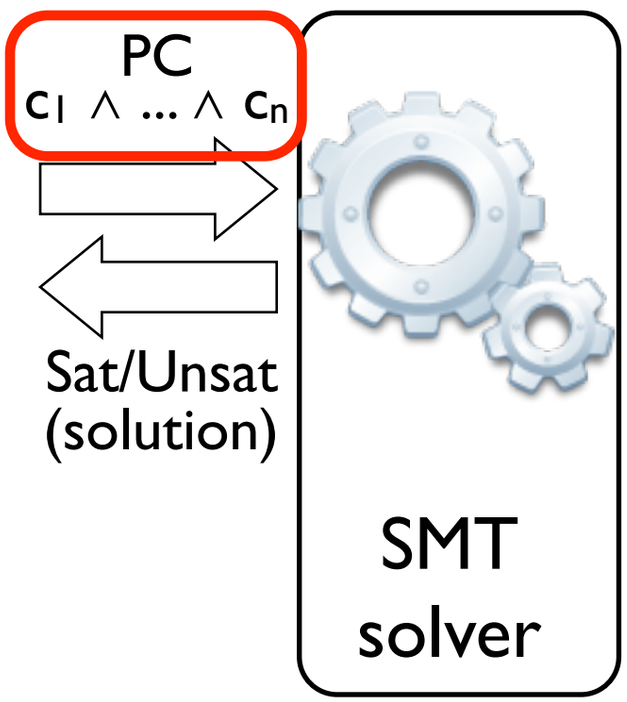
Restrict domain of constraints to be solved
by leveraging solutions of similar PCs

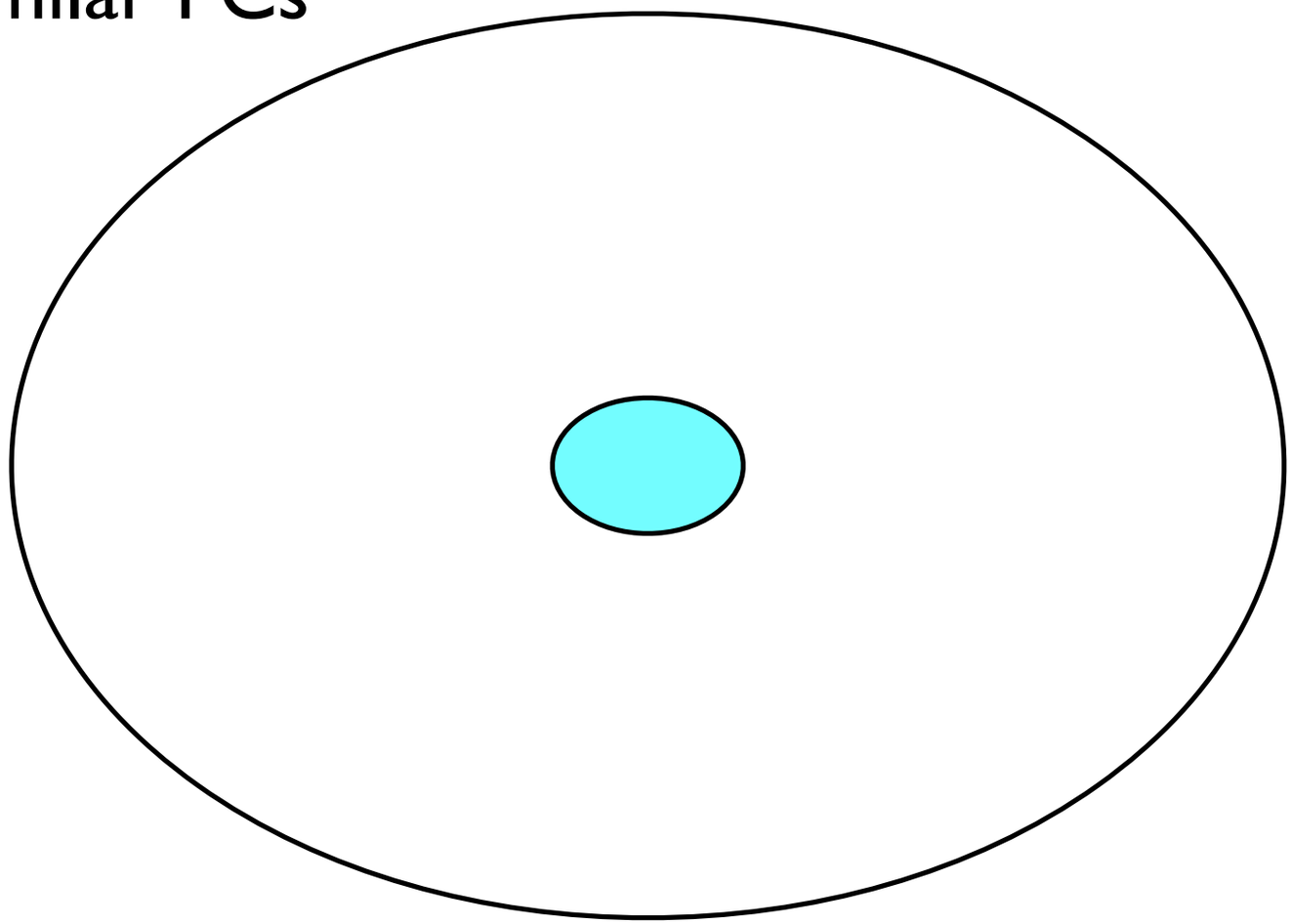




DomainReduce: Intuitive View

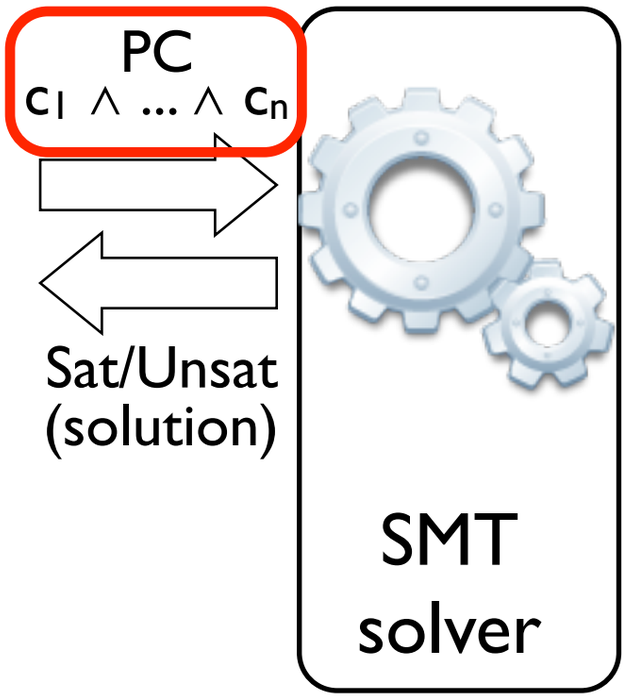
Restrict domain of constraints to be solved
by leveraging solutions of similar PCs

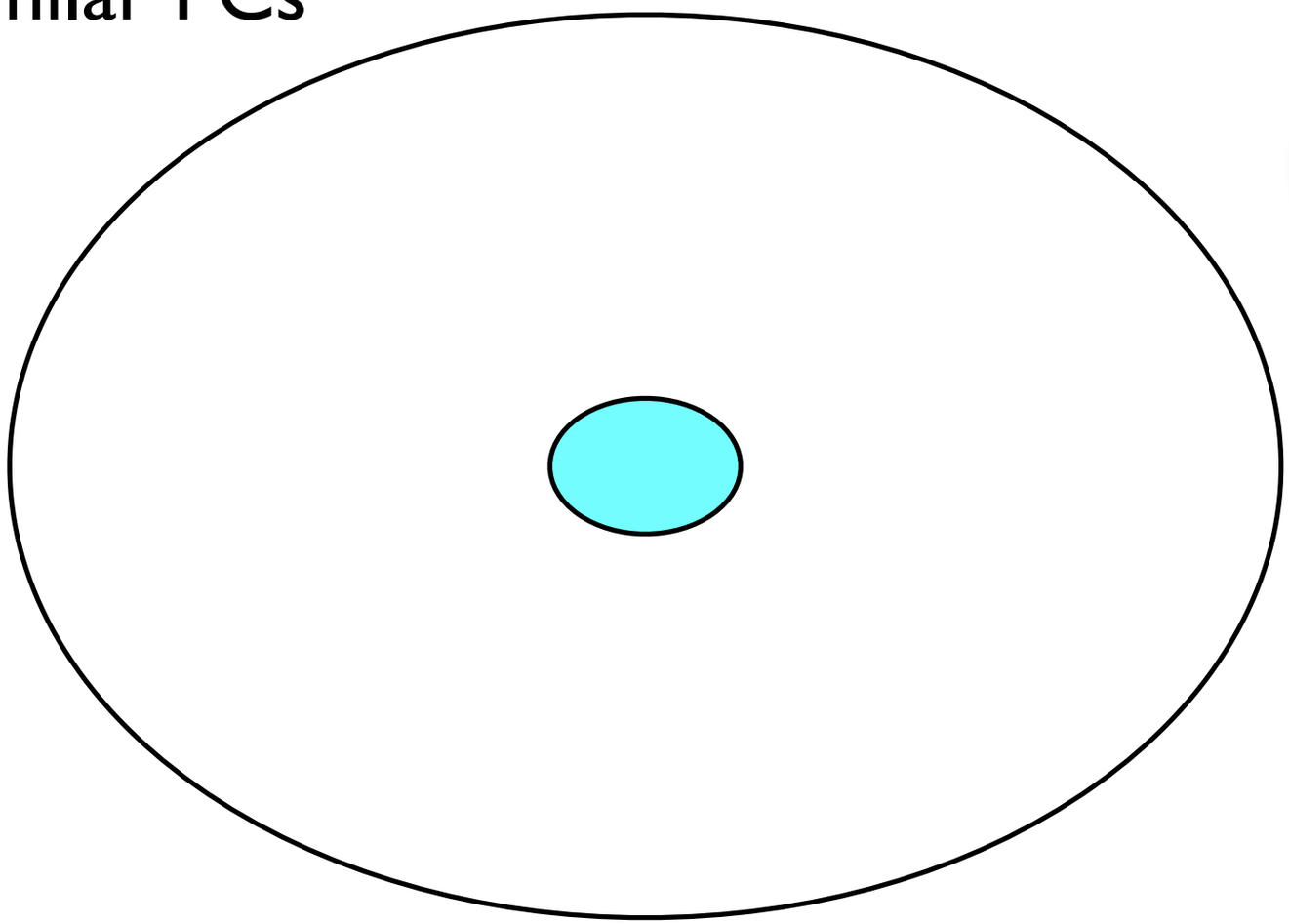




DomainReduce: Intuitive View

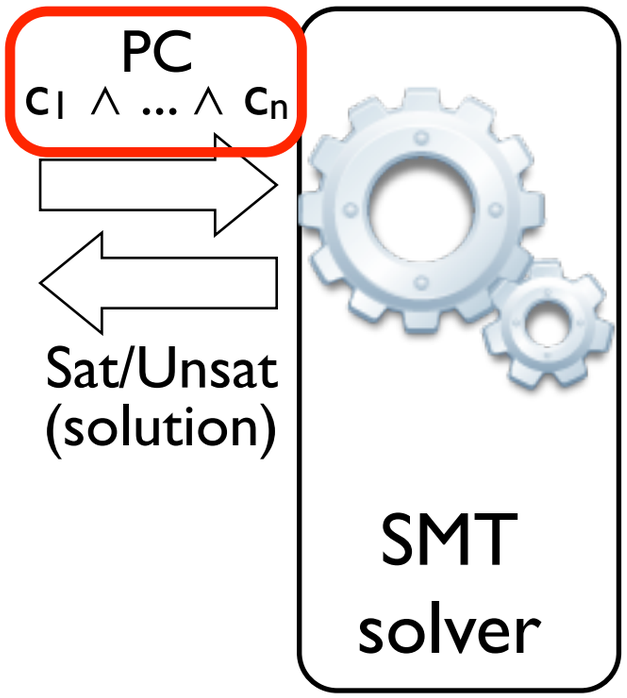
Restrict domain of constraints to be solved
by leveraging solutions of similar PCs

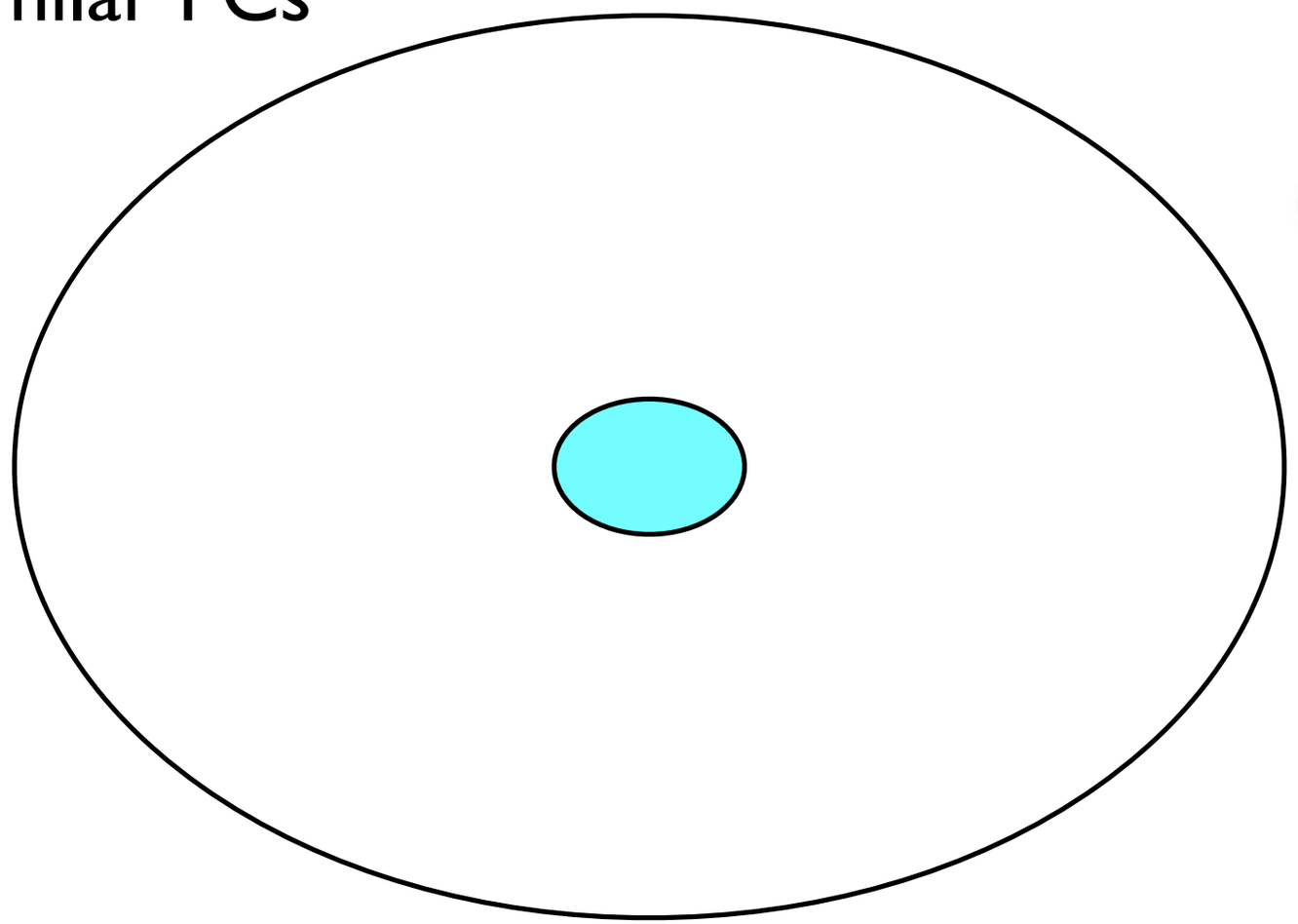




DomainReduce: Intuitive View

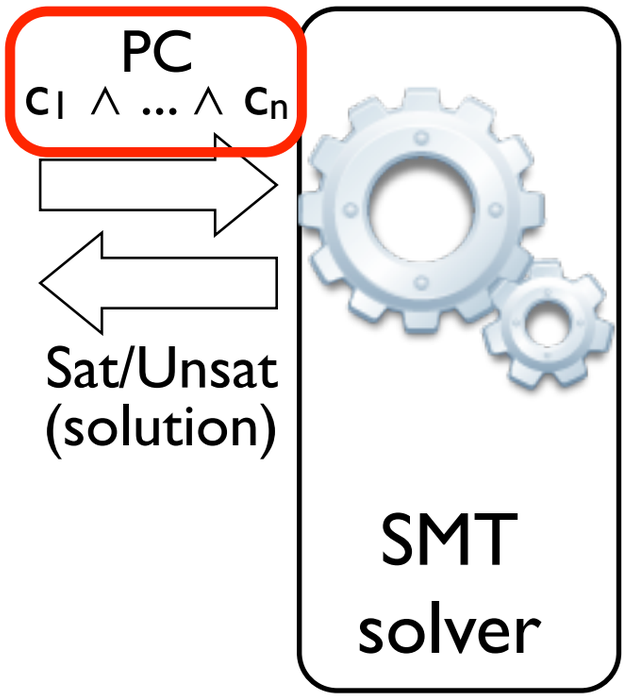
Restrict domain of constraints to be solved
by leveraging solutions of similar PCs

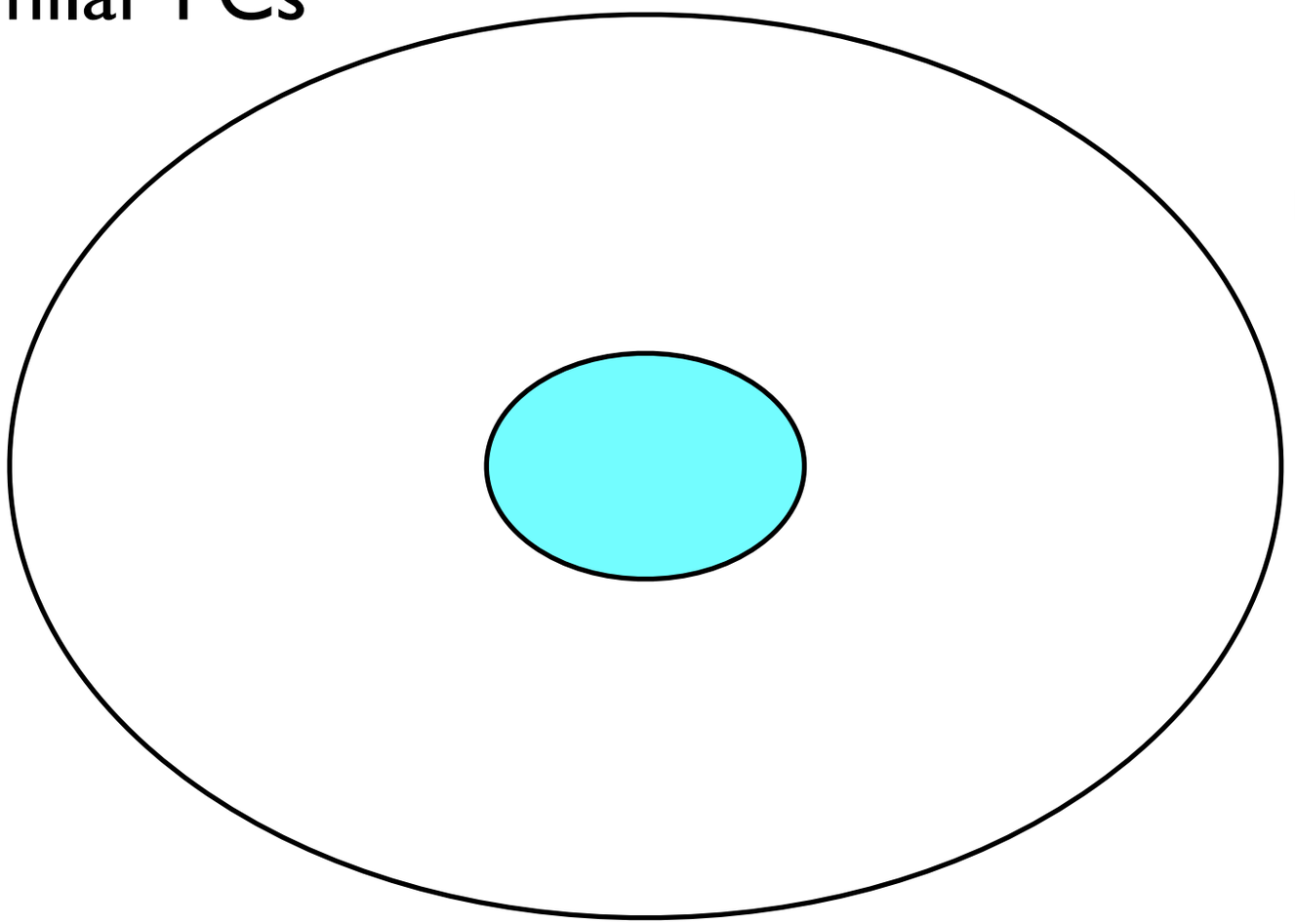




DomainReduce: Intuitive View

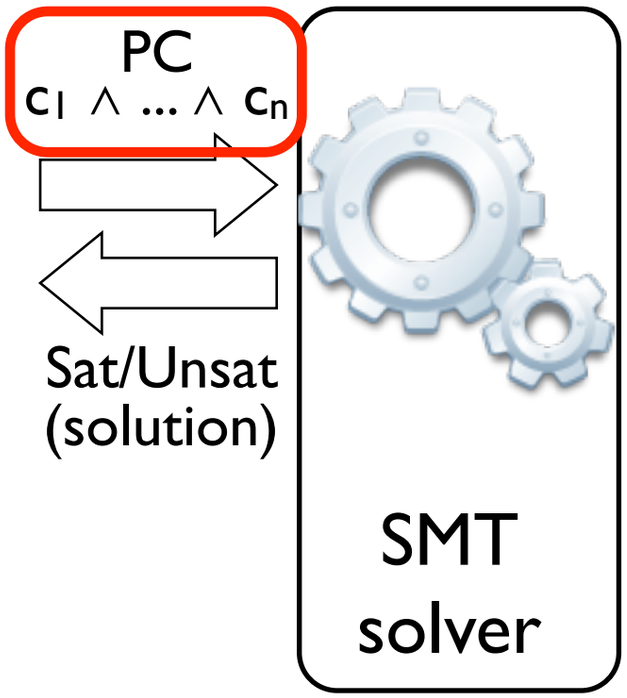
Restrict domain of constraints to be solved
by leveraging solutions of similar PCs

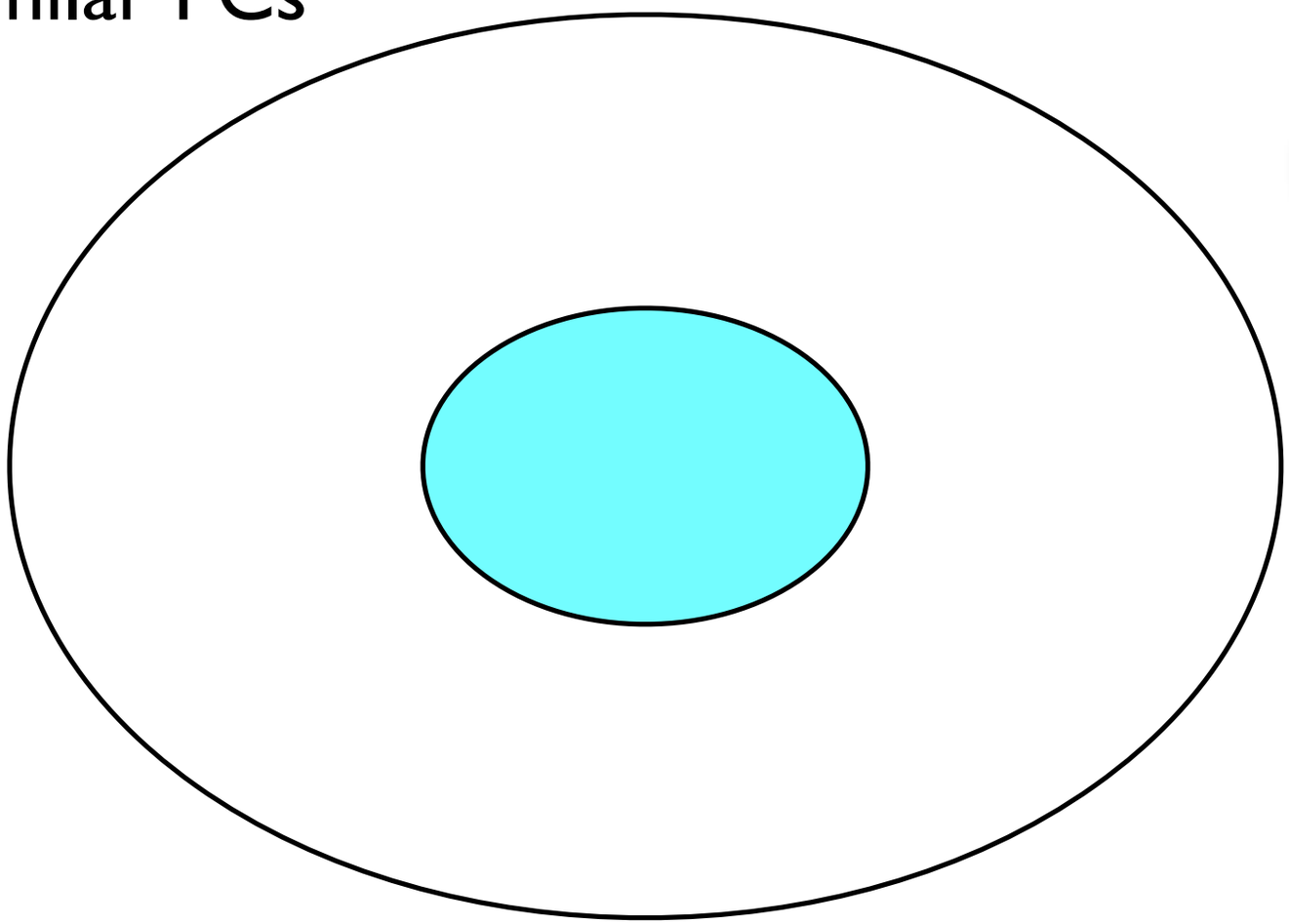




DomainReduce: Intuitive View

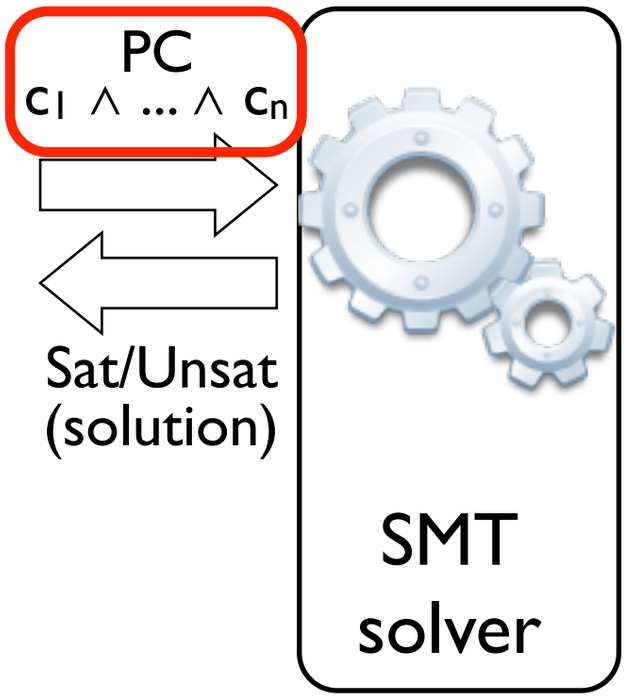
Restrict domain of constraints to be solved
by leveraging solutions of similar PCs

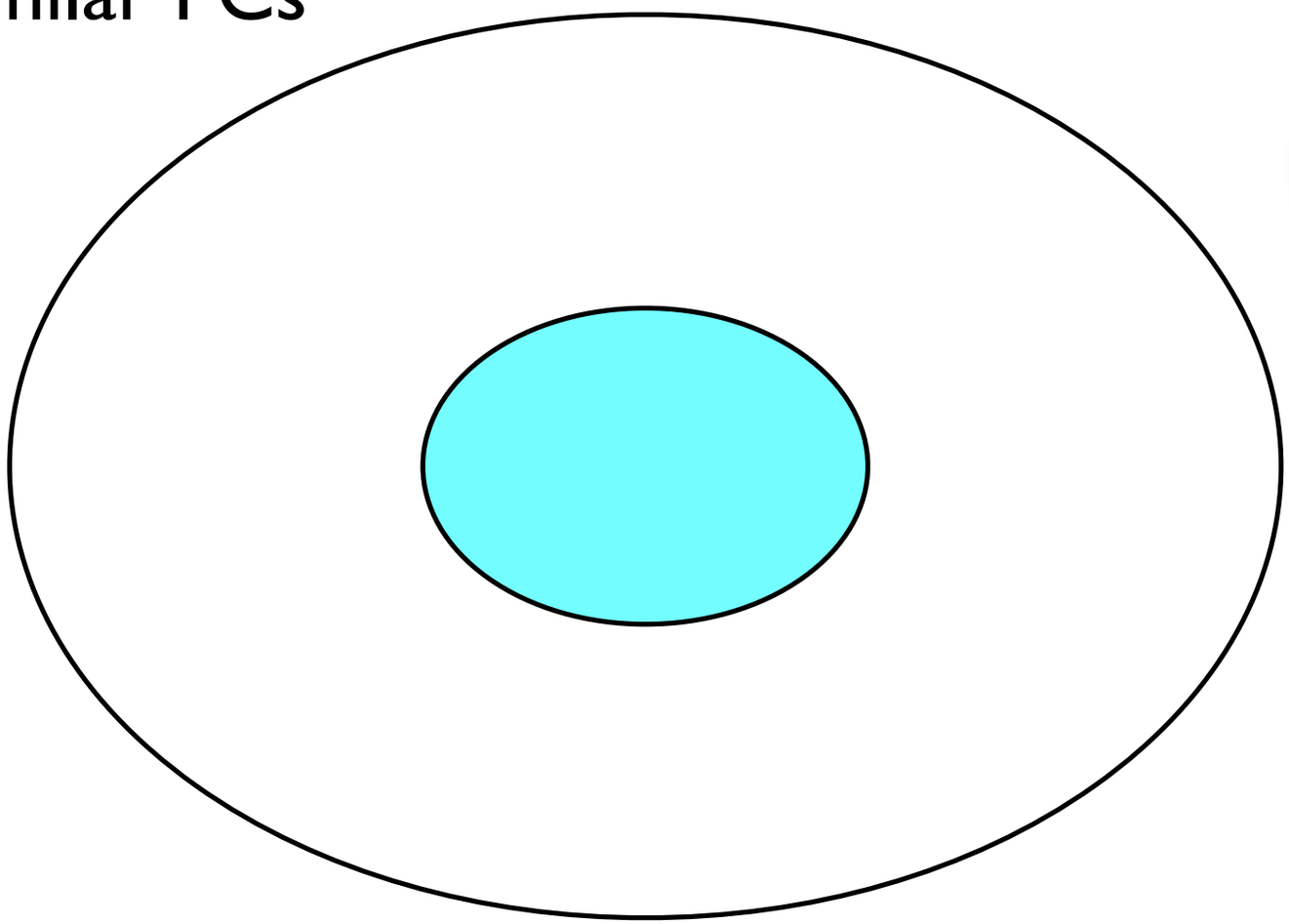




DomainReduce: Intuitive View

Restrict domain of constraints to be solved
by leveraging solutions of similar PCs





Trade-off speed/likelihood of finding solutions

DomainReduce Example
(with dependencies)

DomainReduce Example
(with dependencies)

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0)$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

$$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

$$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

$$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

$$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$$ ✘

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

With dependencies:

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

$$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$$ ✘

$$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$$

$$(c_0 > a_0) \wedge (5 <= 5) \wedge (a_0 < d_0 + 10) \wedge (5 >= c_0)$$

DomainReduce Example
(with dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

With dependencies:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$ ✘

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(c_0 > a_0) \wedge (5 <= 5) \wedge (a_0 < d_0 + 10) \wedge (5 >= c_0)$ ✔

DomainReduce Example (without dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

Without dependencies:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$ ✘

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

DomainReduce Example (without dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

Without dependencies:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$ ✘

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(c_0 > 4) \wedge (5 <= 5) \wedge (4 < 1 + 10) \wedge (5 >= c_0)$

DomainReduce Example (without dependencies)

$$\begin{cases} (c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 < c_0) \\ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \end{cases}$$

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

Without dependencies:

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(5 > 4) \wedge (b_0 <= 5) \wedge (4 < 1 + 10) \wedge (b_0 >= 6)$ ✘

$(c_0 > a_0) \wedge (b_0 <= 5) \wedge (a_0 < d_0 + 10) \wedge (b_0 >= c_0)$

$(c_0 > 4) \wedge (5 <= 5) \wedge (4 < 1 + 10) \wedge (5 >= c_0)$ ✔

Terminology

- Target constraint:
  negated constraint
- Target variables:
  variables in negated constraint
- Direct dependency (c_a, c_b):
  $vars(c_a) \cap vars(c_b) \neq 0$
- Indirect dependency (c_a, c_b):
  $vars(c_a) \cap vars(c_1) \neq 0, vars(c_1) \cap vars(c_2) \neq 0, ...,$
  $vars(c_n) \cap vars(c_b) \neq 0$

DomainReduce Algorithm

- s = 1
- until sat or s=max or time limit reached
  - select next subset TV of target variables of size s
  - if no more subsets, increase s and reiterate
  - identify variables dependents on TV and add them to TV
  - keep variables in TV symbolic
  - concretize all other variables
  - invoke solver

Other Techniques Considered

- Incremental solving (Sen et al, 2005)
  - Eliminates irrelevant constraints
  - Analogous to worst case for DomainReduce with dependencies
- Subsumption (Godefroid et al, 2008)
  - Eliminates implied constraints in input-bound loops
  - In hindsight, not really a constraint-optimization approach

Empirical Evaluation

Goal: Quantitative initial investigation of the usefulness of constraint optimization

RQ1: Are constraint optimization techniques effective?

RQ2: How do the different techniques compare to each other?

Experimental Infrastructure

- **Tool**
  Customized JFuzz/JPF framework
- **Software subjects**
  HTMLParser
  XMLParser
  K-Nearest Neighbor
- **Solvers**
  CVC3, Z3
- **Data**
- Ten input sets per subject
- Over 5,000 real path conditions; ∀ technique and constraint:
  – Number of path conditions solved by the technique
  – Time necessary to solve the condition (10 minutes timeout)

Experimental Infrastructure

- **Tool**
  Customized JFuzz/JPF framework
- **Software subjects**
  HTMLParser
  XMLParser
  K-Nearest Neighbor
- **Solvers**
  CVC3, Z3
- **Data**
- Ten input sets per subject
- Over 5,000 real path conditions; ∀ technique and constraint:
  – Number of path conditions solved by the technique
  – Time necessary to solve the condition (10 minutes timeout)

Infrastructure and data freely available online:
http://www.cc.gatech.edu/~ikpeme/software/

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser not compelling

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
  - DomainReduce produces negative results for K-NN (worst case)

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
  - DomainReduce produces negative results for K-NN (worst case)
- Optimizations effective for CVC3

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
  - DomainReduce produces negative results for K-NN (worst case)
- Optimizations effective for CVC3
  - Small improvement for K-NN

Study Results 1

(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		**Subsumption**		**Incremental solving**		**DomainReduce with dependencies**		**DomainReduce without dependencies**	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

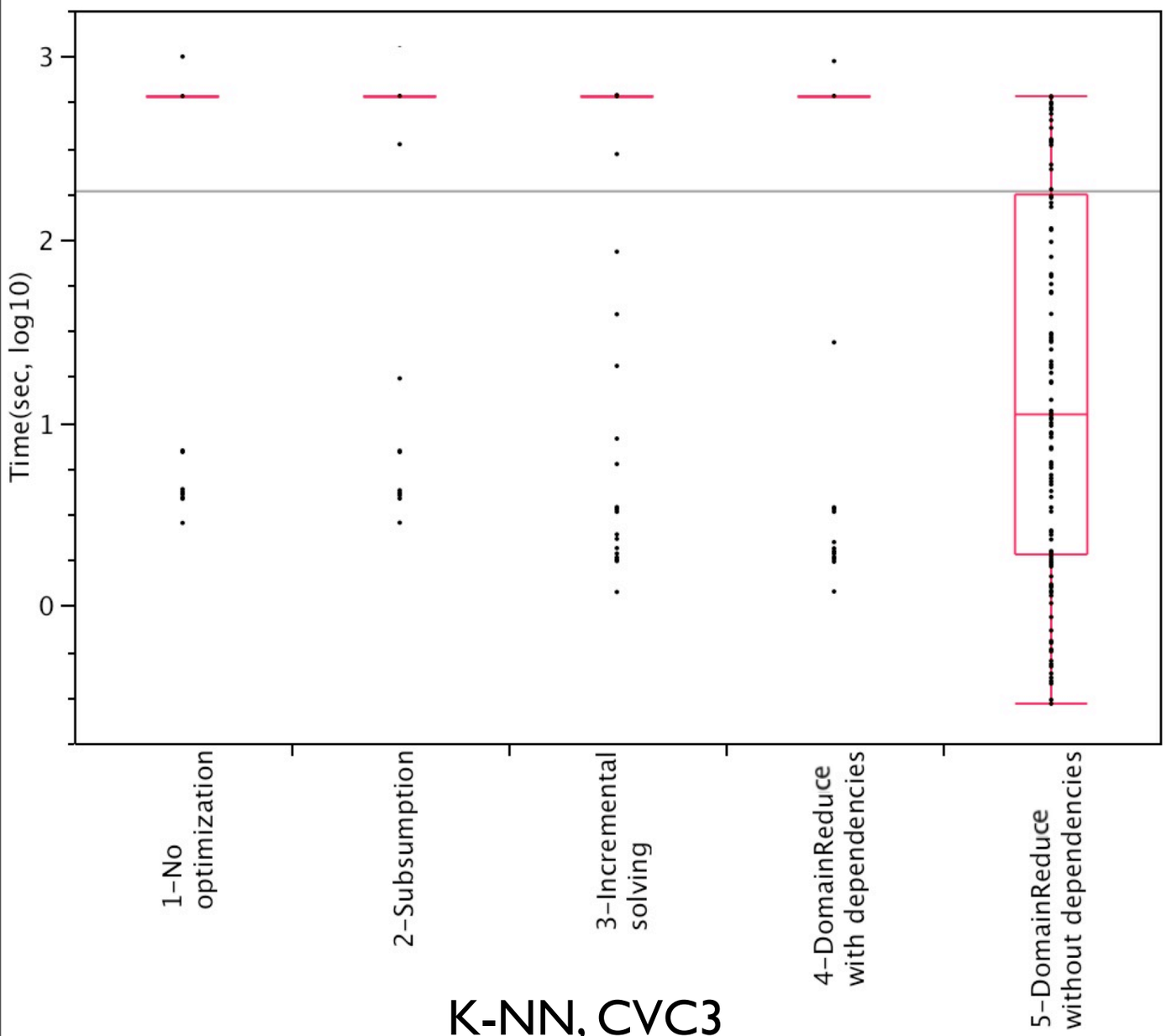
- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
  - DomainReduce produces negative results for K-NN (worst case)
- Optimizations effective for CVC3
  - Small improvement for K-NN
  - Dramatic improvement for XMLParser (25% ➡ 100%)

Study Results 2

(time to process constraints)

Study Results 2

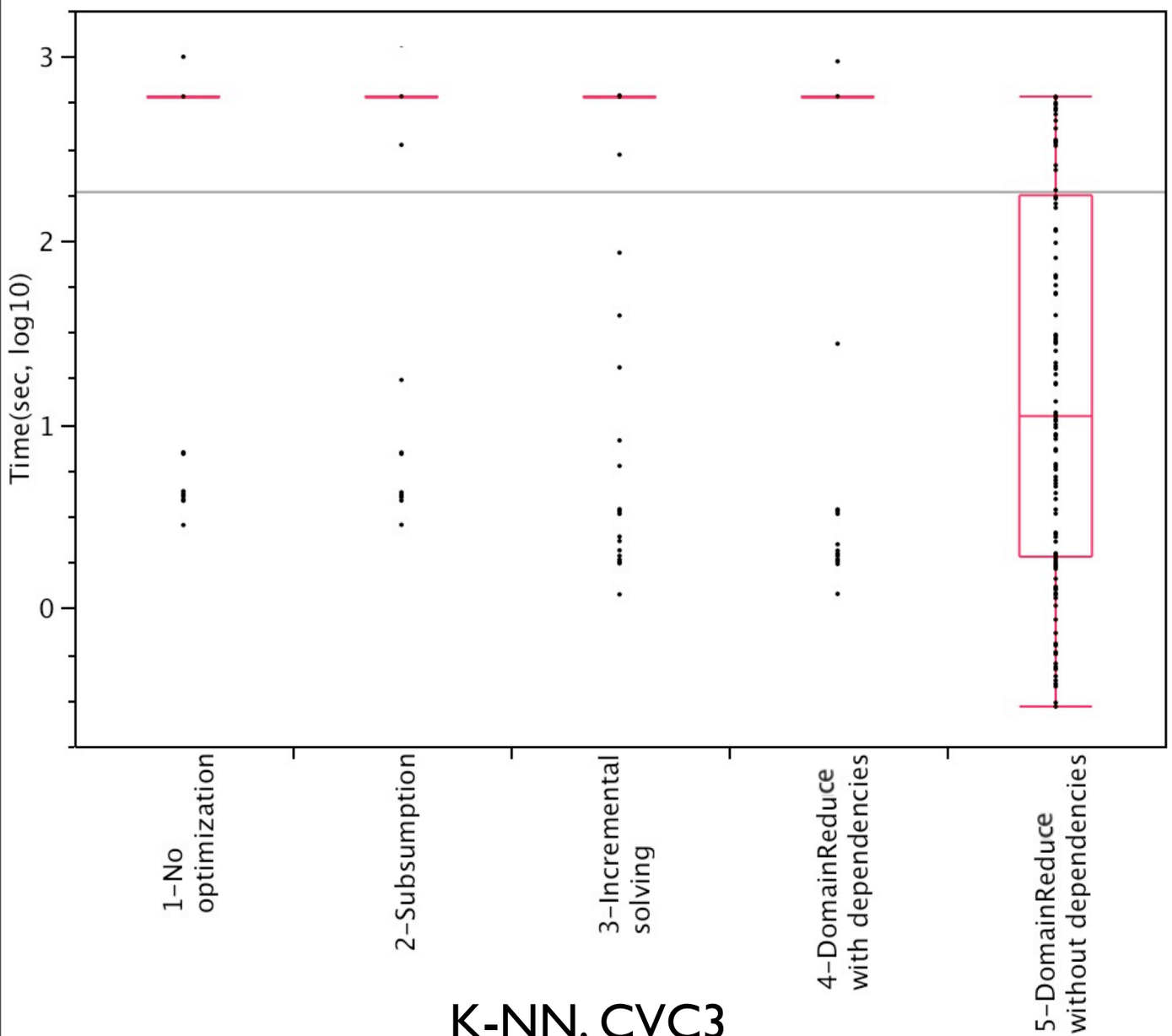
(time to process constraints)



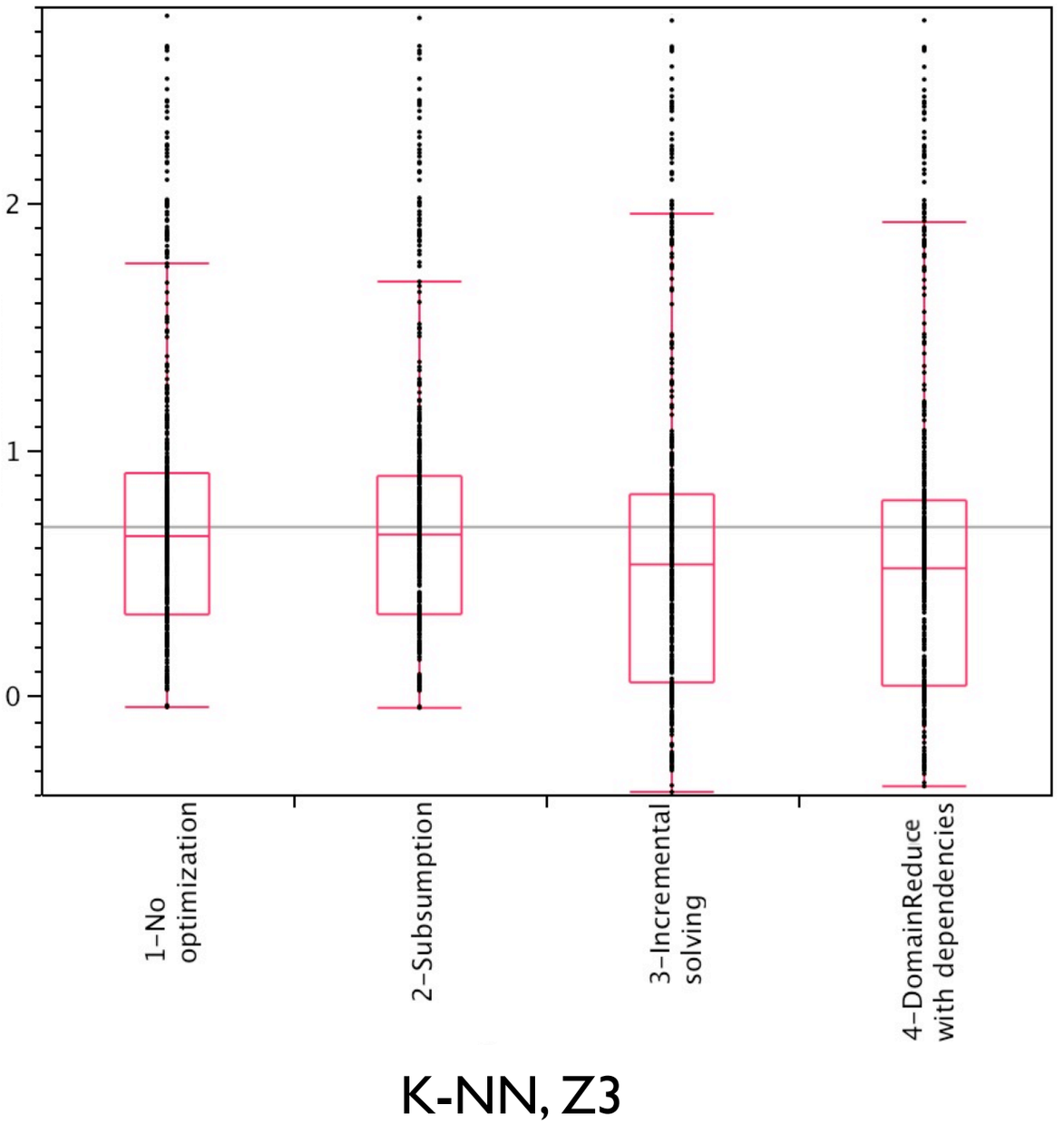
K-NN, CVC3

Study Results 2

(time to process constraints)



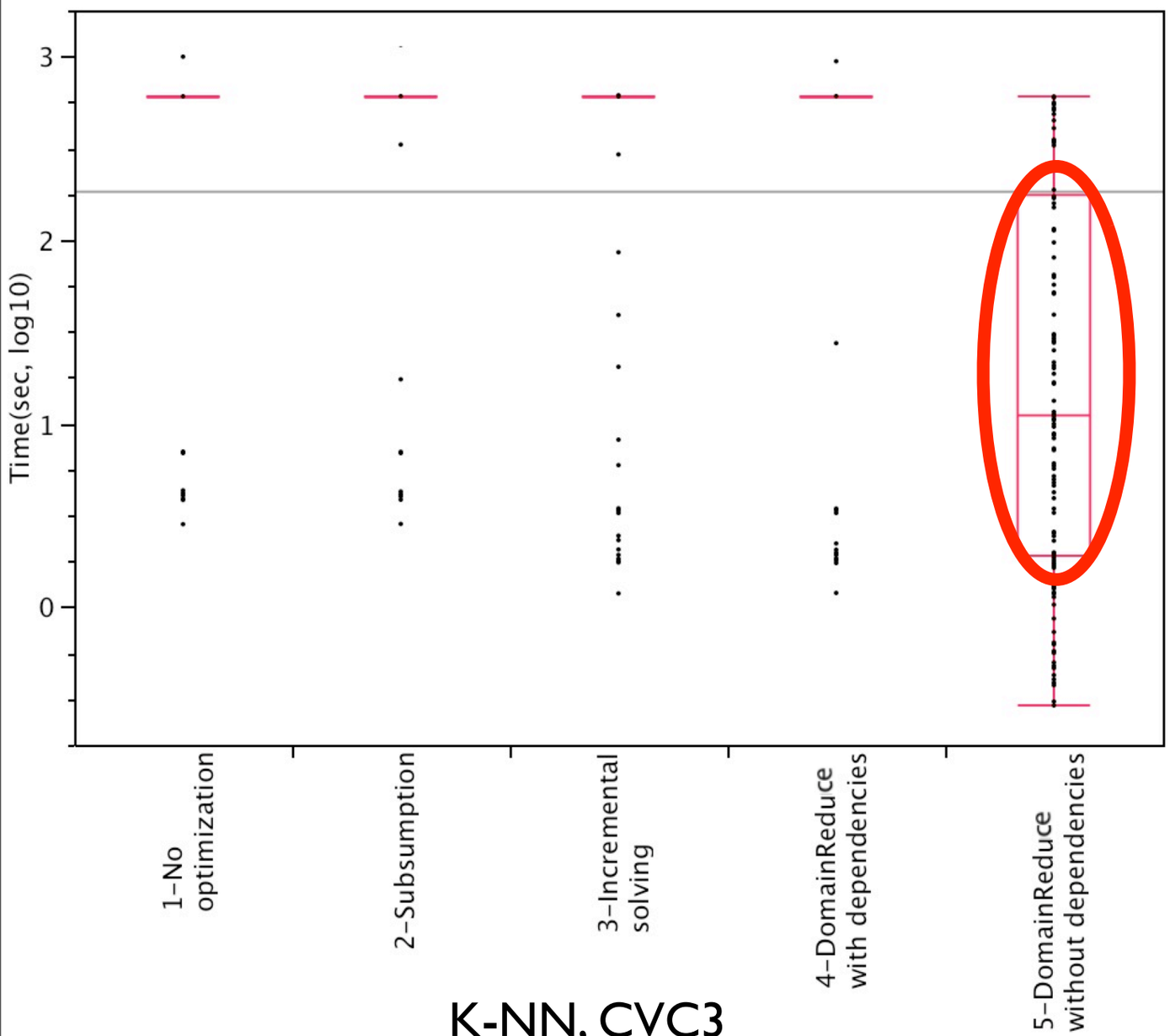
K-NN, CVC3



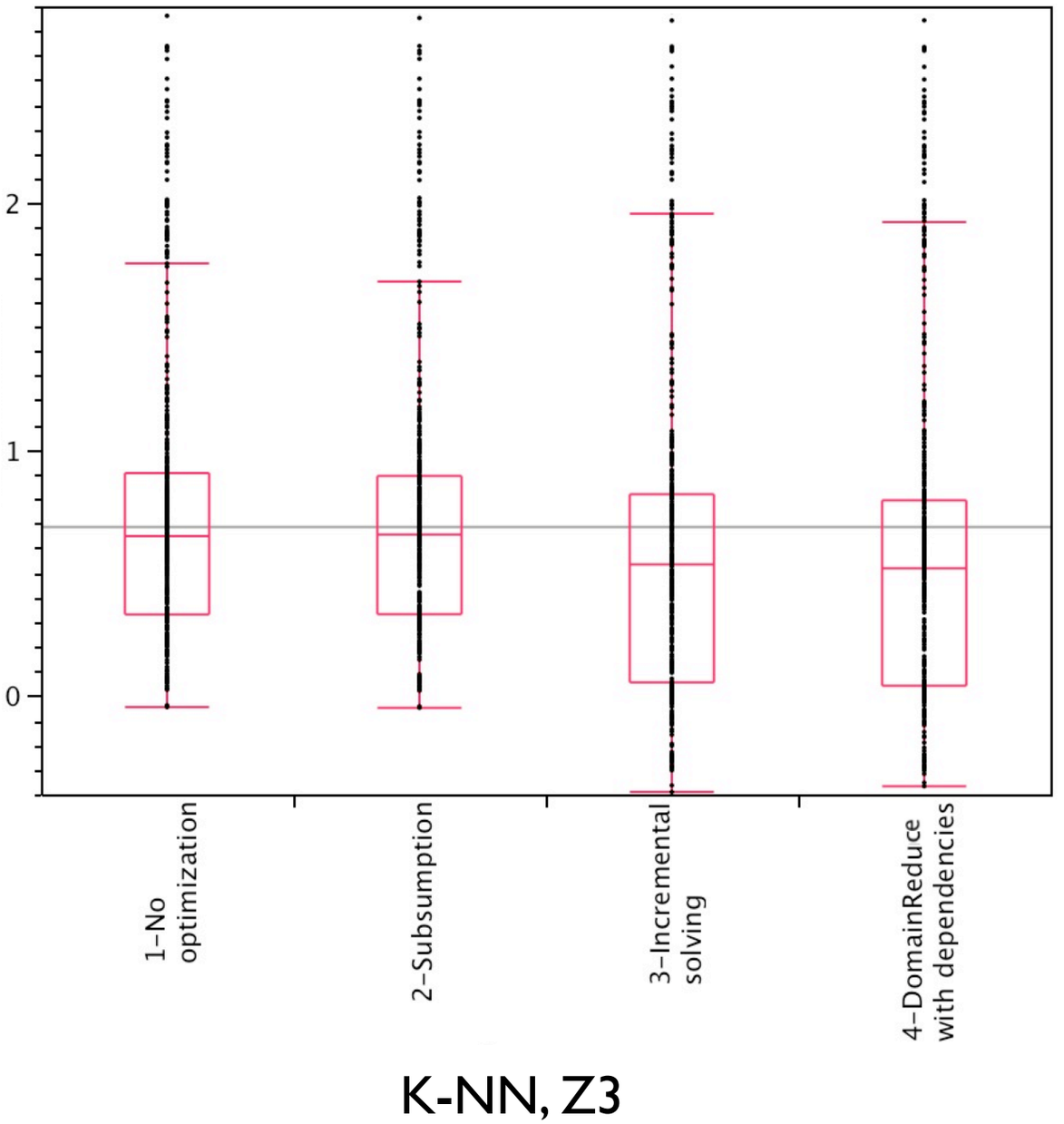
K-NN, Z3

Study Results 2

(time to process constraints)



K-NN, CVC3



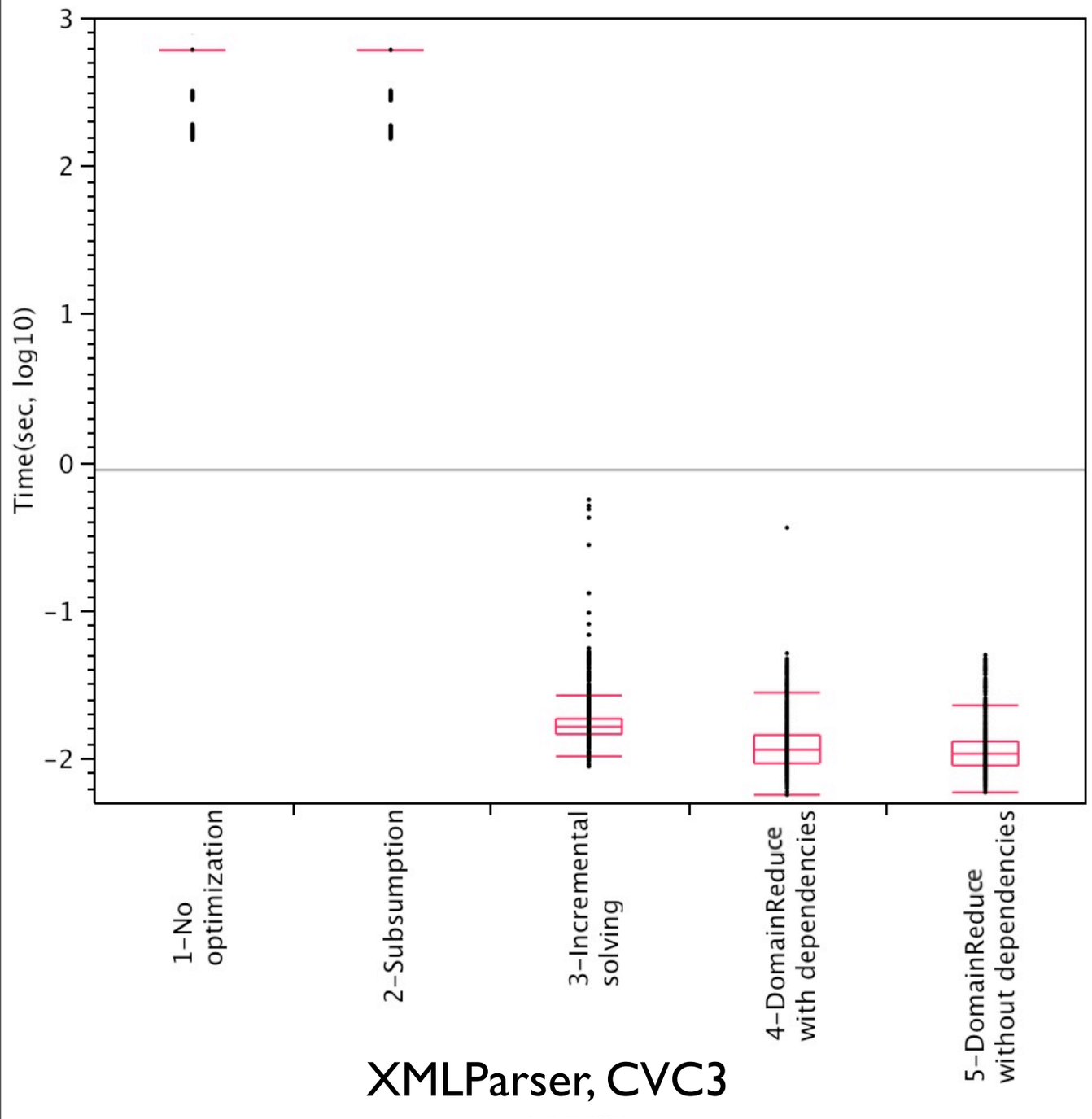
K-NN, Z3

Study Results 2

(time to process constraints)

Study Results 2

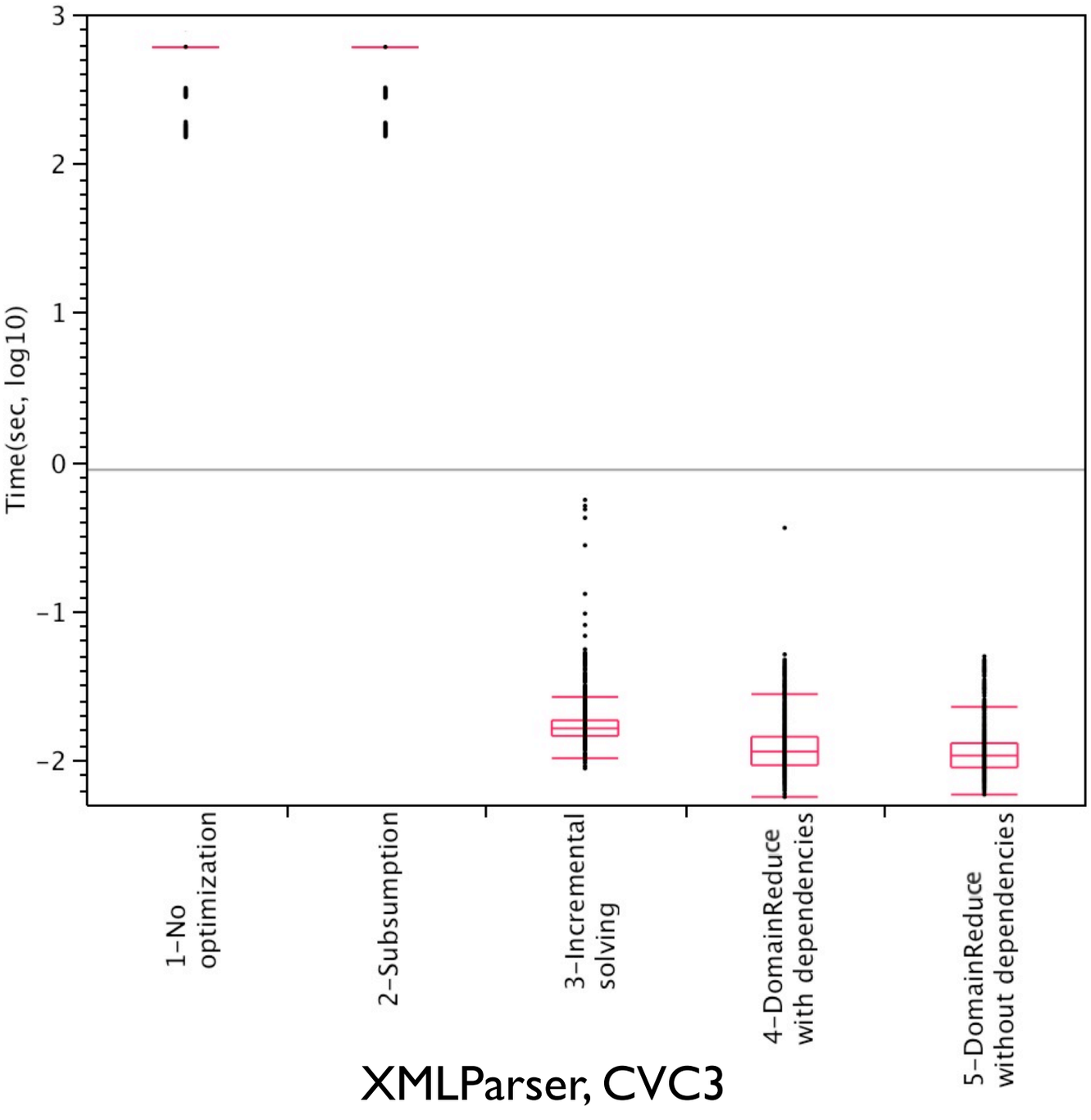
(time to process constraints)



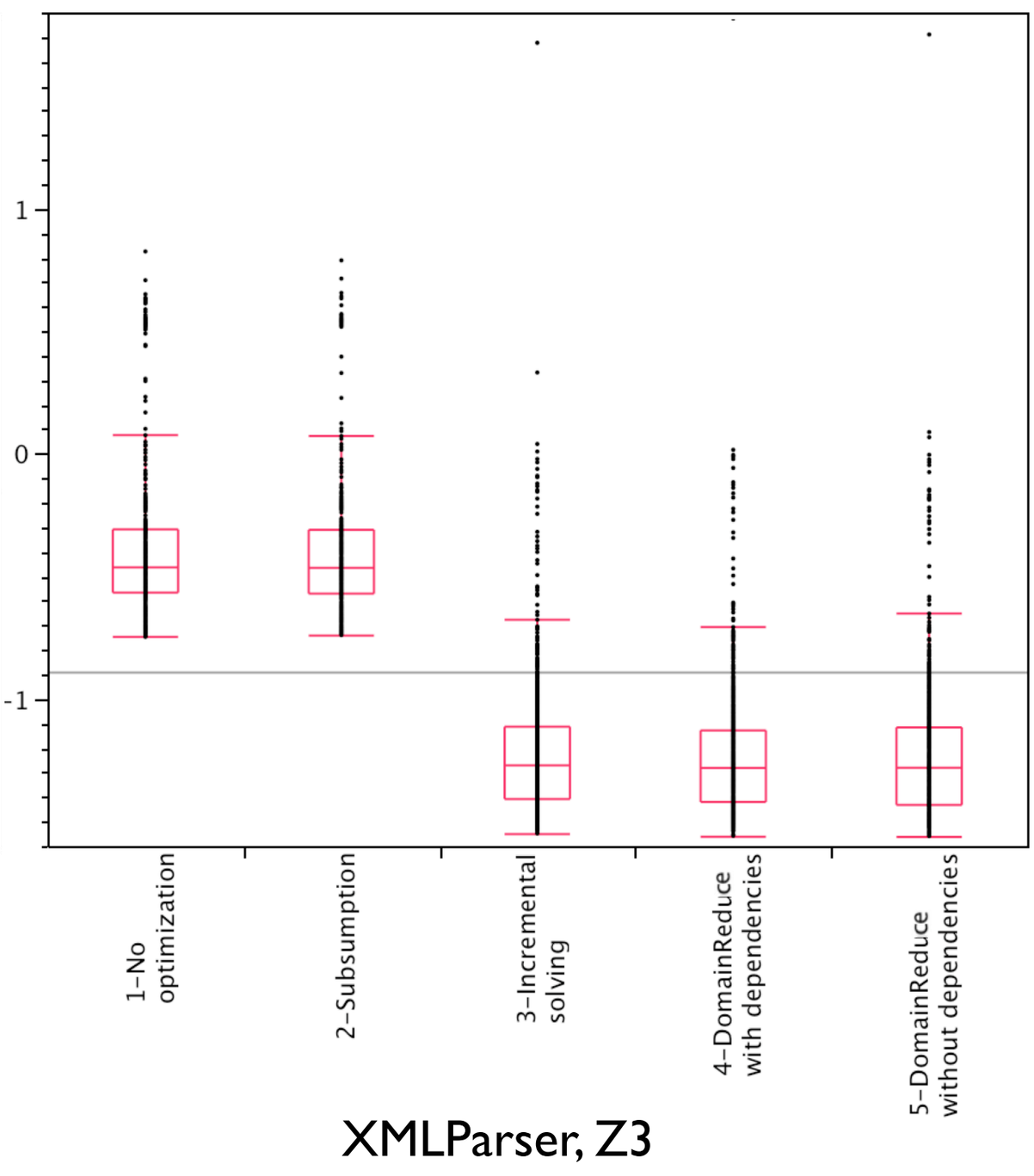
XMLParser, CVC3

Study Results 2

(time to process constraints)



XMLParser, CVC3

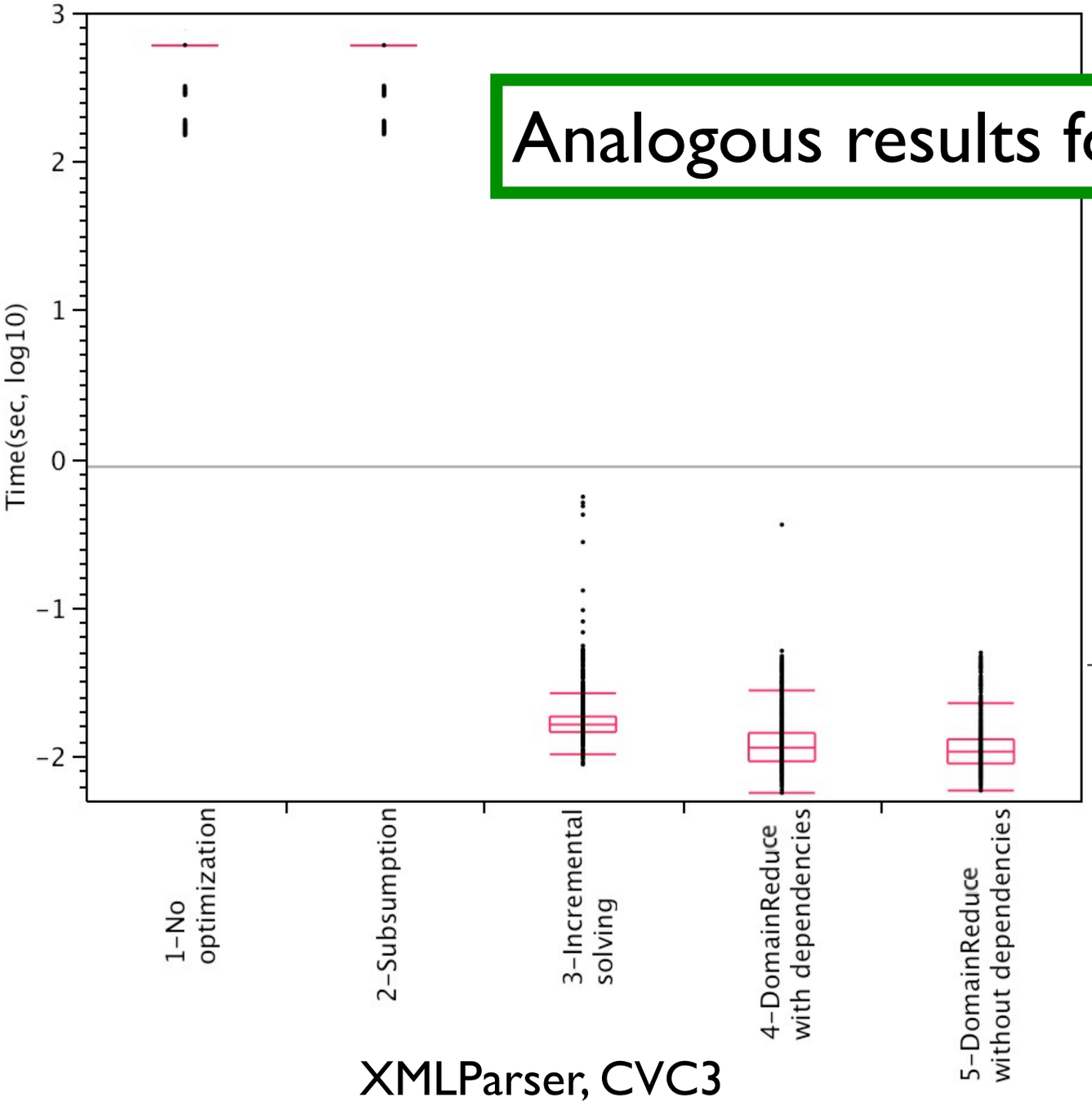


XMLParser, Z3

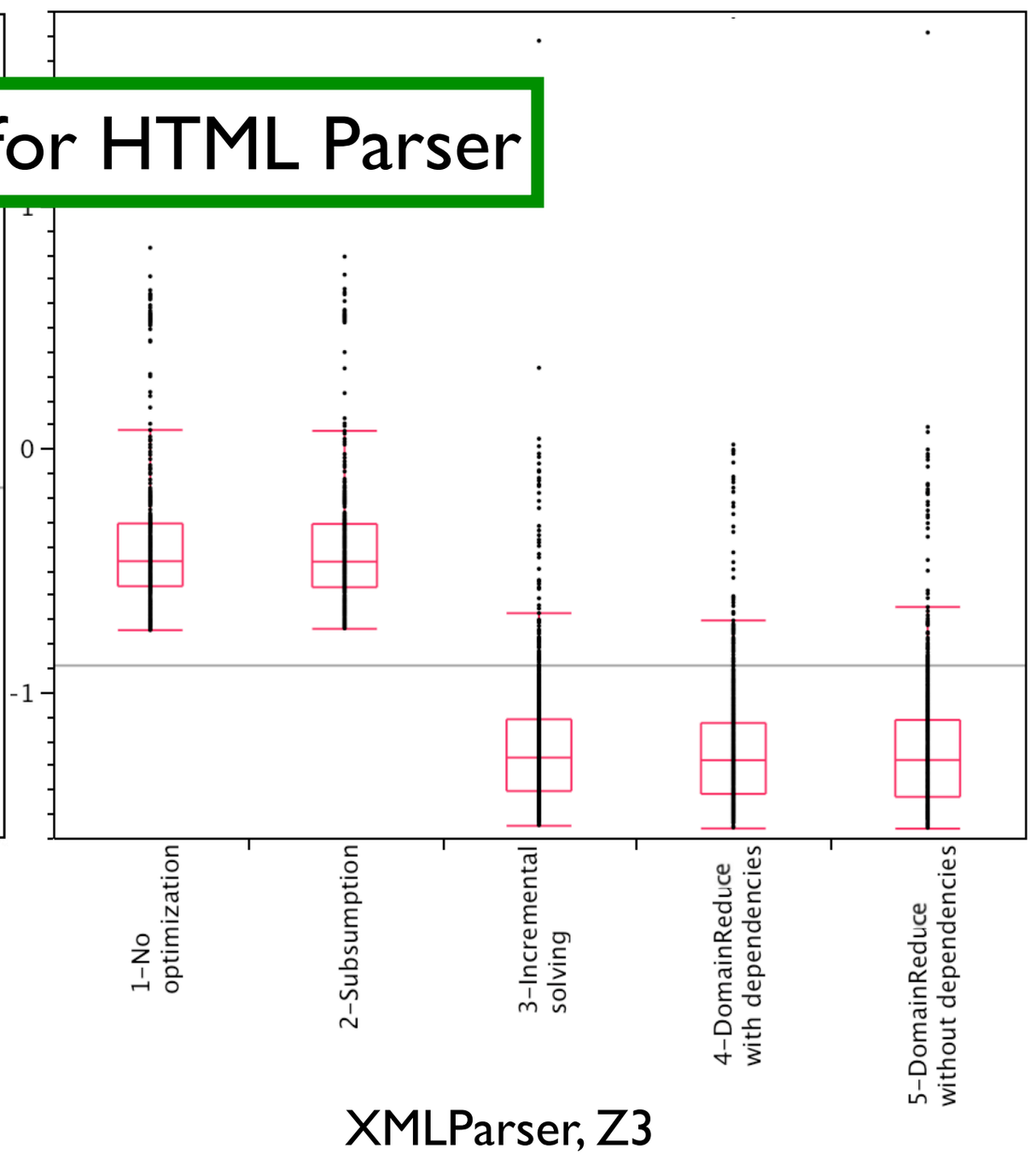
Study Results 2

(time to process constraints)

Analogous results for HTML Parser



XMLParser, CVC3



XMLParser, Z3

Study Results 2

(time to process constraints)

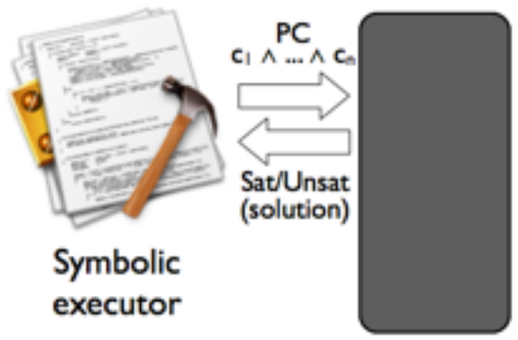
Study Results 2
(time to process constraints)

- K-NN
  - All but one optimizations provided no benefits (timeout or unsat after a long time)
  - DomainReduce with no dependencies finds solutions for less constraints, but very quickly, for K-NN and CVC3

Study Results 2
(time to process constraints)

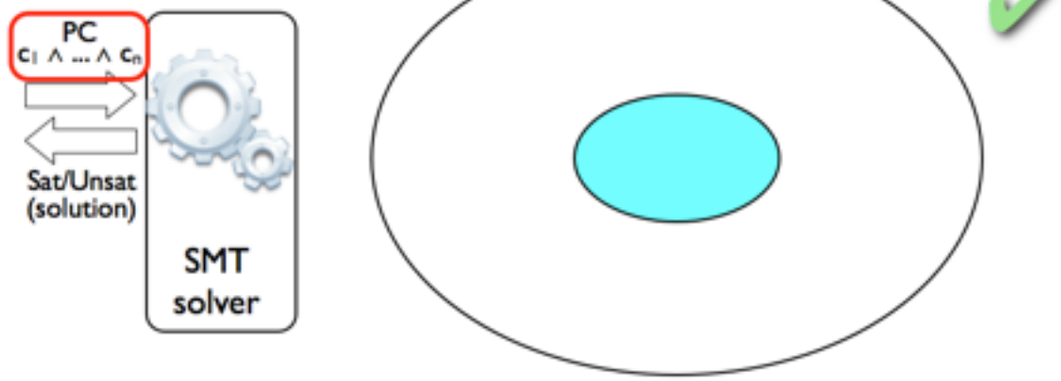
- K-NN
  - All but one optimizations provided no benefits (timeout or unsat after a long time)
    - DomainReduce with no dependencies finds solutions for less constraints, but very quickly, for K-NN and CVC3
- HTMLParser and XMLParser
  - Almost all optimizations improve efficiency of constraint solvers dramatically (several orders of magnitude)

Symbolic Execution and SMT Solving



DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solution of similar PC



Tradeoff speed/likelihood of finding solution

Study Results

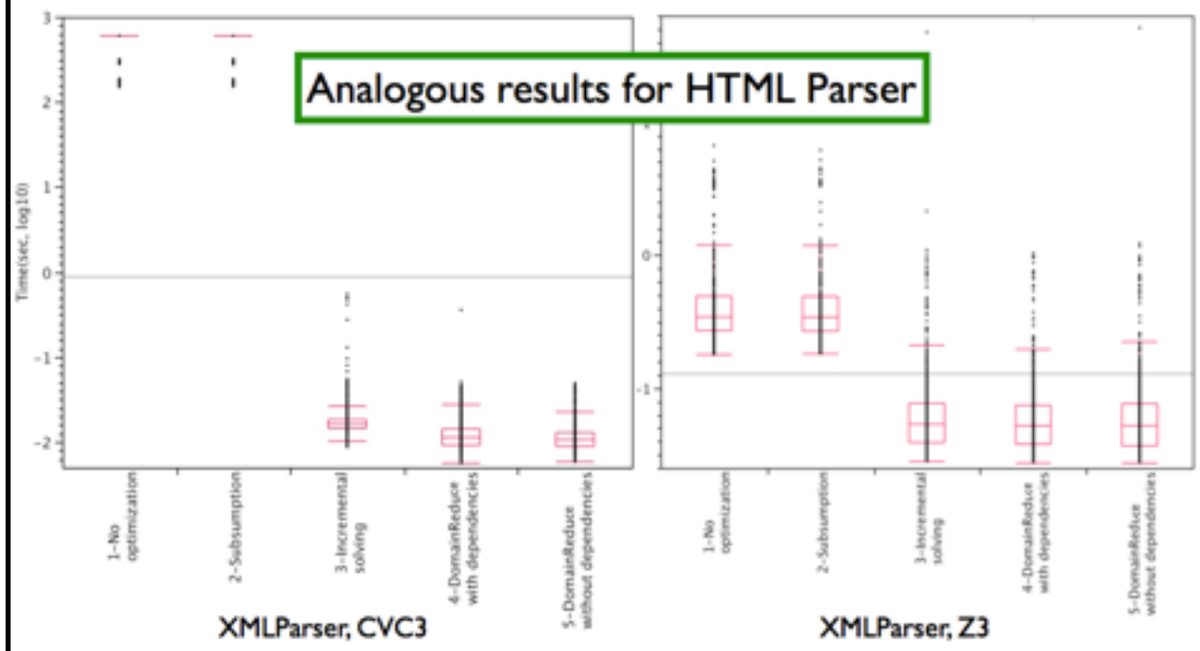
(# constraints processed)

Subjects	PCs considered	PCs successfully processed unsat+sat									
		No optimization		Subsumption		Incremental solving		DomainReduce with dependencies		DomainReduce without dependencies	
		cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3	cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832	**473** 49+424	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0	**261** 261+0	**936** 936+0	**271** 271+0	**937** 937+0	**262** 262+0	**878** 878+0	**111** 0+111	**0** 0+0

- Results for HTMLParser uninteresting
- Optimizations ineffective for Z3,
  - Useless or ineffective with one exception
  - DomainReduce even produce negative results for K-NN (worst case)
- Optimizations effective for CVC3
  - Small improvement for K-NN
  - Dramatic improvement for XMLParser (25% ➡ 100%)

Study Results

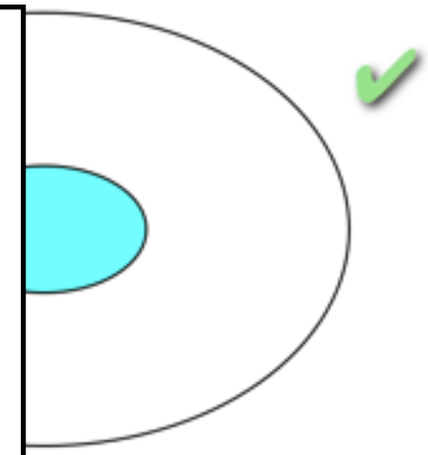
(time to process constraints)



Symbolic Execution and SMT Solving

DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solution of similar PC



ood of finding solution

Future work

- More experiments (subjects, solvers, configurations)
- Investigate why optimizations work/don't work
- Apply optimizations in parallel
- More sophisticated optimizations (program structure or properties)
- Tighter integration

St

(# co

Subjects	PCs considered	No optimization	
		cvc3	z3
HTMLParser	1879	**1879** 43+1836	**1879** 43+1836
XMLParser	1881	**473** 49+424	**1881** 49+1832
K-NN	1930	**261** 261+0	**936** 936+0

- Results for HTM
- Optimizations ineffective for Z3;
  - Useless or ineffective with one exception
  - DomainReduce even produce negative results for K-NN (worst case)
- Optimizations effective for CVC3
  - Small improvement for K-NN
  - Dramatic improvement for XMLParser (25% ➡ 100%)

lts

straints)

L Parser