UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Supporting Software Product Line Testing by Optimizing Code Configuration Coverage

László Vidács[1], Ferenc Horváth[2], József Mihalicza[3], Béla Vancsics[2] and Árpád Beszédes[2]

[1]MTA-SZTE Research Group
on Artificial Intelligence, Hungary

[2]Department of Software Engineering,
University of Szeged, Hungary

[3]NNG LLC,
Budapest, Hungary

Context

NNG ships navigation solutions on a broad spectrum:

- Automotive line fit solutions for tier 1 clients
  Over 30 car brands carry iGO navigation
  (Qnx, Android, Linux, WinCE)
- White label core product
  After-market head units, mobile apps
  (iOS, WinCE, WinMobile, Android)
- Mobile navigation app for B2C end users

Product Line

NNG philosophy:

„Navigation for All”

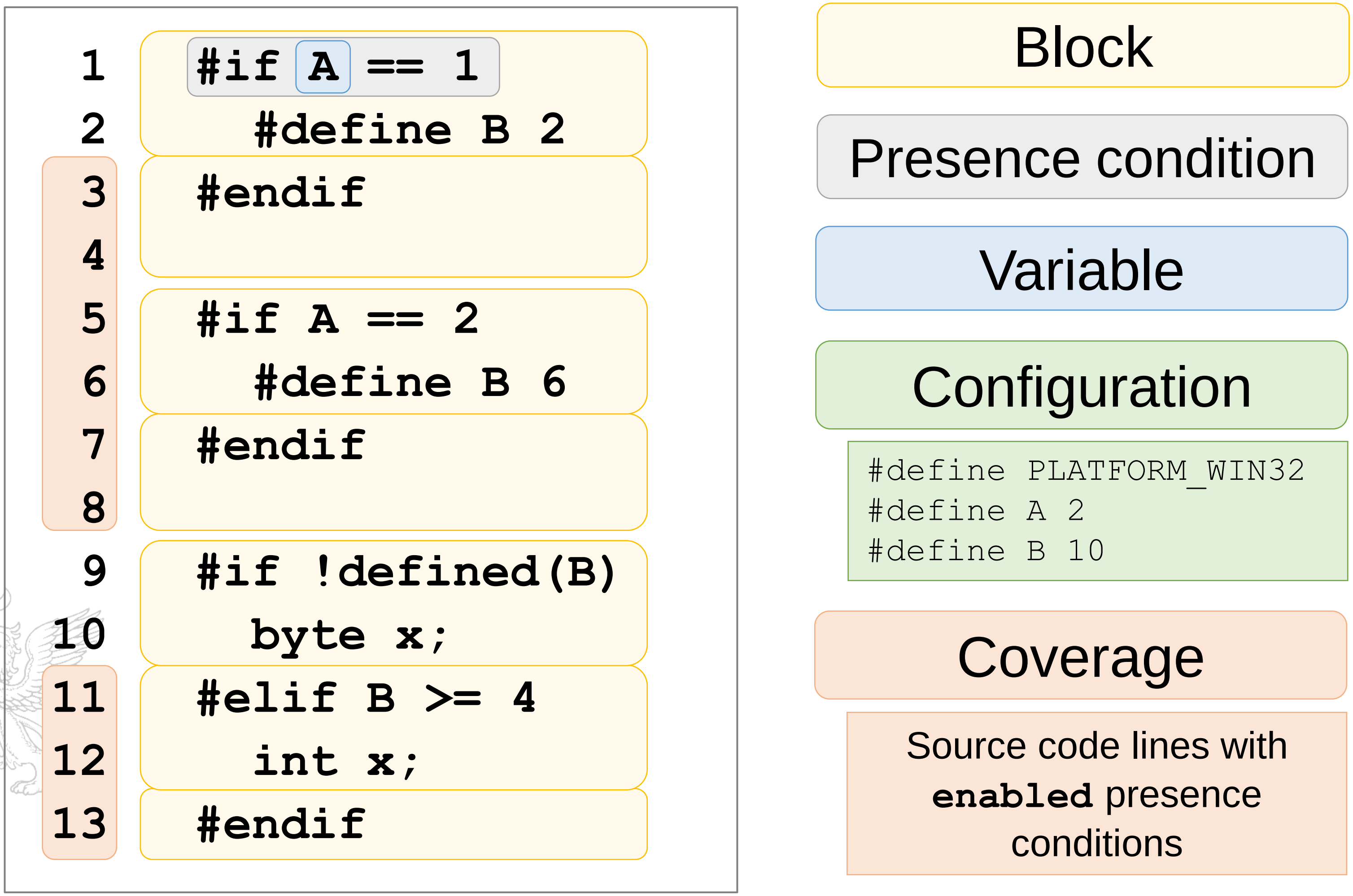
- Achieved by a single code base for core functionalities
- Customizations should integrate well with core features
- SPL: code variability at preprocessor level
  - Platforms (and variants), compilers, rendering engines, 32bit/64bit
  - Windows CE/Mobile/PC, QNX, Linux, Android, iOS
  - Features, customizations

Research goal

- Testing release configurations is not sufficient
  - Get a feature from Config A and turn it on in Config B
- Efficient testing of the configurable core code
- Research goal:

Select small number of configurations which cover large amount of code

Preprocessor based Variability



```
#if A == 1
   #define B 2
#endif

#if A == 2
   #define B 6
#endif

#if !defined(B)
   byte x;
#elif B >= 4
   int x;
#endif
```

Block

Presence condition

Variable

Configuration

```
#define PLATFORM_WIN32
#define A 2
#define B 10
```

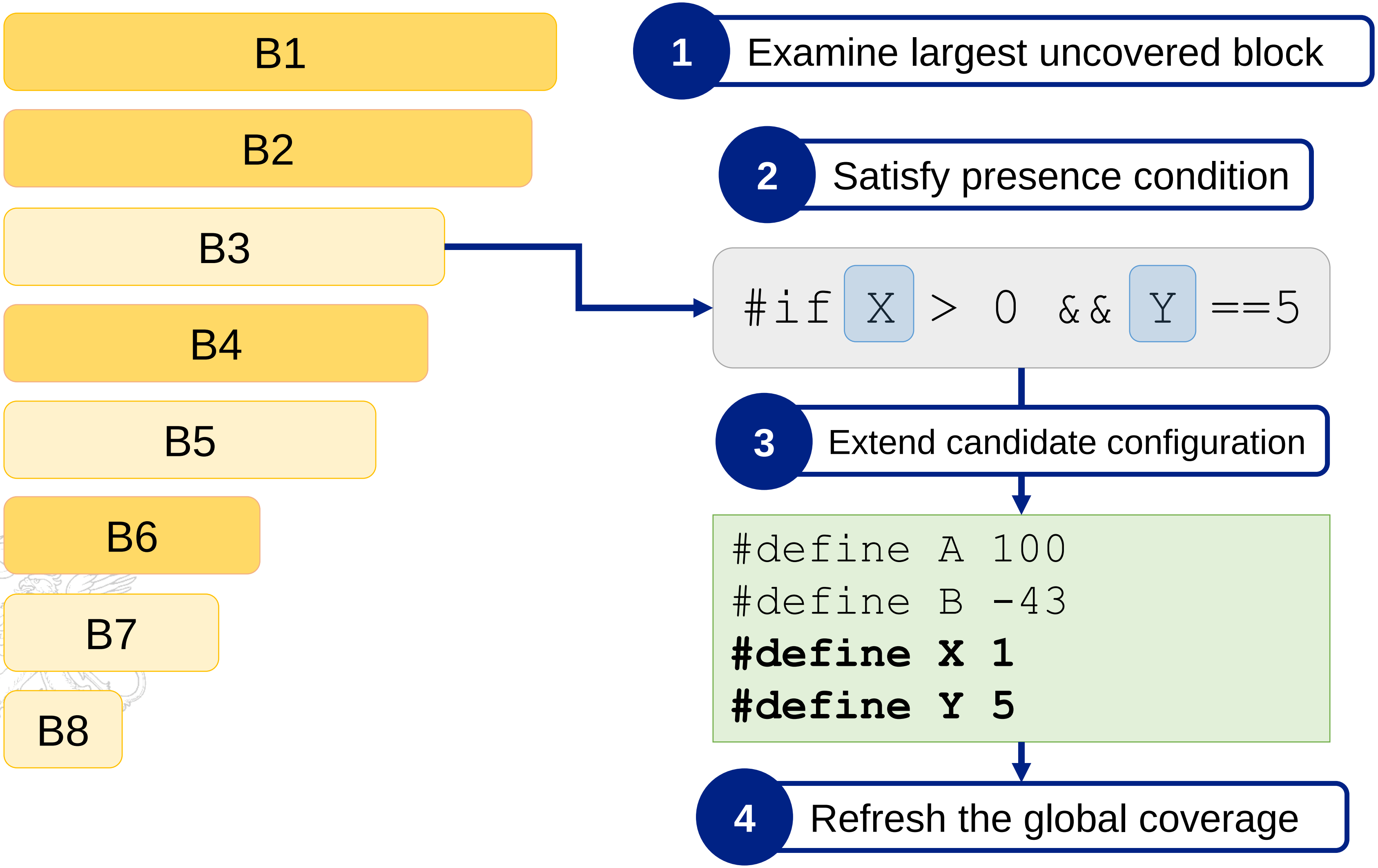
Coverage

Source code lines with **enabled** presence conditions

Search algorithms

- Find N(<10) configurations with highest possible coverage
- Approach
  - Build each configuration incrementally (greedy approach)
  - Create new configurations until N is reached
- **Block-based** approach
  - Try to cover the largest uncovered block
- **Variable-based** approach
  - Select the variable which results the highest overall coverage increase

Block-based algorithm



B1

B2

B3

B4

B5

B6

B7

B8

1 Examine largest uncovered block

2 Satisfy presence condition

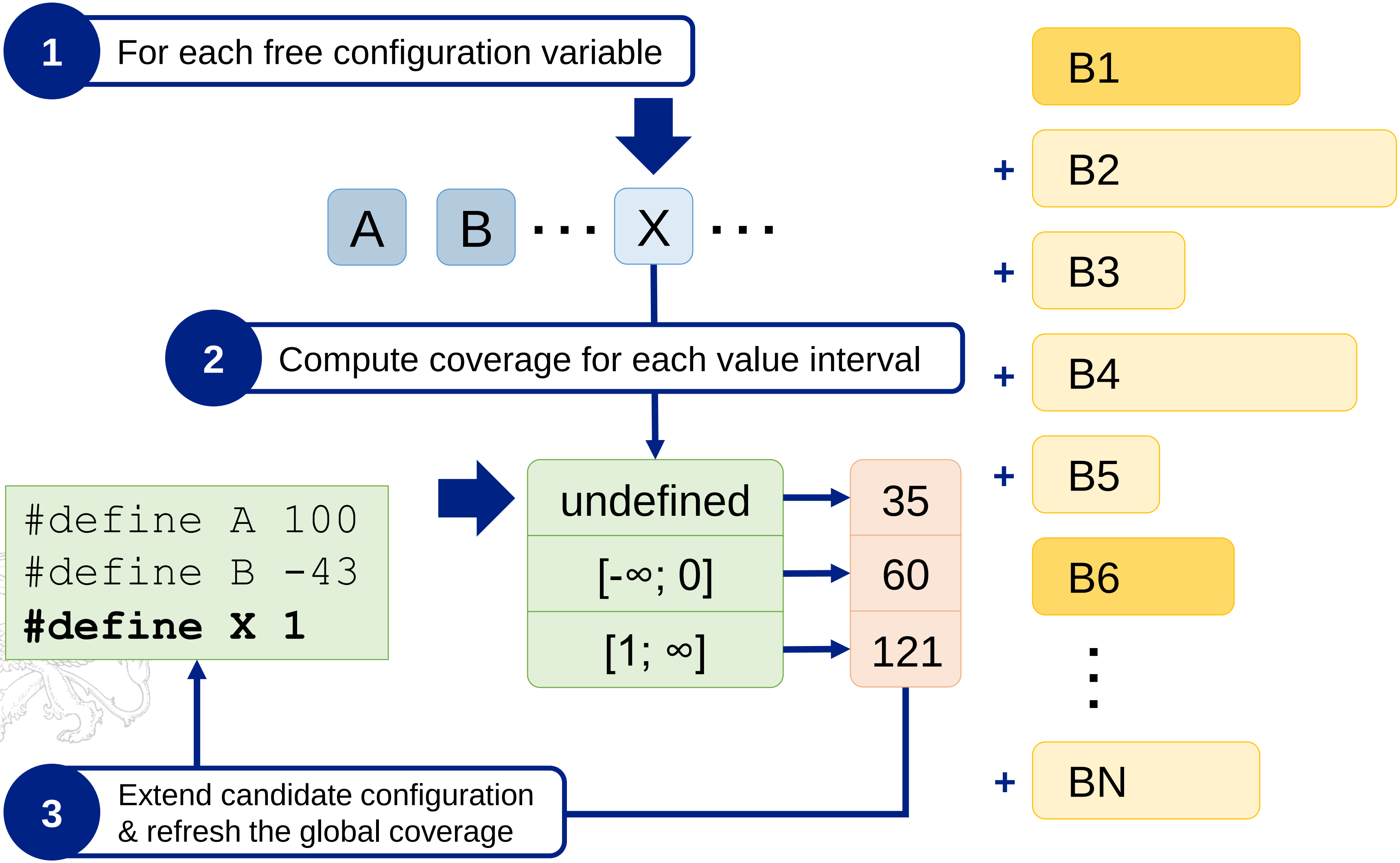
```
#if X > 0 && Y ==5
```

3 Extend candidate configuration

```
#define A 100
#define B -43
#define X 1
#define Y 5
```

4 Refresh the global coverage

Variable-based algorithm



1 For each free configuration variable

A B … X …

2 Compute coverage for each value interval

```
#define A 100
#define B -43
#define X 1
```

undefined	35
[-∞; 0]	60
[1; ∞]	121

3 Extend candidate configuration & refresh the global coverage

B1
+ B2
+ B3
+ B4
+ B5
B6
⋮
+ BN

iGO Navigation Measurements

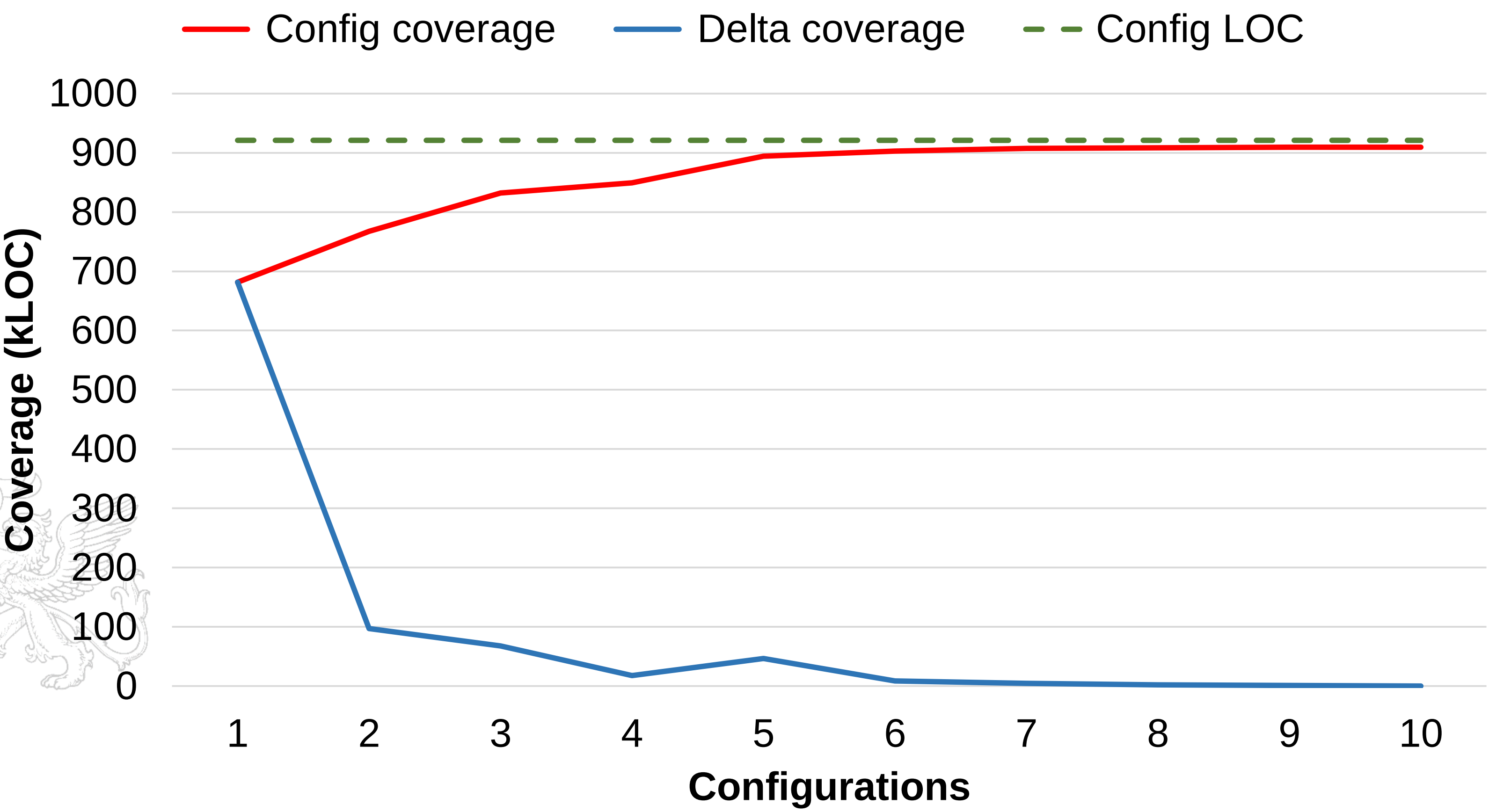
Condition type	Blocks	LOC
Filtered (T, F, #error)	11,847 (25%)	682,300 (35%)
Configuration	**22,067 (47%)**	**920,926 (48%)**
Mixed	10,085 (22%)	271,710 (14%)
Non-configuration	2,811 (6%)	50,064 (3%)
Total	**46,810**	**1,925,000**

- `#error` directives
  - Prevent invalid configurations
- Non-configuration variables -> mixed conditions
  - `MODULE1_DETAILED_DIAG, PERSONAL_PATCHES_JOE`

Results: block-based, N = 10

Config coverage
98.74%

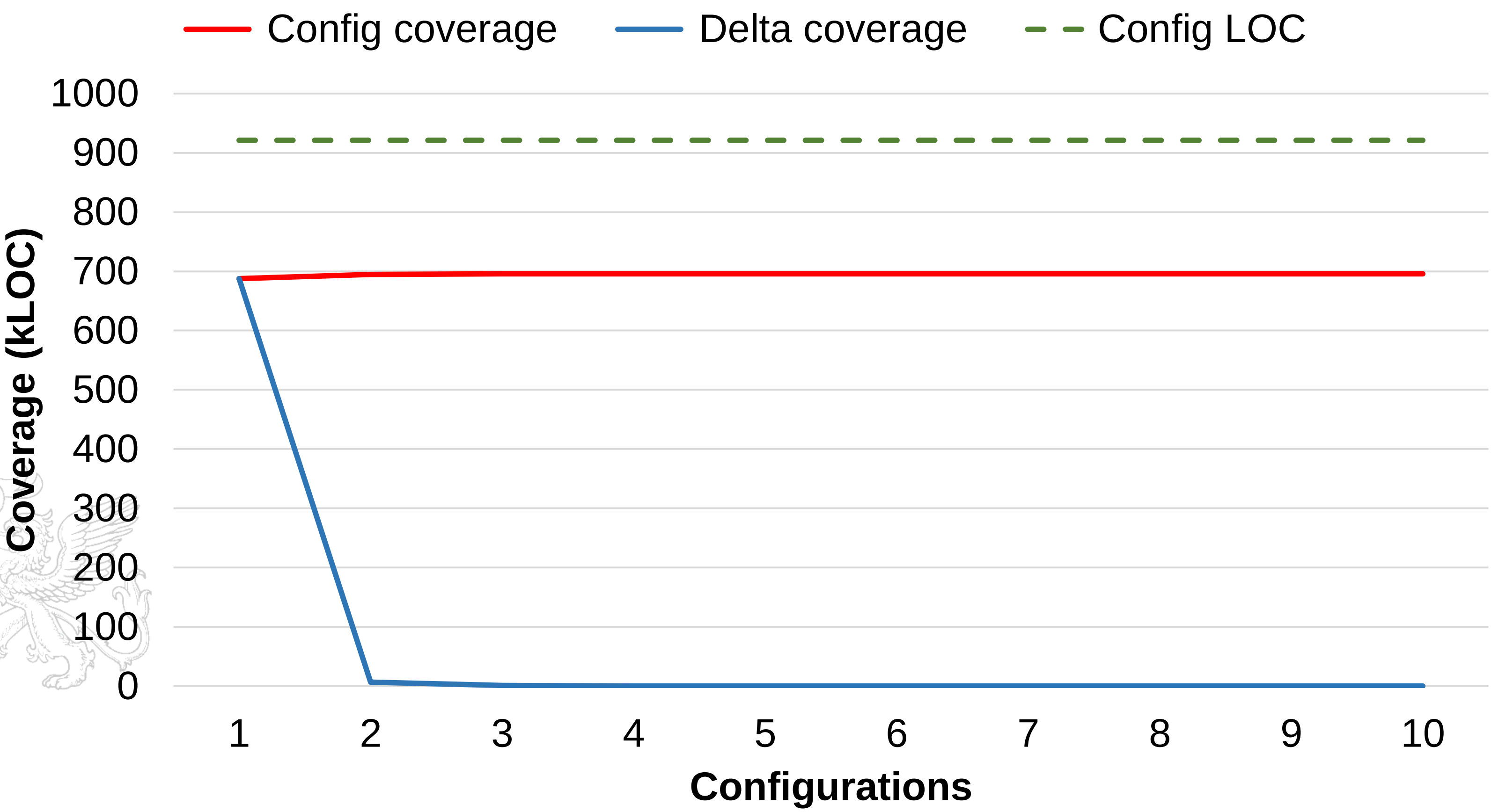
Search time
22 min



Results: variable-based, N = 10

Config coverage
75.56%

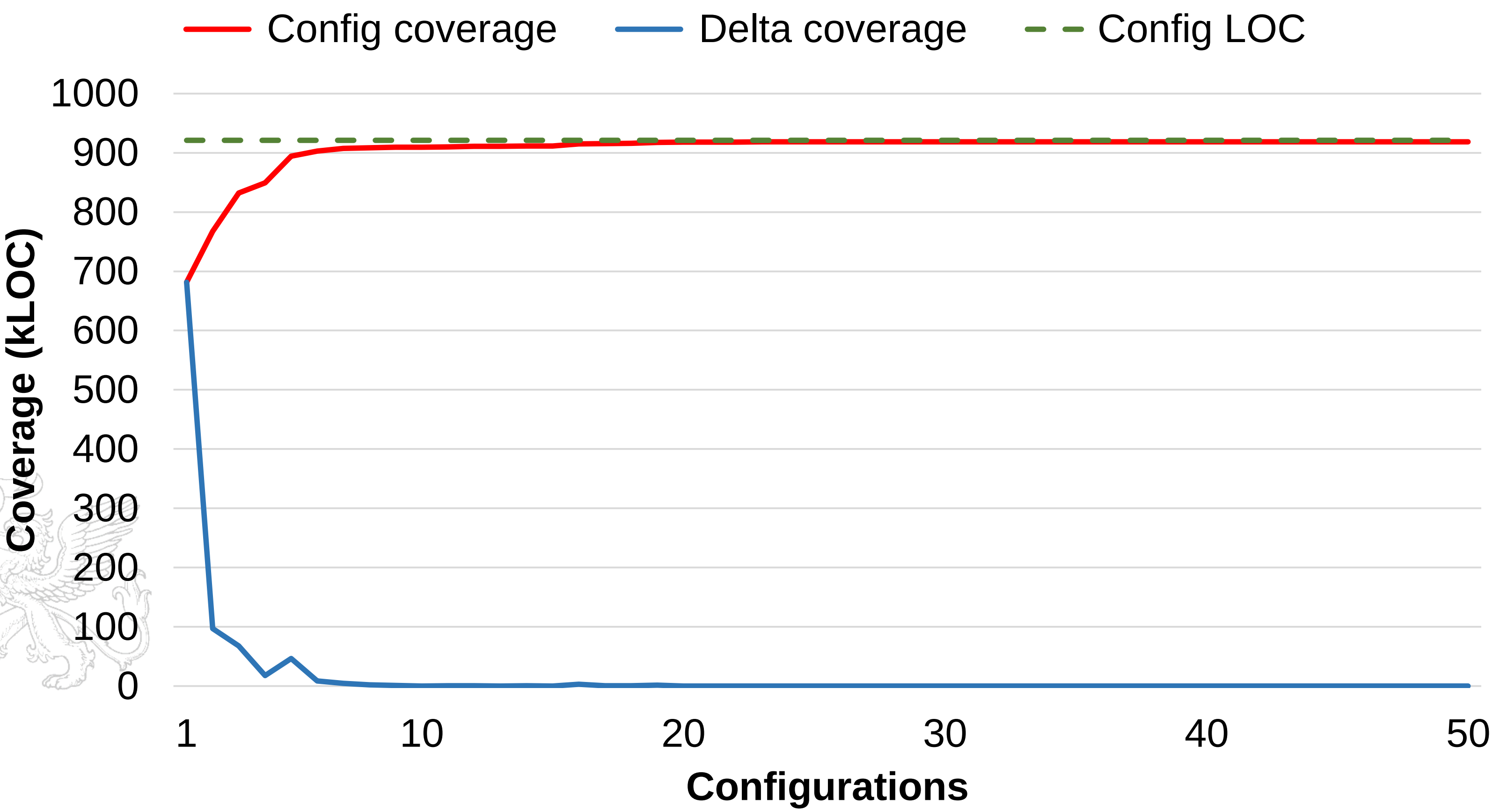
Search time
27 min



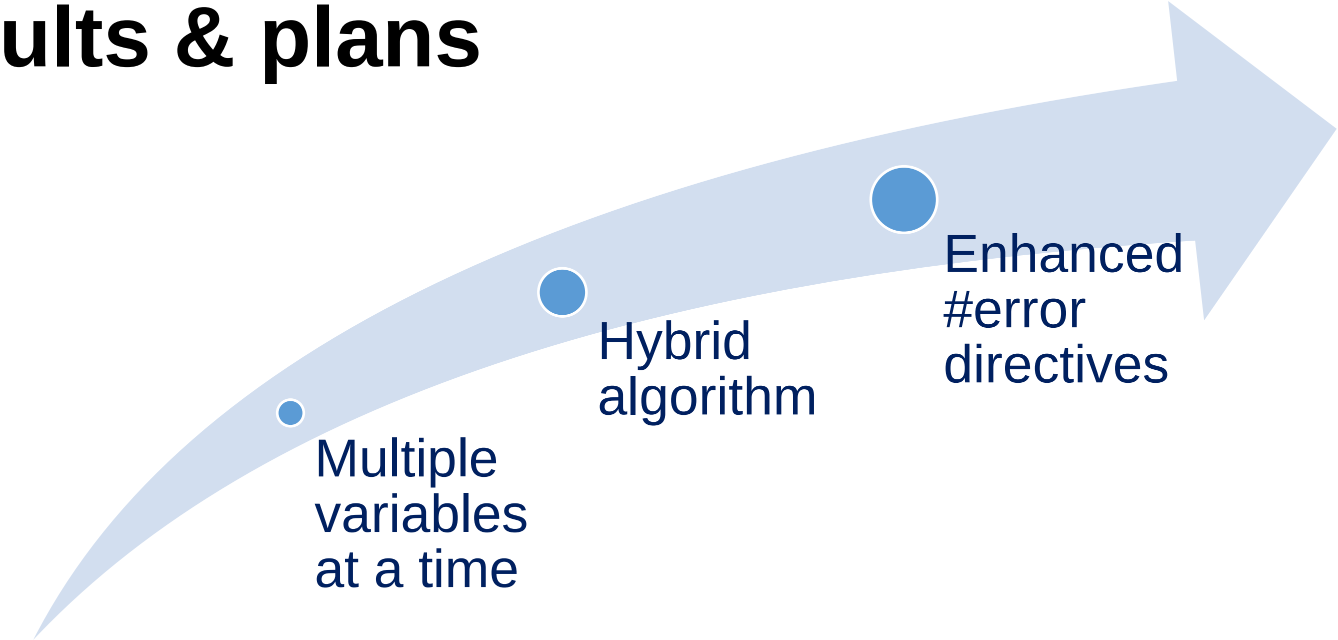
Results: block-based, N = 50

Config coverage
99.74%

Search time
2 h



Results & plans



Multiple variables at a time

Hybrid algorithm

Enhanced #error directives

BLOCK-BASED

98.74%

N=10, 22 min

VARIABLE-BASED

75.56%

N=10, 27 min