Search-Based Ambiguity Detection in Context-Free Grammars

N K Vasudevan and L Tratt
Informatics
King’s College London

Literature

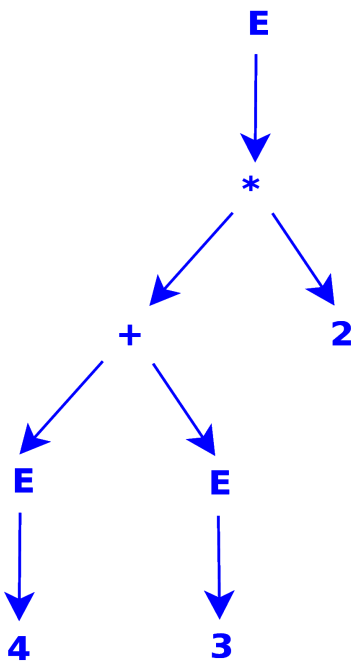
- What is ambiguity?
  - Grammar - parsed in multiple ways
- Ambiguity - undecidable problem
- Multiple parse trees
  - Undesirable for programming languages

E -> E '+' E | E '*' E

---

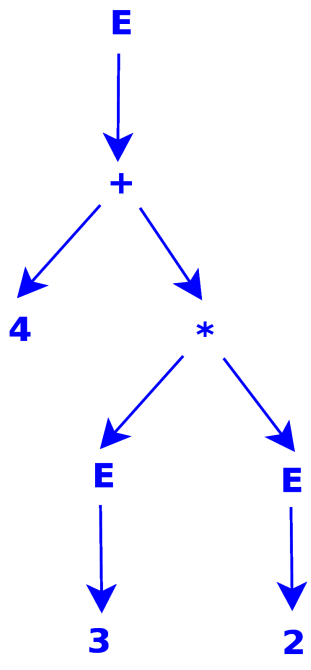
4 + 3 * 2

Parse Tree 1



Value = 14

Parse Tree 2

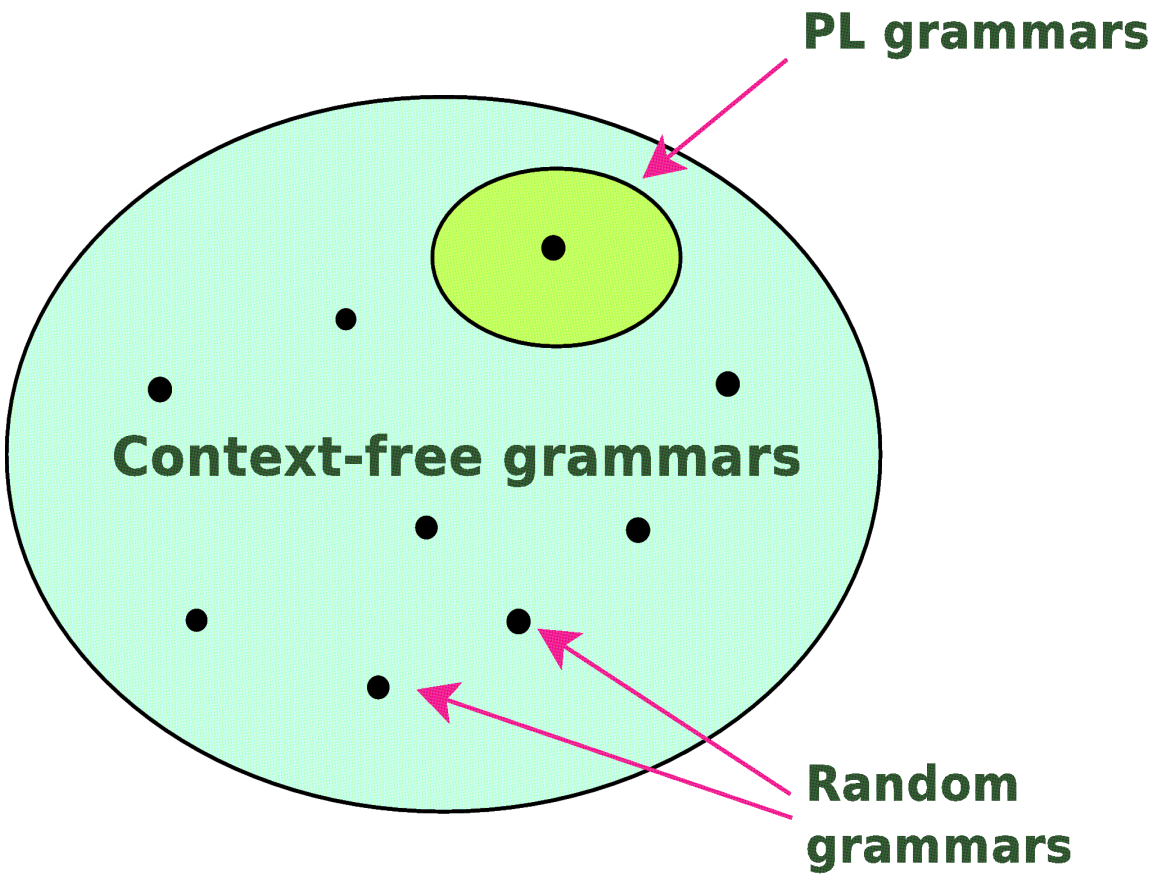


Value = 10

Ambiguity Detection

- Exhaustive search
  - Amber, CFGA
- Approximation
  - ACLA, NUT
- Hybrid= Approximation + Exhaustive
  - AmbiDexter = (grammar filtering + Exhaustive)

CFGs



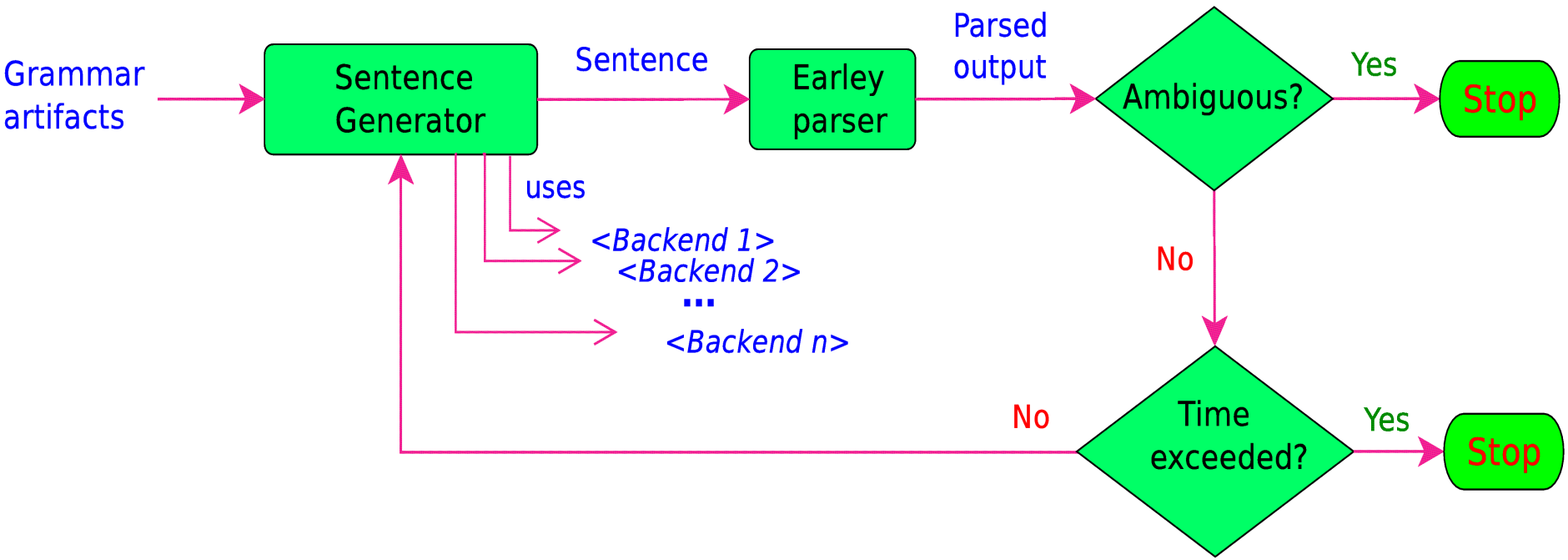
- Programming Language (PL) grammars - **tiny subset**
- Random grammars – span a **wider subset** of CFGs

Search-Based

- Aim to find ***good enough*** solutions
- Applied to wide range of problems
- Two types:
- Purely random
  - Simple technique
  - Scan search space randomly
- Metaheuristic
  - Fitness function
  - Guided search
  - Hill Climbing

Framework

- Search-Based Ambiguity Detection (SinBAD)



Definitions

- CFG is a tuple: $G = \langle N,T,P,S \rangle$
  - N – set of nonterminals
  - T – set of terminals
  - P – set of production rules
  - S – start symbol
- Example grammar
  - S : A ⇐ A production rule
  - A : 'a' | 'b' ⇐ A rule with two alternatives

Dynamic1

Generate(G, rule, D, T, Sentence):

- Exit if (t > T)
- Keep count of rule.entered and rule.exited
- if (d > D)

    alt ← Favour-alternative(rule)
- else

    alt ← Pick an alternative randomly
- for nonterminals in alt → Generate(...)
- for terminals in alt → append to Sentence
- **return** Sentence

Dynamic1

```
Favour-alternative(rule)
    scores ← []
    for each alt in rule
        for each nt in alt
            if rule[nt].entered > 0
                score += (1 – (rule[nt].exited/rule[nt].entered))
        scores ← score
    min-alts ← alt in rule with score = min(scores)
    return random(min-alts)
```

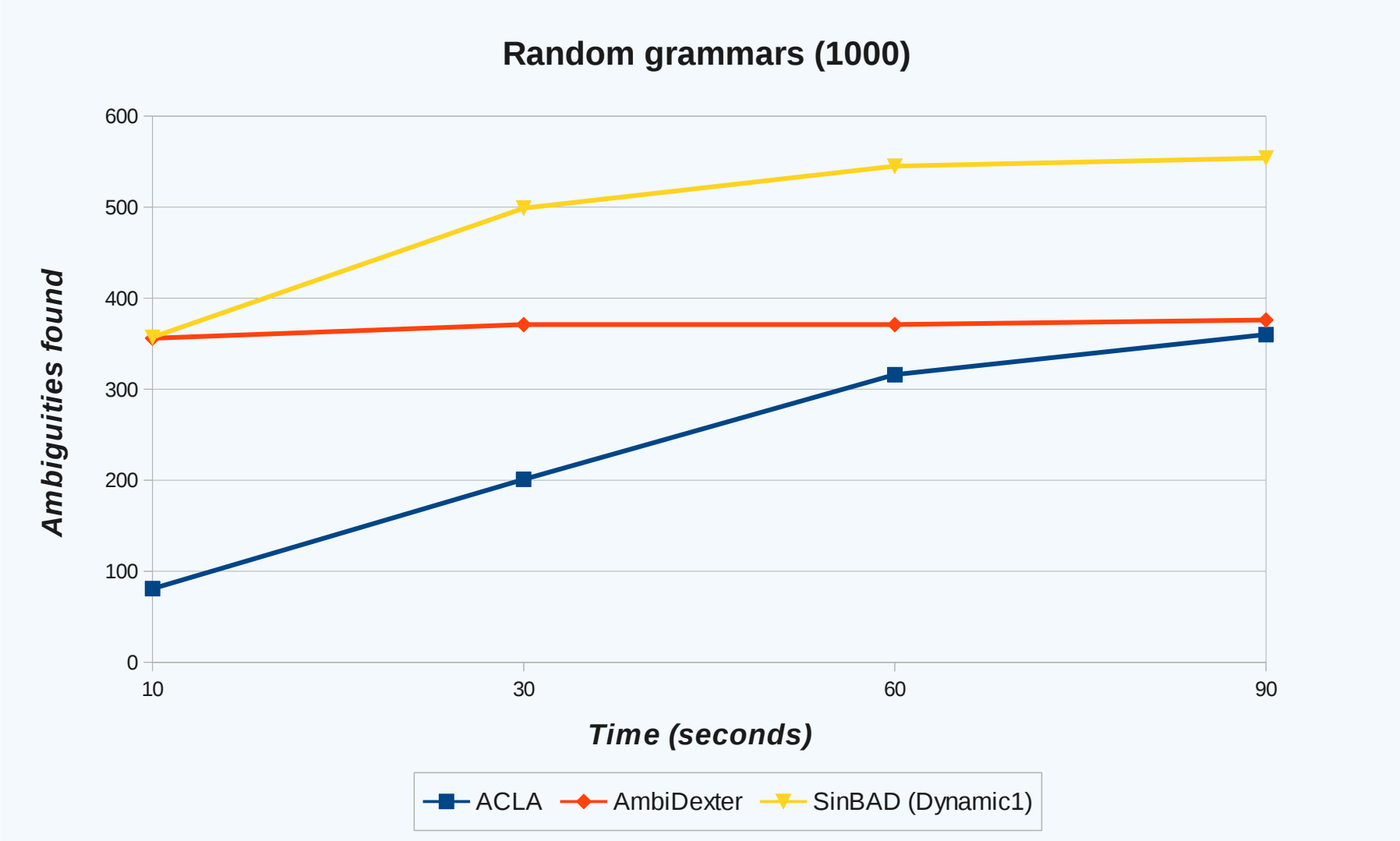
Experiment

- To study how search-based approach uncovers ambiguity
- Evaluate three tools
  - ACLA – Approximation
  - AmbiDexter – Hybrid
  - SinBAD – Random search

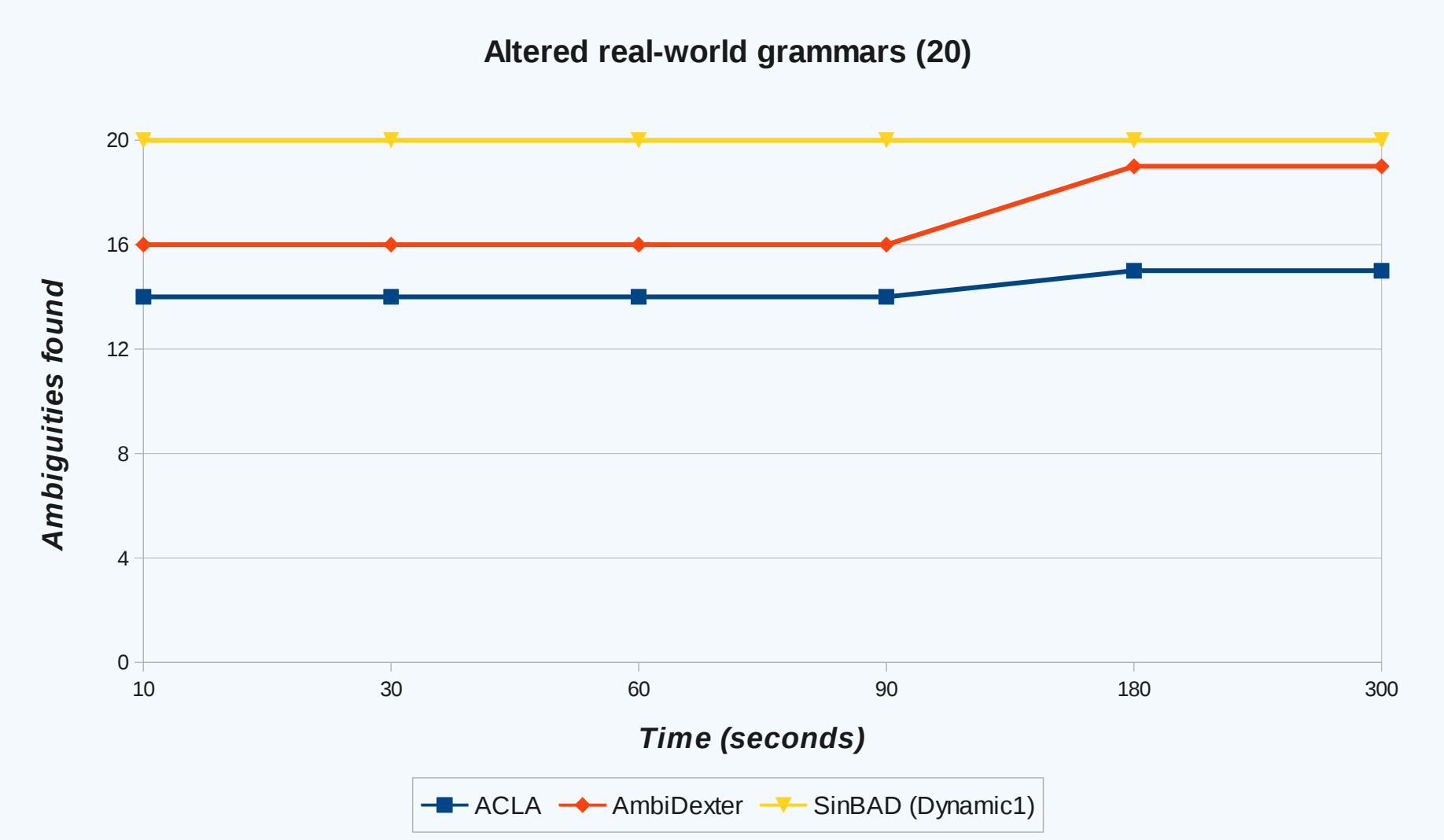
Experiment

- Two sets of grammars
  - Random grammars
  - Real-world PL grammars – altered to be ambiguous
- Time limits
  - Varying time limits (10, 30, 60, 90 seconds)
  - To study how each approach performs

Results


Random grammars (1000)
Ambiguities found
600
500
400
300
200
100
0
10
30
60
90
Time (seconds)
ACLA
AmbiDexter
SinBAD (Dynamic1)

Results


Altered real-world grammars (20)
Ambiguities found
20
16
12
8
4
0
10
30
60
90
180
300
Time (seconds)
ACLA
AmbiDexter
SinBAD (Dynamic1)

Analysis

- ACLA
  - Reports grammar as ambiguous, Unambigous, and possibly ambiguous
  - Random grammars
    - Increasing time limits → diminishing returns
  - Didn't complete for some of the large C grammars

Analysis

- AmbiDexter
  - Fared better than ACLA
  - Random grammars
    - Increasing time limits did **not** improve result
  - Real-world grammars
    - Increasing time limit → better results for large grammars
    - Large grammars → memory intensive and time consuming

Analysis

- SinBAD (Dynamic1)
  - Random grammars
    - Fared better than ACLA and AmbiDexter
    - Increasing time limits improved result
  - Real-world grammars
    - Found ambiguities in relatively short time

Conclusion

- SinBAD
  - **Seems** to do better than other tools on random grammars
  - As well as other tools on real-world grammars
- Download
  - https://github.com/nvasudevan/sinbad

Questions?