5 Papers

Title	Publication source	Year
Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-grained Benchmark, and Feature Evaluation	TSE	2015
On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization	ICSME	2014
Improved bug localization based on code change histories and bug reports	IST	2016
FineLocator: A novel approach to method-level fine-grained bug localization by query expansion	IST	2019
Locating bugs without looking back	MSR	2016

On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization

汇报人：王昭丹

CONTENTS

INTRODUCTION

Dynamic information based

- require instrumenting the software in order to collect the desired data

1

MSR-based

- require collecting and analyzing historical data of software

2

Static analysis based

- only require the version of the software system to be modified in order to extract structural information

3

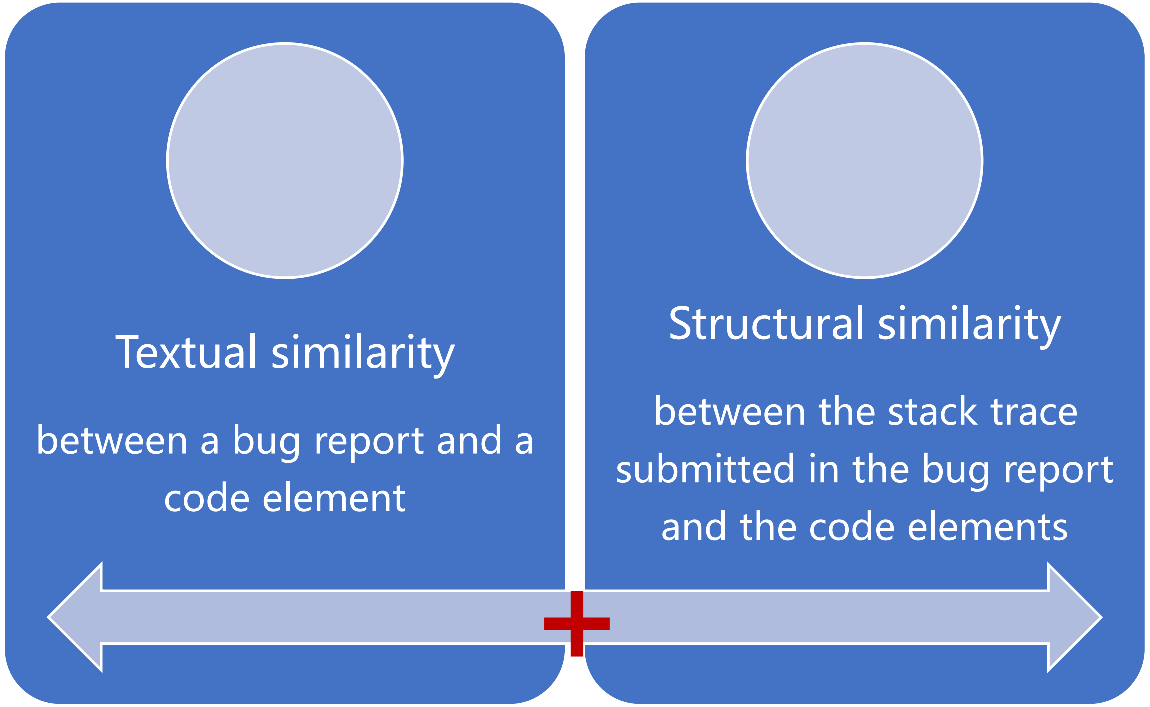
Focus

improving static TR-based bug localization approaches by using additional information → Lobster

INTRODUCTION

Lobster

- the text of bug reports and source code of a software system;
- the stack traces submitted in bug reports;
- the source code structure, in order to identify code relevant to a bug report.



Combine two similarity

TEXT RETRIEVAL & STACK TRACES

Textual Similarity

- define the textual similarity between a bug report and a code element **e** as:

$$sim_{textual}(bugReport, e) = score_{TR}(bugReport, e)$$

- given by Lucene, a TR model that combines VSM and a Boolean model.

range [0,1]

one (1) maximum similarity

zero (0) no similarity

Structural Similarity

- program dependence graph
- define the distance between a stack trace and a code element **e**:

$$dist(stackTrace, e) = \min\begin{pmatrix} \forall_{d \in stackTrace} shortestPath(d, e) \\ \cup \forall_{d \in stackTrace} shortestPath(e, d) \end{pmatrix}$$

- define the structural similarity as the complement of the normalized distance:

$$\underline{sim_{struct}(stackTrace, e)} = 1 - \frac{\min(dist(stackTrace, e), \lambda)}{\boxed{\lambda}}, \quad \lambda \geq 1$$

a threshold defining the maximum considered distance

TEXT RETRIEVAL & STACK TRACES

Structural Similarity

$$\text{sim}_{\text{struct}}(stackTrace, e) = 1 - \frac{\min(\text{dist}(stackTrace, e), \lambda)}{\boxed{\lambda}},$$

indicates how far (in the dependence graph) from the stack trace's elements a code element can be to be considered similar

- **λ=1**

➢ structural similarity: one (1) or zero (0)

➢ one (1) when the code element e is listed in the stack trace (i.e., dist(stackTrace, e) = 0)

- **λ=2**

➢ structural similarity: one (1) ; 0.5 ; or zero (0)

➢ one (1) when the code element e is listed in the stack trace;

➢ 0.5 when the code element e is directly called by or calls an element in the stack trace (i.e., dist(stackTrace, e) = 1);

TEXT RETRIEVAL & STACK TRACES

Total Similarity

- define the total similarity between a bug report and a code element **e** as a linear combination between their textual and structural similarities:

$$\mathrm{sim}(bugReport, e) = (1-\alpha) * \mathrm{sim}_{\mathrm{textual}}(bugReport, e) + \boxed{\alpha} * \mathrm{sim}_{\mathrm{struct}}(\underline{\mathrm{getStackTrace}(bugReport)}, e)$$

$\alpha \in [0,1]$

adjusts the weights of the textual and the structural similarities within the total similarity

Function *getStackTrace* extracts the stack traces from the bug report

EVALUATION

Subject Systems

- Data from **17** versions of **14** open source software systems written in Java
- Extracted the issues whose resolution was marked as **"fixed"** or **"closed"** and whose patch files were available (i.e., attached to the issue)
- Class level

1

TABLE I. SYSTEMS USED IN THE EVALUATION AND THEIR PROPERTIES

System	Version	# of Bug Reports	# of Bug Reports with Stack Traces	# of Classes
ArgoUML[a]	0.22	91	20	1,635
BookKeeper[b]	4.1.0	43	8	587
Derby[b]	10.7.1.1	33	10	3,040
	10.9.1.0	96	26	3,132
Hibernate[b]	3.5.0b2	21	3	4,037
JabRef[a]	2.6	39	3	856
jEdit[a]	4.3	150	8	1,014
Lucene[b]	4.0	35	5	4,317
Mahout[b]	0.8	30	7	3,260
muCommander[a]	0.8.5	92	4	1,443
OpenJPA[b]	2.0.1	35	6	4,438
	2.2.0	18	4	4,955
Pig[b]	0.8.0	85	17	2,095
	0.11.1	48	12	2,506
Solr[b]	4.4.0	55	3	1,863
Tika[b]	1.3	23	3	582
ZooKeeper[b]	3.4.5	80	16	752
Total		*974*	*155*	*40,512*

[a] Part of a benchmark in TR [10]

[b] Data automatically extracted from JIRA

EVALUATION

TR Technique

- Apache Lucene

Text preprocessing techniques

- Split identifiers —— Camel case notation
  removing special characters, keep the original ones.
- remove common English stop words and programming keywords
- stem the words using the Porter stemmer

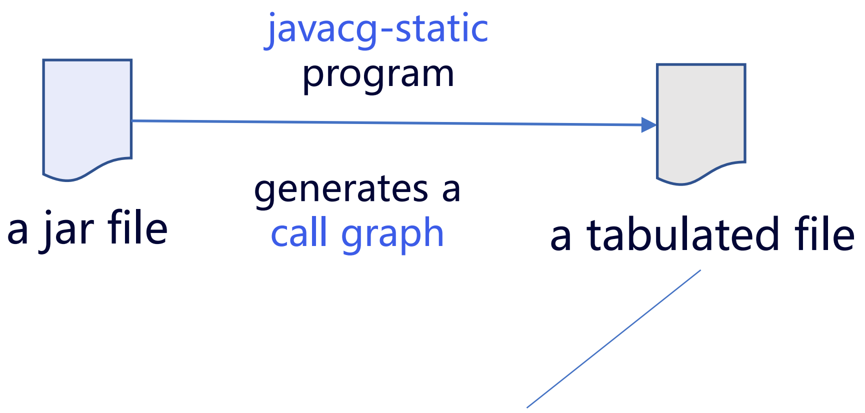
Corpus Creation

- **''bags of words "**
- Create one document for each bug report and class in each subject system.
- For bug reports, we extract the text from their title and description.
- For code classes, we extract the text from their identifiers, comments, and literals.
- Normalize the text of all documents.

EVALUATION

Program Dependence Graph Extraction

- call graph —— java-callgraph suite



contain caller-callee relationships
of classes and methods

Stack Trace Identification

- **Regular expressions** to extract the stack traces from bug reports;
- **Focus on** classes listed in the stack frames;

- Use a regular expression derived from the next abstraction to identify such classes:

```
[packageName]?[className].[methodName](
[filename].java:[lineNumber]|[unknown source|native method])
```

EVALUATION

Methodology

- varying the distance threshold **λ** from 1 to 3
- weight of the structural similarity **α** from 0 to 1 with steps of 0.1
- **Baseline**: setting α=0, i.e., using Lucene solely

$$\text{effectiveness}(q) = \min\left(\forall_{r \in R_q} \text{rank}(r)\right)$$

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{\forall q \in Q} \frac{1}{|R_q|} \sum_{\forall r \in R_q} \text{precision}(\text{rank}(r))$$

$$\text{MRR}(Q) = \frac{1}{|Q|} \sum_{\forall q \in Q} \frac{1}{\text{effectiveness}(q)}$$

Methodology

- **Effectiveness:** the best rank obtained by the set of documents **Rq** relevant to a query **q** within the list of retrieved documents, when sorted in descendent order with respect to the similarity between the query and each document in the corpus.
- **MAP (Mean Average Precision):** the average precision of the set of queries **Q**.
- **MRR (Mean inverse Rank):** the average between the reciprocal effectiveness of a set of queries **Q**.

RESULTS AND DISCUSSION

TABLE II. BUG REPORTS (BRS) AND THEIR PROPERTIES IN TERMS OF STACK TRACES (STS) AND PATCHED CLASSES (PCS)

System	# of BRs with STs	# of PCs[a]	# of PCs with[b]					# of BRs with PCs with[b]	
			dist(*ST*,*PC*)=0	dist(*ST*,*PC*)=1	dist(*ST*,*PC*)=2	3≤dist(*ST*,*PC*) ≤7	dist(*ST*,*PC*)=∞	dist(*ST*,*PC*)=0	dist(*ST*,*PC*)≠∞
ArgoUML 0.22	20	33 (1.7)	10 (30.3%)	14 (42.4%)	7 (21.2%)	0 (0.0%)	2 (6.1%)	9 (45.0%)	19 (95.0%)
BookKeeper 4.1.0	8	30 (3.8)	9 (29.0%)	17 (54.8%)	0 (0.0%)	2 (6.5%)	3 (9.7%)	6 (75.0%)	8 (100%)
Derby 10.7.1.1	10	18 (1.8)	7 (38.9%)	3 (16.7%)	0 (0.0%)	3 (16.7%)	5 (27.8%)	7 (70.0%)	9 (90.0%)
Derby 10.9.1.0	26	49 (1.9)	18 (36.7%)	12 (24.5%)	7 (14.3%)	9 (18.4%)	3 (6.1%)	17 (65.4%)	24 (92.3%)
Hibernate 3.5.0b2	3	7 (2.3)	3 (42.9%)	1 (14.3%)	3 (42.9%)	0 (0.0%)	0 (0.0%)	2 (66.7%)	3 (100%)
JabRef 2.6	3	4 (1.3)	3 (75.0%)	1 (25.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	3 (100.0%)	3 (100%)
jEdit 4.3	8	9 (1.1)	7 (77.8%)	1 (11.1%)	1 (11.1%)	0 (0.0%)	0 (0.0%)	6 (75.0%)	8 (100%)
Lucene 4.0	5	11 (2.2)	8 (72.7%)	3 (27.3%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	5 (100.0%)	5 (100%)
Mahout 0.8	7	11 (1.6)	5 (45.5%)	6 (54.5%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	5 (71.4%)	7 (100%)
muCommander 0.8.5	4	6 (1.5)	2 (33.3%)	4 (66.7%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (50.0%)	4 (100%)
OpenJPA 2.0.1	6	8 (1.3)	5 (62.5%)	1 (12.5%)	1 (12.5%)	0 (0.0%)	1 (12.5%)	5 (83.3%)	6 (100%)
OpenJPA 2.2.0	4	12 (3.0)	2 (16.7%)	4 (33.3%)	5 (41.7%)	1 (8.3%)	0 (0.0%)	2 (50.0%)	4 (100%)
Pig 0.8.0	17	26 (2.2)	6 (23.1%)	5 (19.2%)	3 (11.5%)	0 (0.0%)	12 (46.2%)	5 (41.7%)	9 (75.0%)
Pig 0.11.1	12	46 (2.7)	8 (17.4%)	18 (39.1%)	5 (10.9%)	10 (21.7%)	5 (10.9%)	8 (47.1%)	16 (94.1%)
Solr 4.4.0	3	5 (1.7)	3 (60.0%)	2 (40.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	3 (100.0%)	3 (100%)
Tika 1.3	3	3 (1.0)	2 (66.7%)	1 (33.3%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	2 (66.7%)	3 (100%)
ZooKeeper 3.4.5	16	36 (2.3)	16 (44.4%)	16 (44.4%)	4 (11.1%)	0 (0.0%)	0 (0.0%)	13 (81.3%)	16 (100%)
All	*155*	*314 (2.0)*	*114 (36.2%)*	*109 (34.6%)*	*36 (11.4%)*	*25 (7.9%)*	*31 (9.8%)*	*100 (64.5%)*	147 (94.8%)

a. In parenthesis, average of patched classes per bug report

b. In parenthesis, percentage values

RESULTS AND DISCUSSION

The Impact of λ and α on Lobster's Performance

- **λ** defines the maximum considered distance when computing the structural similarity between a stack trace and a class;
- **α** defines the weight of this structural similarity.
- $\lambda \in \{1,2,3\}$ at $\alpha \in [0.1,1]$

1

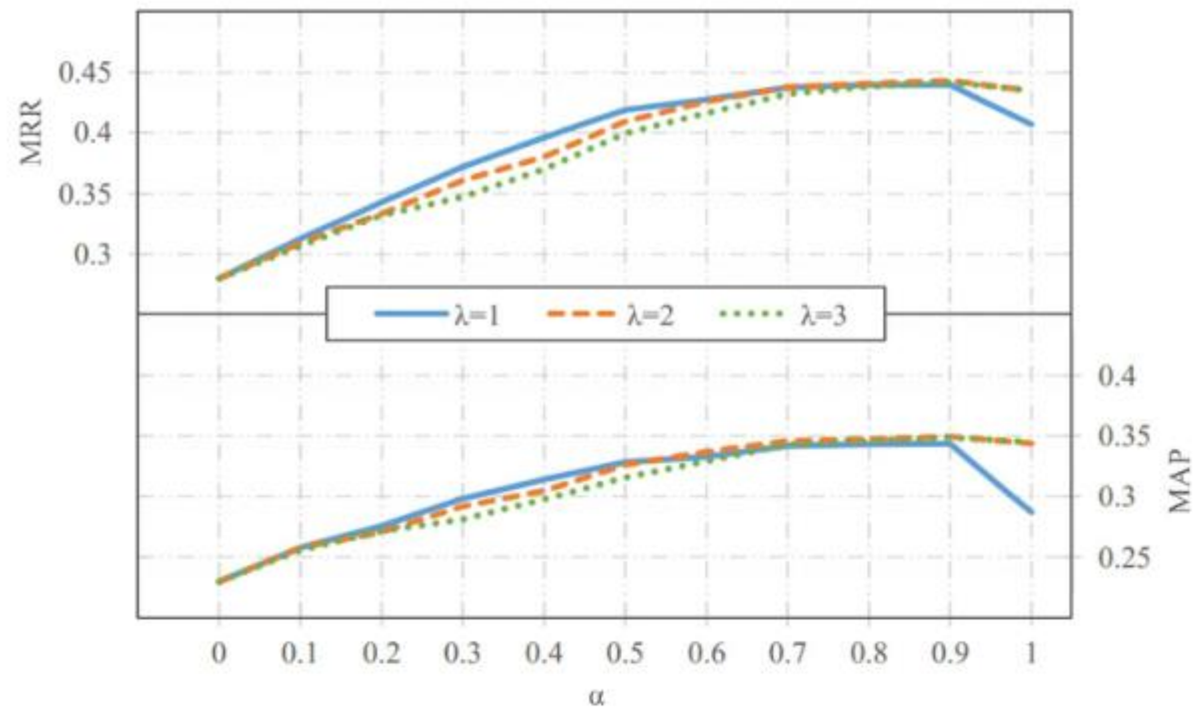


Fig. 1. MRR and MAP values obtained by Lobster on the entire data set with $\lambda \in \{1,2,3\}$ at different values of α.

RESULTS AND DISCUSSION

Lobster vs. Classic TR-based Bug Localization

- **Baseline**: Lucene (i.e., Lobster with $\alpha = 0$)
- Lobster improves, in average, Lucene's effectiveness obtained for **52.6%** of the queries and maintains it for **29.0%** of them. Only in **18.4%** of the cases, in average, Lobster's effectiveness degrades compared to Lucene's.

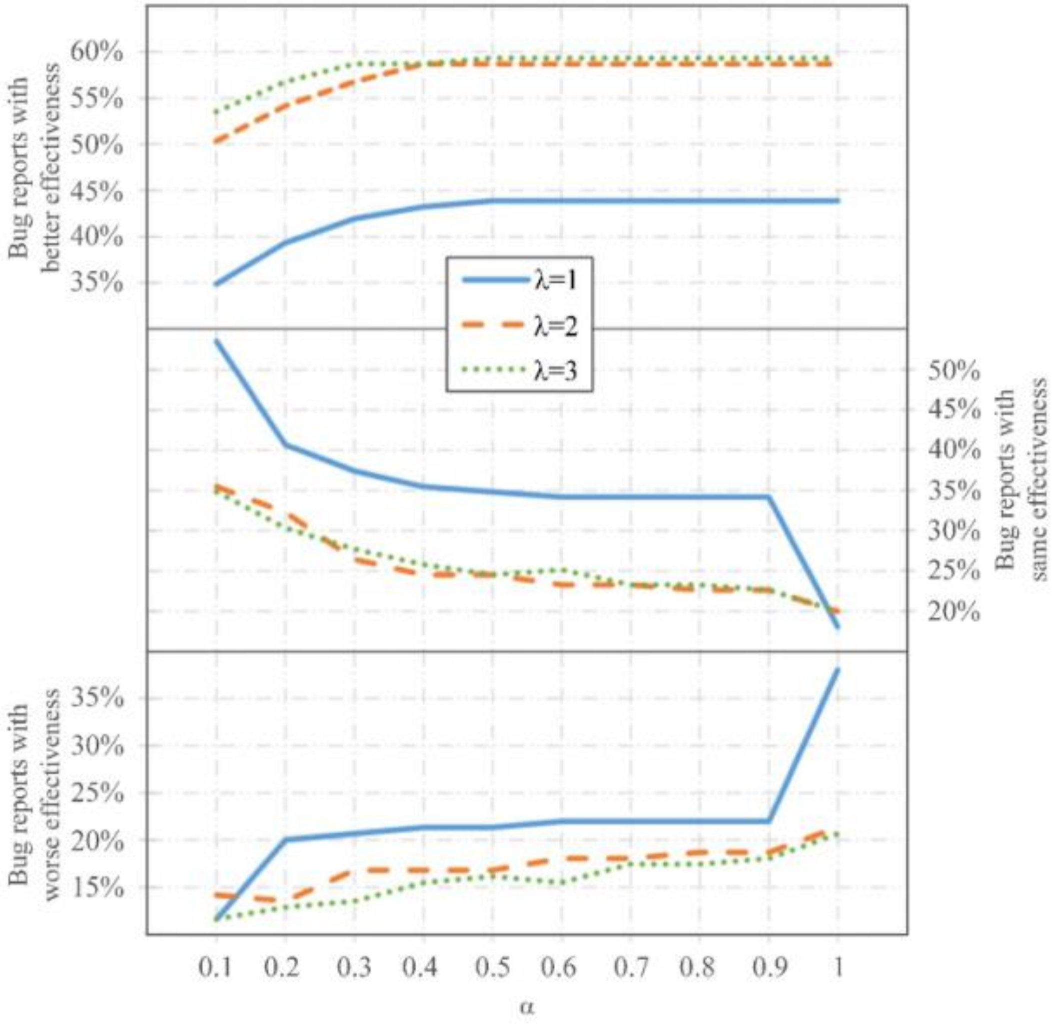


Fig. 2. Percentage of bug reports where Lobster obtains better, the same and worse effectiveness than Lucene with $\lambda \in \{1,2,3\}$ at different values of α

RESULTS AND DISCUSSION

TABLE V. AVERAGE EFFECTIVENESS OF LOBSTER FOR DIFFERENT VALUES OF λ AND α, AND NUMBER OF BUG REPORTS WHERE LOBSTER IS BETTER THAN, SAME AS AND WORSE THAN LUCENE

α	λ=1				λ=2				λ=3			
	Avg. Effect. [a]	***Better*** [b]	***Same*** [b]	***Worse*** [b]	***Avg. Effect.*** [a]	***Better*** [b]	***Same*** [b]	***Worse*** [b]	***Avg. Effect.*** [a]	***Better*** [b]	***Same*** [b]	***Worse*** [b]
0	99.9 (16)	-	-	-	99.9 (16)	-	-	-	99.9 (16)	-	-	-
0.1	82.5 (12)	54 (34.8%)	83 (53.5%)	18 (11.6%)	79.0 (12)	78 (50.3%)	55 (35.5%)	22 (14.2%)	83.1 (12)	83 (53.5%)	54 (34.8%)	18 (11.6%)
0.2	76.1 (8)	61 (39.4%)	63 (40.6%)	31 (20.0%)	69.2 (10)	84 (54.2%)	50 (32.3%)	21 (13.5%)	75.2 (11)	88 (56.8%)	47 (30.3%)	20 (12.9%)
0.3	72.6 (6)	65 (41.9%)	58 (37.4%)	32 (20.6%)	64.7 (8)	88 (56.8%)	41 (26.5%)	26 (16.8%)	72.7 (9)	91 (58.7%)	43 (27.7%)	21 (13.5%)
0.4	71.2 (5)	67 (43.2%)	55 (35.5%)	33 (21.3%)	62.5 (6)	91 (58.7%)	38 (24.5%)	26 (16.8%)	71.4 (8)	91 (58.7%)	40 (25.8%)	24 (15.5%)
0.5	70.7 (5)	68 (43.9%)	54 (34.8%)	33 (21.3%)	63.2 (5)	91 (58.7%)	38 (24.5%)	26 (16.8%)	71.1 (6)	92 (59.4%)	38 (24.5%)	25 (16.1%)
0.6	70.5 (4)	68 (43.9%)	53 (34.2%)	34 (21.9%)	63.7 (5)	91 (58.7%)	36 (23.2%)	28 (18.1%)	73.0 (5)	92 (59.4%)	39 (25.2%)	24 (15.5%)
0.7	70.4 (4)	68 (43.9%)	53 (34.2%)	34 (21.9%)	63.3 (4)	91 (58.7%)	36 (23.2%)	28 (18.1%)	73.5 (4)	92 (59.4%)	36 (23.2%)	27 (17.4%)
0.8	70.4 (4)	68 (43.9%)	53 (34.2%)	34 (21.9%)	63.1 (4)	91 (58.7%)	35 (22.6%)	29 (18.7%)	73.3 (4)	92 (59.4%)	36 (23.2%)	27 (17.4%)
0.9	70.4 (4)	68 (43.9%)	53 (34.2%)	34 (21.9%)	63.1 (4)	91 (58.7%)	35 (22.6%)	29 (18.7%)	73.2 (4)	92 (59.4%)	35 (22.6%)	28 (18.1%)
1	813.7 (5)	68 (43.9%)	28 (18.1%)	59 (38.1%)	374.2 (4)	91 (58.7%)	31 (20.0%)	33 (21.3%)	263.3 (4)	92 (59.4%)	31 (20.0%)	32 (20.6%)

[a.] In parenthesis, median values

[b.] In parenthesis, percentage values

学 习 进 展 & 暑 期 计 划

感谢您的聆听

汇报人：王昭丹