Genetic Improvement on GSON using Gin

Team 9

Task

2019 SSBSE Challenge Track

- GSON[5]
  - convert Java Objects into JSON
  - Huge Open Source Project

Improve GSON using a tool called Gin[1,2]

Gin

Genetic Improvement Tool for JAVA

- Mutation
- Test framework
- Fitness
  - Execution Time Analysis
  - CPU Time Analysis
  - Number of test cases passed
- (https://github.com/gintool/gin)

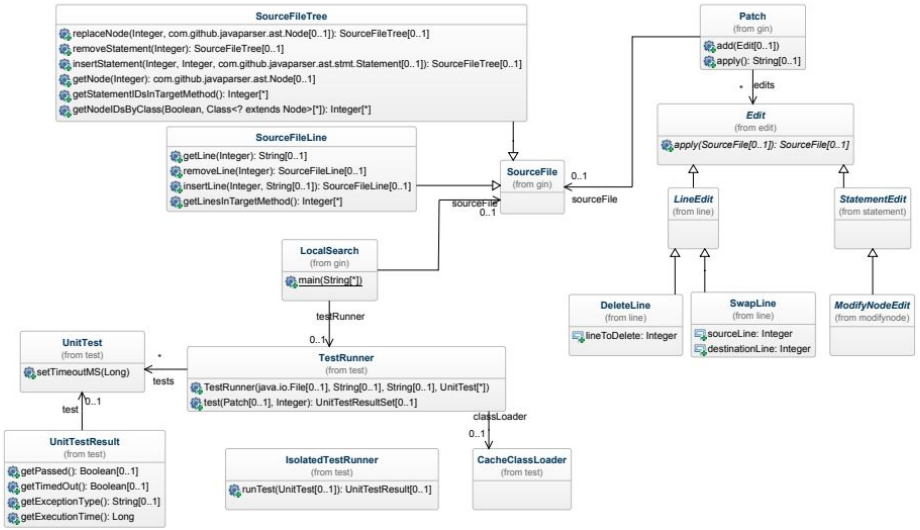


Figure 2: Gin core classes. Attributes are omitted and only a subset of method signatures are shown for simplicity. Also note that several Edit implementations and supporting utility classes are omitted.

Previous Study[3]

- 1-Point Crossover
  - takes two parents and creates two children: one comprising the first half of edits of parent 1 and the second half of edits of parent 2; the second child containing the remaining edits.
- Mutation
  - delete, copy, swap and replace operations
  - binary operator
    - e.g. == with !=
  - unary operator
    - e.g. ++ with --
- Tested with runtime & 3 single injected bugs on GsonBuilder.java

```
List<TypeAdapterFactory> factories = new ArrayList<TypeAdapterFactory>(
this.factories.size()+ this.hierarchyFactories.size()+ 3);
```

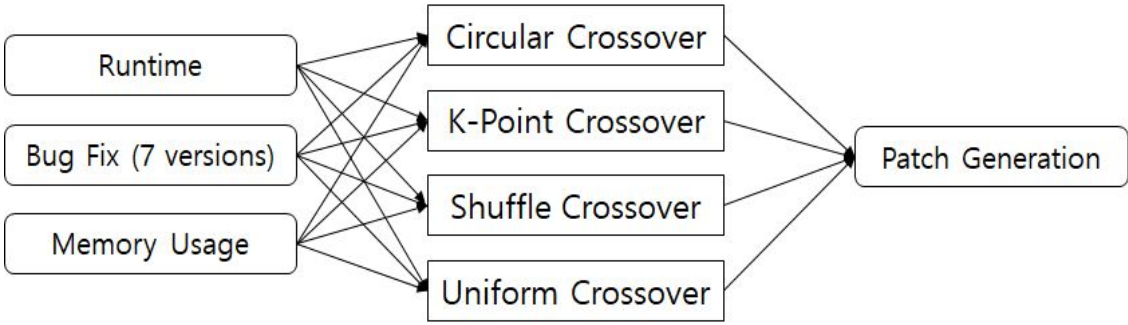
The mutant looked like this:

```
List<TypeAdapterFactory> factories = new ArrayList<TypeAdapterFactory>(
this.factories.size()+ this.hierarchyFactories.size()- 3);
```

What did we do

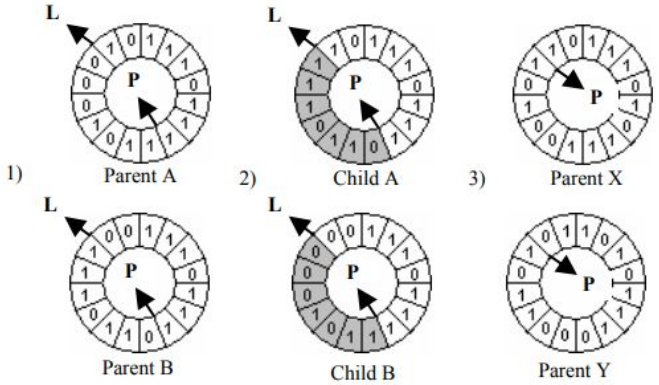
Improve former study by adding

- Implement Memory usage Analysis in Gin
- 4 new crossover operators
  - Circular crossover
  - K-point crossover
  - Shuffle crossover
  - Uniform crossover
- New mutation operator
- Generated 7 bugs to test bug fix on



Circular Crossover

- Make gene in circular form
- P : any point, L (length) : random positive number (L <= len(chromosome))
- interchange two respective segments
- the next bit after the segment of the last crossover becomes new starting point



A Hybrid Fuzzy-Genetic Algorithm DOI: 10.1007/11816157_63

K-Point Crossover

- Total of k cut points (0 <k <n).
- Cut points divide the gene into k + 1 subsets.
- Child is recombined by alternatively selecting subsets of the genes of two parents.

Parent1:	1 | 0 1 | 1 1 | 0 0 1 0 | 1
Parent2:	1 | 1 0 | 0 1 | 1 0 0 1 | 0
Offspring1:	1 | 1 0 | 1 1 | 1 0 0 1 | 1
Offspring2:	1 | 0 1 | 0 1 | 0 0 1 0 | 0

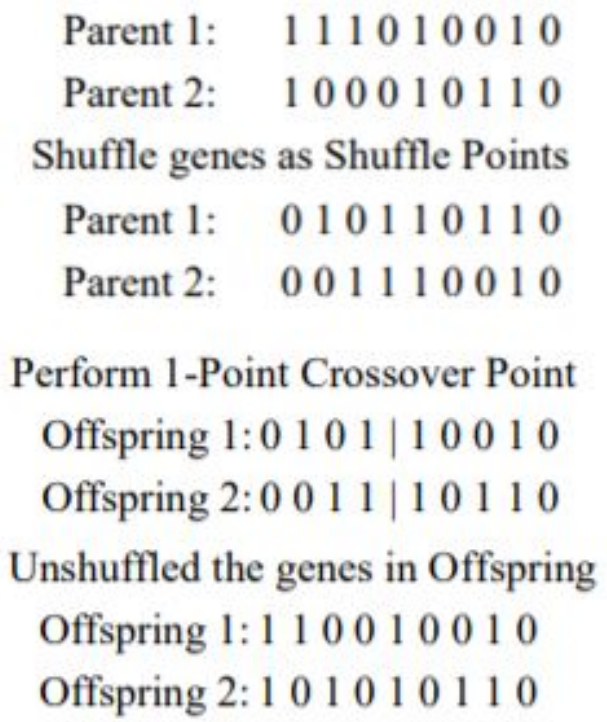
Uniform Crossover

- Make a random real number between 0 and 1 for each iteration .
- The random real number determines whether the child chooses first or second parent as the i-th gene.
- The rest of the child will choose the remaining parent's i-th gene.

Parent1:	1 0 1 1 1 0 0 1 0 1
Parent2:	1 1 0 0 1 1 0 0 1 0
Offspring1:	1 0 0 1 1 1 0 0 1 0
Offspring2:	1 1 1 0 1 0 0 1 0 1

Shuffle Crossover

- Apply the same shuffle on both parents.
- Then apply 1-point crossover.
- Reverse and swap unshuffled genes.



Mutation

- Previous Study
  - addRandomEdit
    - add a random edit to a random location of the patch
  - patch(chromosome) length grow at every iteration
- Ours
  - Replace a random edit instead of just adding one
  - We joined these two methods, letting the operator add a random edit for a certain probability p, and replace a random node with a random edit (without changing the other nodes) for probability 1-p.

Injected Bugs

- PI Test[6]
  - mutation testing system
  - gives fault that test suite can detect
- We use 7 bugs
  - 1 ~ 5 : operator change
  - 6 : statement order change
  - 7 : inject new method


```
# 1
587
    List<TypeAdapterFactory> factories = new ArrayList<TypeAdapterFactory>(this.factories.size() + this.hierarchyFactories.size() + 3);
    List<TypeAdapterFactory> factories = new ArrayList<TypeAdapterFactory>(this.factories.size() + this.hierarchyFactories.size() - 3);

# 2
611
    if (datePattern != null && !"".equals(datePattern.trim()))
    if (!datePattern == null && !"".equals(datePattern.trim()))

# 3
611
    if (datePattern != null && !"".equals(datePattern.trim()))
    if (datePattern != null && "".equals(datePattern.trim()))

# 4
615
    else if (dateStyle != DateFormat.DEFAULT && timeStyle != DateFormat.DEFAULT)
    else if (dateStyle == DateFormat.DEFAULT && timeStyle != DateFormat.DEFAULT)

# 5
615
    else if (dateStyle != DateFormat.DEFAULT && timeStyle != DateFormat.DEFAULT)
    else if (dateStyle != DateFormat.DEFAULT && timeStyle == DateFormat.DEFAULT)
```

Injected Bugs

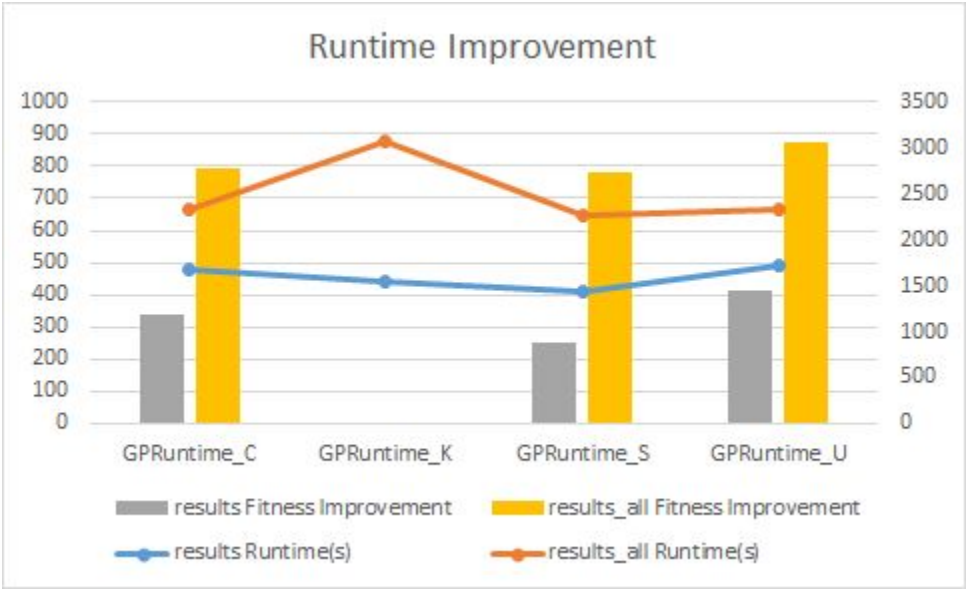
- PI Test[6]
  - mutation testing system
  - gives fault that test suite can detect
- We use 7 bugs
  - 1 ~ 5 : operator change
  - 6 : statement order change
  - 7 : inject new method


```
# 6
588
        # delete
        factories.addAll(this.factories);
599
        # insert
        factories.addAll(this.factories);

# 7
590
        # insert
        Collections.reverse(factories);
```

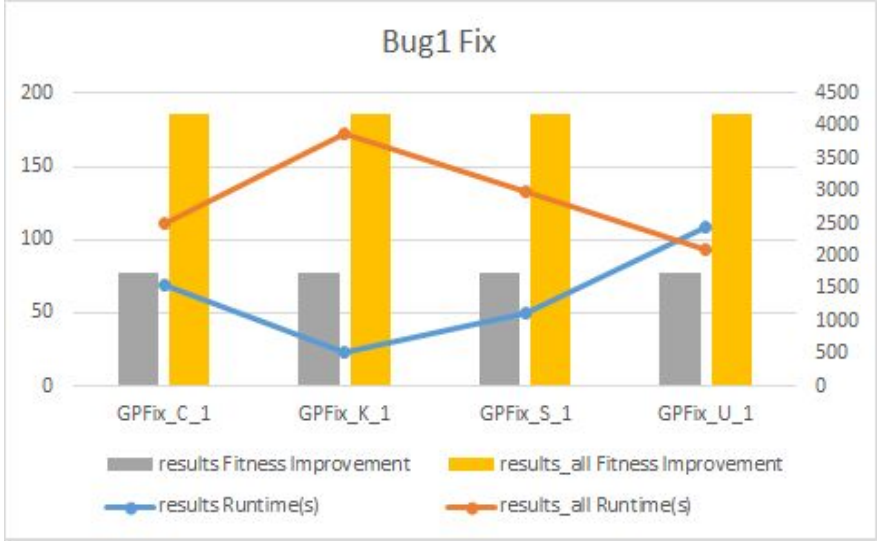
Result - Runtime Improvement

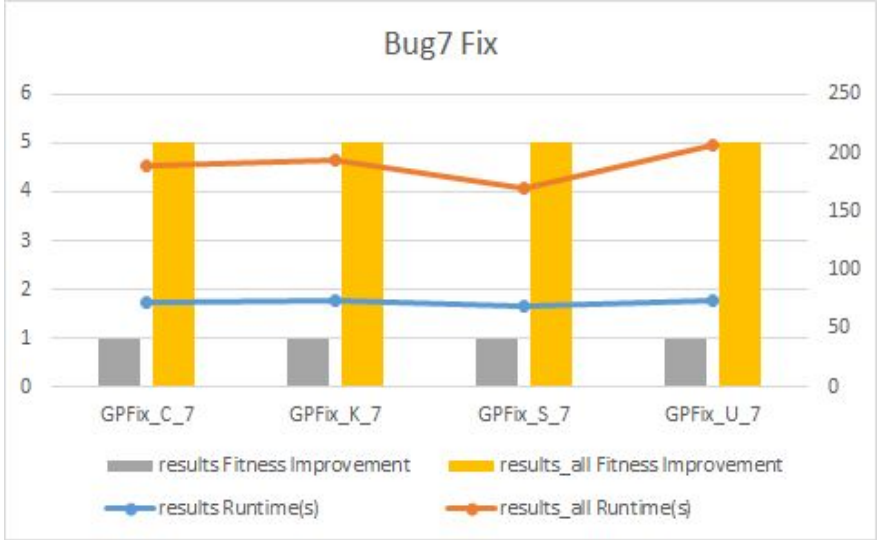
- Only k-point crossover couldn't improve runtime of GSON.
- The others show similar performance, but uniform crossover shows slightly better fitness improvement.



Result - Bug Fix

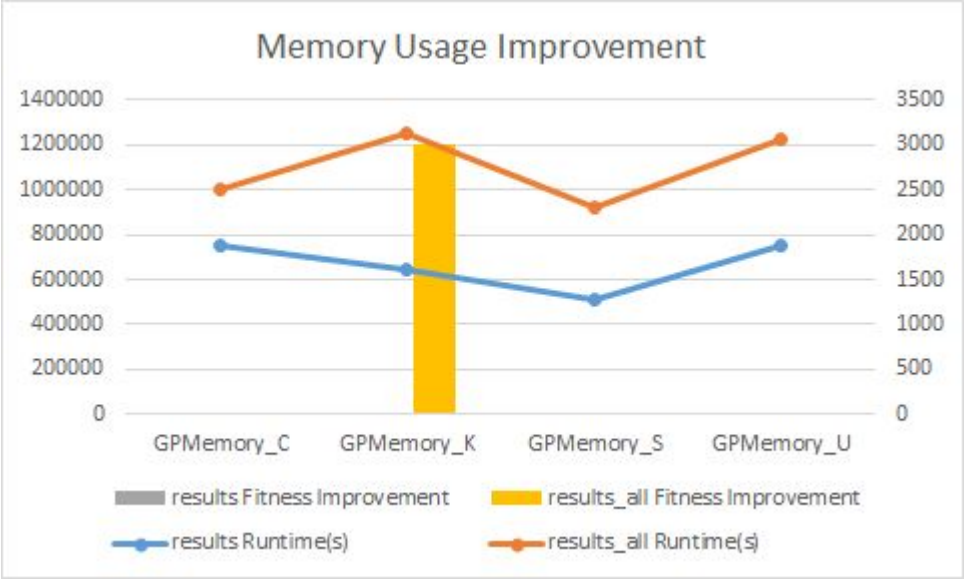
- Only bug 1 and bug 7 were fixed.
- Shows exactly same increment in number of test cases passed between different crossover operators.
- Different crossover operators result in different runtime in fixing bug 1, as opposed to bug 7.





Result - Memory Usage Improvement

- Different from runtime improvement, only k-point crossover improve memory usage of GSON.
- Since k-point found only single patch that improves memory usage while others do not, we cannot confirm that k-point crossover is certainly better than other crossover.



Analysis

- Overall, 4 crossover operators shows similar performance and ability.
- Memory usage was harder to improve than runtime, since simply deleting, copying, swapping statement or changing operator cannot definitely decrease the current memory usage.
- If fault is hardly affected by other parts of codes, the probability of locating the fault significantly decreases.
  - Fault related to *variable a* → fix fault by other part that access or affect *variable a*
  - *a* == *b* to *a* != *b* → fix fault only by changing != operator to ==
- Patch bloat : the size of patch increases as generation number increases → not suitable for fixing small bugs, limit the size of patch
- It would have been better if more data or experiment cases were provided.

Conclusion

- Applied genetic improvement to GSON using Gin
  - with 4 kinds of crossover
  - on 3 domains
- Negligible performance difference between crossover operators
- Successfully improved runtime and memory usage, fixed 2 bugs.
- Problems to be solved:
  - patch bloat
  - fault localization for fixing bugs

Bibliography

1. David R. White. 2017. GI in no time. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17). ACM, New York, NY, USA, 1549-1550. DOI: https://doi.org/10.1145/3067695.3082515
2. Brownlee, Alexander & Petke, Justyna & Alexander, Brad & Barr, Earl & Wagner, Markus & White, David. (2019). Gin: Genetic Improvement Research Made Easy. 985-993. 10.1145/3321707.3321841.
3. Petke J., Brownlee A.E.I. (2019) Software Improvement with Gin: A Case Study. In: Nejati S., Gay G. (eds) Search-Based Software Engineering. SSBSE 2019. Lecture Notes in Computer Science, vol 11664. Springer, Cham
4. Umbarkar, Dr. Anantkumar & Sheth, P.. (2015). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1). 6. 10.21917/ijsc.2015.0150.
5. Google. (2019). GSON, Github Repository, https://github.com/google/gson
6. jekyll. (2019). pitest, software, https://pitest.org/