Appendix: Model-Driven Optimization: Generating Smart Mutation Operators for Multi-Objective Problems[*]

Niels van Harten
niels@vharten.com

Carlos Diego N. Damasceno
d.damasceno@cs.ru.nl

Daniel Strüber
danstru@chalmers.se

June 8, 2022

A Appendix

In this work, we used the single-point crossover with a probability of Pc = 0.9. This crossover operator is frequently used as component of genetic algorithms to solve the NRP [1, 2]. Each mutation rule was executed with a probability of Pm = 0.6. More details about this study are found at [3]. The source code for this project is available at [4].

A.1 RQ1: Operator generation

For each initialization method, we applied our technique to model A, configuring the upper tier to a population size of 60 and 15 iterations with each a timeout of 180 seconds, and the lower tier to a population size of 8 and a maximum of 200 evaluations. We repeated the generation 5 times. We performed the experiment for all five initialization strategies. While testing, we experienced that using the ruleset produced by *extremes* initialization outperformed *rand+x* initialization when using *rand+x* initialization. Therefore, we ignore the *rand+x* ruleset and use the *extremes* ruleset instead for evaluation using *rand+x* initialization. Per initialization method, we used top-scoring generated mutation operator for that initialization method to models A-E, with the exception of *rand+x*, for which we used the *extremes* mutation operator (discussed in paper body).

[*]Appendix of the paper submitted to SEAA 2022

Table 1: RQ2: Results using the fixed (baseline) mutation operator, times in mm:ss:x.

	Empty				**Complete**				**Random**			
Input	NRP			Time	NRP			Time	NRP			Time
model	best	median	stdev	median	best	median	stdev	median	best	median	stdev	median
A	0.407	0.353	0.022	00:12.1	**0.457**	**0.446**	**0.013**	**00:10.0**	0.454	0.439	0.017	00:11.4
B	0.408	0.347	0.020	00:39.9	0.526	0.504	**0.013**	**00:35.3**	**0.589**	**0.554**	0.023	00:38.4
C	0.311	0.279	**0.012**	01:04.0	0.379	0.357	**0.012**	**00:47.8**	**0.508**	**0.461**	0.025	00:59.2
D	0.418	0.399	**0.007**	01:26.3	0.314	0.276	0.010	**01:04.3**	**0.434**	**0.403**	0.019	01:20.9
E	0.198	0.145	0.015	02:11.9	0.218	0.207	**0.007**	**01:48.6**	**0.272**	**0.226**	0.023	02:15.0

Table 2: RQ1: Results for model B (mean, (stdev)), times denoted as mm:ss.xxx.

	Baseline (fixed)			**Contribution (fixedXORgen)**		
Initialization	**HV**	**Spread**	**Runtime**	**HV**	**Spread**	**Runtime**
Complete	0.6967	0.9358	00:21.951	0.1853	**0.4311**	00:20.488
	(0.0414)	(0.0314)	(00:00.546)	(0.019)	**(0.0536)**	(00:00.398)
Empty	0.4125	0.7319	00:32.047	0.3926	0.705	00:21.658
	(0.073)	(0.058)	(00:00.718)	(0.0271)	(0.0329)	(00:00.428)
Extremes	0.2278	0.468	00:27.213	0.231	0.4649	00:21.131
	(0.0161)	(0.0424)	(00:00.315)	(0.0215)	(0.0493)	(00:01.417)
Rand+x	0.222	0.4697	00:29.022	0.2102	0.4708	00:20.806
	(0.0193)	(0.04)	(00:00.339)	(0.0179)	(0.0428)	(00:00.401)
Random	0.267	0.7146	00:28.057	**0.178**	0.5283	00:23.433
	(0.0267)	(0.0737)	(00:00.399)	**(0.0255)**	(0.0488)	(00:02.271)

A.2 RQ2: Initialization strategies and operator generation

Like in the multi-objective case in RQ1, the initialization method has a significant impact on the results of the mono-objective implementation. Because of that, we use different initialization methods for evaluation. Given the results from Table 1, we decided to include both random and complete initialization in our further evaluation: Random initialization seems to perform best for larger models and/or fewer iterations. Complete initialization seems to work best for small models and/or more iterations and seems to lead to a smaller standard deviation and smallest run time.

Using the single objective function shown in the paper, we generated the mutation operator for each initialization method using the smallest model A and reuse it for all models to limit overhead. To create our own mutation operators, we applied our technique to input model A two times. The first time, we generated mutation operators using *complete* initialization, configuring the upper tier to a population size of sixty and fifteen iterations with each a timeout of ninety seconds, and the lower tier to a population size of two and twenty iterations. We repeated the generation ten times; the median run took 13:28 minutes. Next, we generated mutation operators using *random* initialization, configuring the upper tier to a population size of sixty and fifteen iterations with each a timeout of ninety seconds, and the lower tier to a population size of ten and ten iterations. We repeated the generation ten times; the median run took 10:08 minutes.

References

[1] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem," in *GECCO*, 2007.

[2] J. J. Durillo, Y. Zhang, E. Alba, M. Harman, and A. J. Nebro, "A study of the bi-objective next release problem," *Empirical Software Engineering*, vol. 16, 2011.

[3] N. van Harten, "Generating Mutation Operators for a Search-Based Model-Driven Implementation of the Next Release Problem," Jun. 2021. [Online]. Available: https://www.cs.ru.nl/bachelors-theses/2021/Niels_van_Harten___1012159___Generating_Mutation_Operators_for_a_Search-Based_Model-Driven_Implementation_of_the_Next_Release_Problem.pdf

[4] ——, "FitnessStudio applied on the next release problem," 6 2021. [Online]. Available: https://doi.org/10.5281/zenodo.6620783