Front. Comput. Sci., 2025, 0(0): 1–3
https://doi.org/10.1007/sxxxxx-yyy-zzzz-1

LETTERS

GenSC: A Novel and General Local Search Framework for Set Covering Problem

Chuan LUO[1], Taoyu CHEN[2], Renyu Yang(✉)[1], Wei WU[3], Chunming HU[1]

1 School of Software, Beihang University, Beijing 100191, China
2 Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
3 School of Computer Science and Engineering, Central South University, Changsha 410083, China

1 Introduction

Set covering problem (SCP) is a fundamental combinatorial optimization problem in computer science, and it has been extensively applied in real-world, industrial scenarios. Given a universal set of items X and a collection of subsets $Y \subseteq 2^X$, where each subset $y_j \in Y$ is a subset of X, and the union of all subsets in Y is equal to X, each subset $y_j \in Y$ is associated with a positive integer as its weight $\omega(y_j)$. The objective of SCP is to find a collection $S \subseteq Y$ with the minimum total weight to ensure that the union of all subsets in S equals X.

Local search is known to be an effective approach to tackling computationally hard problems [1]. In fact, current state-of-the-art practical SCP algorithms, *i.e.*, *DomSAT* [2] and *NuSC* [3], are based on local search paradigm.

To enhance the flexibility of local search algorithms, a powerful and novel algorithm design paradigm called programming by optimization (PbO) [4] has been proposed. The main idea of PbO encourages algorithm developers to greatly expand the design space of an algorithm through incorporating various effective techniques. To our knowledge, there is no existing work that applies the PbO paradigm to solving SCP. Hence, it is promising to develop a PbO-based local search algorithm for solving extensive types of SCP instances. The main contributions of this work are as follows:

- **Extensive Algorithmic Components.** *GenSC* consists of many algorithmic components, where each component is considered as an abstraction and has multiple candidate instantiations.
- **Automatic Combination of Techniques.** *GenSC* is equipped with diverse algorithmic techniques that are automatically selected and combined through a powerful configurator, and *GenSC* is expected to achieve state-of-the-art performance on various SCP instances.

2 Our Novel *GenSC* Framework for SCP

GenSC incorporates various effective techniques and can determine suitable combinations of techniques for different SCP instances. It takes an SCP instance I as input, and outputs a feasible solution S^*. *GenSC* consists of three stages, *i.e.*, pre-processing stage, initialization stage and local search stage.

2.1 Pre-processing Stage

Given a SCP instance $I = (X, Y)$, an item $x_i \in X$ is an unit item if x_i is included by only one subset in Y. A subset $y_j \in Y$ is a compulsory subset if y_j includes at least one unit item. Since a feasible solution covers all items, it must contain all compulsory subsets, denoted by S'. The instance can be simplified to I' by filtering unit items and compulsory subsets.

Received June 07, 2024; accepted Janurary 07, 2025

E-mail: renyuyang@buaa.edu.cn

GenSC keeps track of the current solution as S and the best solution S^* is updated as the search progresses. Once both initialization stage and local search stage terminate, *GenSC* reports the union of S^* and S' as the final, feasible solution.

2.2 Initialization Stage

In the initialization stage, *GenSC* aims to generate a feasible solution S to simplified instance I'. In order to build a highly-quality initial solution, we design a greedy construction component called `GreedyConstruct`. It starts with an empty solution S, and then iteratively adds a subset into S until S becomes a feasible solution.

GenSC implements two instantiations of the component `GreedyConstruct`. Particularly, we employ a categorical hyper-parameter given by a configuration of *GenSC* to indicate which of the instantiations is adopted. We note that all choices regarding multiple instantiations for *GenSC*'s algorithmic components are processed similarly.

2.3 Local Search Stage

At the start of each iteration, if S is feasible, *GenSC* first removes a random subset from S. Then *GenSC* switches between two modes to find better solutions starting from this infeasible solution.

2.3.1 Exploitation Mode

This mode focuses on minimizing the optimization objective. The main techniques in this mode are discussed below.

Forbidden Strategies for Tackling Cycling Challenge. To mitigate the cycling challenge and enhance the effectiveness of *GenSC* when handling various instances, we incorporate 3 effective forbidden strategies into the component `TackleCycling`, which are based on configuration checking and tabu mechanism [3].

Selecting the Subset to be Added. To complete the selection task, we design an `AddSubset` component. *GenSC* includes 3 instantiations of `AddSubset`, considering the weights and the covered items of the subsets. After a subset is added by `AddSubset`, *GenSC* forms a new solution $\overline{S}$, as the output of the exploitation mode.

2.3.2 Exploration Mode

In the exploration mode, local search prefers to better explore the search space. *GenSC* tries to conduct another swapping operation to modify current solution S. For this swapping operation, we design a `RmSubset` component to remove a subset from S and use `AddSubset` to add a subset, forming a new solution $\widehat{S}$. *GenSC* supports 3 instantiations of `RmSubset`.

After swapping, *GenSC* evaluates the quality of the current solution: if $\omega(\widehat{S}) < \omega(S)$, then *GenSC* possibly finds promising search space and updates the current solution as $\widehat{S}$; otherwise, *GenSC* revokes the adding operation and continues the search process.

To decide which mode should be activated in each iteration, we design a determination mechanism. By comparing the weights of the solution returned by exploitation mode and the solution at the start of the iteration, we can examine whether exploitation mode optimizes the objective. *GenSC* tends to exploit more if it is far from local optima; otherwise, it works in the exploration mode.

3 Experimental Design and Results

3.1 Benchmarks and Competitors

We compare *GenSC* with 5 state-of-the-art SCP algorithms, *i.e., NuSC* [3], *DomSAT* [2], *Open-WBO* [5], *Loandra* [6] and *SATLike* [7].

3.2 Configuration Protocol of *GenSC*

The settings of all hyper-parameters of *GenSC* can be automatically decided by an algorithm configurator. We use an effective configurator *SMAC* [8] to determine high-performance configurations for *GenSC*. *SMAC* iteratively records the performance of configurations to construct a predictive model. In each iteration, *SMAC* selects the configuration that the model predicts is most likely to enhance performance.

3.3 Experimental Setup

For each algorithm, we present the best solution weights, denoted by 'min', the average value of all solution weights, denoted by 'avg', and the average running time for achieving the best solution, denoted by 'time'. Moreover, as suggested by the literature [3], we report the metric of average relative percentage deviation, denoted by 'ARPD'. For each instance, the best results are highlighted using the **boldface** font.

3.4 Experimental Results

The comparative results of *GenSC* and its state-of-the-art competitors on the adopted instances are reported in Table 1. The

Table 1 Comparative results of *GenSC* and its state-of-the-art competitors. The value of ARPD is measured in percentage (%), and the running time is measured in second.

Instance	*DomSAT*		*Loandra*		*Open-WBO*		*SATLike*		*NuSC*		*GenSC*	
	min avg	ARPD time	min avg	ARPD time	min avg	ARPD time	min avg	ARPD time	min avg	ARPD time	min avg	ARPD time
`rail-2536*`	700	0.86	1404	103.53	1537	121.15	-	-	698	0.46	**695**	**0.13**
	701.0	530.03	1414.5	60.31	1537.0	1.24	-	-	698.2	476.70	**695.9**	713.72
`rail-2586`	966	1.00	1389	47.40	1293	35.11	-	-	960	0.53	**957**	**0.36**
	966.6	653.06	1410.6	1000.00	1293.0	23.37	-	-	962.1	936.80	**960.4**	985.24
`rail-4284*`	1092	0.77	1887	87.14	2315	113.36	-	-	1087	0.36	**1085**	**0.14**
	1093.4	972.17	2030.5	65.91	2315.0	1.38	-	-	1088.9	716.10	**1086.5**	493.32
`rail-4872`	1558	0.68	2574	84.24	2109	36.06	-	-	1552	0.30	**1550**	**0.09**
	1560.5	627.06	2855.7	59.28	2109.0	24.88	-	-	1554.7	700.60	**1551.4**	573.50
`scpcyc-08*`	344	1.34	373	9.06	448	30.99	371	9.27	344	0.88	**342**	**0.00**
	346.6	527.30	373.0	238.75	448.0	0.03	373.7	39.74	345.0	18.70	**342.0**	1.06
`scpcyc-09`	780	1.11	965	25.00	1024	32.64	925	20.60	780	1.13	**772**	**0.28**
	780.6	159.14	965.0	1000.00	1024.0	0.08	931.0	201.63	780.7	4.20	**774.2**	35.20
`x264*`	5103	0.35	**5095**	**0.00**	5106	0.22	5108	0.38	5101	0.25	5096	0.04
	5112.9	<0.01	**5095.0**	95.97	5106.0	2.78	5114.2	536.43	5107.9	38.80	5097.0	466.79
`BDBC*`	54175	0.01	54241	0.13	**54172**	**0.00**	54177	0.01	54183	0.03	54173	<0.01
	54176.8	33.50	54241.0	24.90	**54172.0**	1.57	54179.9	655.22	54186.2	246.00	54174.1	165.82
`Dune*`	237640	0.58	325022	37.32	264851	11.89	238449	0.93	237199	0.29	**236698**	**0.01**
	238079.0	38.56	325022.0	0.93	264851.0	908.81	238903.9	353.79	237375.6	55.50	**236731.4**	541.02
`hsmgp`	173390	0.11	183611	5.96	231797	33.76	173610	0.27	173315	0.06	**173291**	**0.01**
	173482.1	111.99	183611.0	22.33	231797.0	283.75	173752.7	496.00	173394.7	667.60	**173303.5**	133.42
`hipacc`	289091	0.70	360909	25.62	366085	27.42	289130	0.74	288586	0.53	**287302**	**0.01**
	289315.9	943.20	360909.0	15.01	366085.0	13.05	289417.3	598.06	288829.7	969.20	**287316.6**	667.28
`sac*`	77646599	0.04	78072801	0.60	77933534	0.45	-	-	77671334	0.07	**77620962**	**<0.01**
	77653261.1	1000.00	78084169.0	113.73	77966843.5	1000.00	-	-	77674840.1	987.80	**77621275.0**	734.81
`javagc`	17512677	6.13	17955522	10.19	18708768	12.87	-	-	16937627	2.54	**16575919**	**<0.01**
	17592635.0	956.75	18265103.4	1000.00	18708768.0	912.00	-	-	16997043.2	994.20	**16576267.0**	961.60

training set is indicated with an asterisk. It is clear that *GenSC* stands out as the best SCP algorithm and considerably outperforms all other competing algorithms on all instances except two (*i.e.,* `x264` and `BDBC`). For the largest instances (*i.e.,* `rail` series, `sac` and `javagc`), *SATLike* fails to report a solution, so we mark its results as '–' for these instances in Table 1.

4 Conclusion

In this work, we propose a novel, general local search framework, dubbed *GenSC*, for set covering problem. *GenSC* is designed to be highly configurable, and its design space incorporates many effective algorithmic components for solving SCP, enabling *GenSC* to be configured to instantiate various concrete local search SCP algorithms. Overall, our experimental results demonstrate that *GenSC* substantially pushes forward the state of the art in SCP solving.

Data Availability Statement. The long version of this paper, the implementation of *GenSC*, and all benchmarks adopted in our experiments are publicly available in our repository: `https://github.com/chuanluocs/GenSC`.

Acknowledgements This work was supported in part by the National Key Research and Development Program of China (Grant 2023YFB3307503), in part by the National Natural Science Foundation of China (Grants 62202025, 62302528, 62402024) and Beijing Natural Science Foundation (L241050), in part by the Young Elite Scientist Sponsorship Program by CAST (Grant YESS20230566), in part by CCF-Huawei Populus Grove Fund (Grant CCF-HuaweiFM2024005), and in part by the Fundamental Research Funds for the Central Universities.

References

1. Liu C, Liu G, Luo C, Cai S, Lei Z, Zhang W, Chu Y, Zhang G. Optimizing local search-based partial MaxSAT solving via initial assignment prediction. Science China Information Sciences, 2025, 68(2)
2. Lei Z, Cai S. Solving set cover and dominating set via maximum satisfiability. In: Proceedings of AAAI 2020. 2020, 1569–1576
3. Luo C, Xing W, Cai S, Hu C. NuSC: An effective local search algorithm for solving the set covering problem. IEEE Transactions on Cybernetics, 2022, 1–14
4. Hoos H H. Programming by optimization. Communications of the ACM, 2012, 55(2): 70–80
5. Martins R, Manquinho V M, Lynce I. Open-WBO: A modular MaxSAT solver,. In: Proceedings of SAT 2014. 2014, 438–445
6. Berg J, Demirović E, Stuckey P J. Core-boosted linear search for incomplete MaxSAT. In: Proceedings of CPAIOR 2019. 2019, 39–56
7. Cai S, Lei Z. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. Artificial Intelligence, 2020, 287: 103354
8. Hutter F, Hoos H H, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: Proceedings of LION 2011. 2011, 507–523