TRB 2026
Annual Meeting
Washington, D.C.

Repositioning in Bike Sharing Systems with Broken Bikes Considering On-site Repairs

Optimizing truck routes and repairer deployment to reduce user dissatisfaction

Runqiu Hu[1], W.Y. Szeto[1,2,3], Sin C. Ho[4]

[1]Dept. of Civil Engineering, HKU [2]HKU Shenzhen Institute of Research and Innovation [3]GD-HK-Macau Joint Lab for Smart Cities [4]CUHK

Transportation Research Part E (2025)
DOI: 10.1016/j.tre.2025.104155

The Challenge: Bike Sharing Operations

Demand Imbalance

[Placeholder: Illustration of stations becoming empty or full during peak hours]

Stations become empty or full
→ Users cannot pick up/return bikes

Broken Bikes

Barcelona: 67% bikes damaged
NYC: 2% need daily repairs

Current solutions use trucks to relocate bikes between stations.
But what about **repairing bikes on-site**?

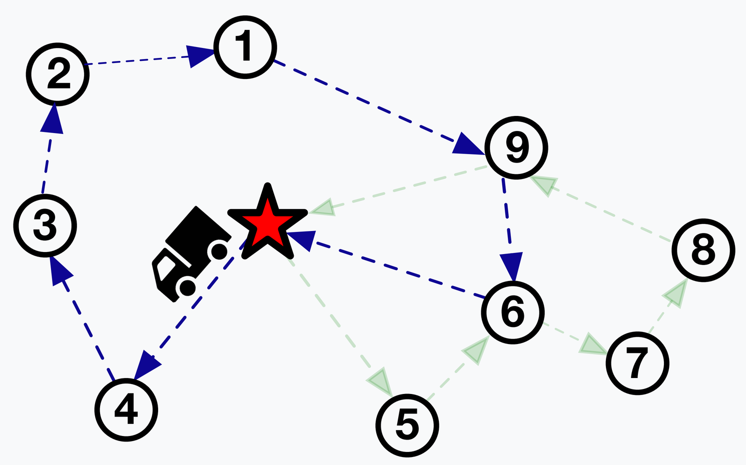
Research Gap & Our Question

Existing Studies Focus On:

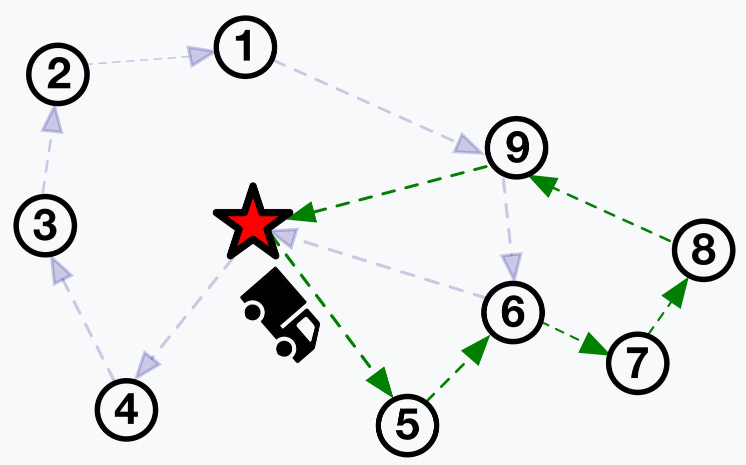
- ✓ Truck-based bike repositioning
- ✓ Moving broken bikes to repair depots
- × On-site repairs by dedicated repairers

Research Question

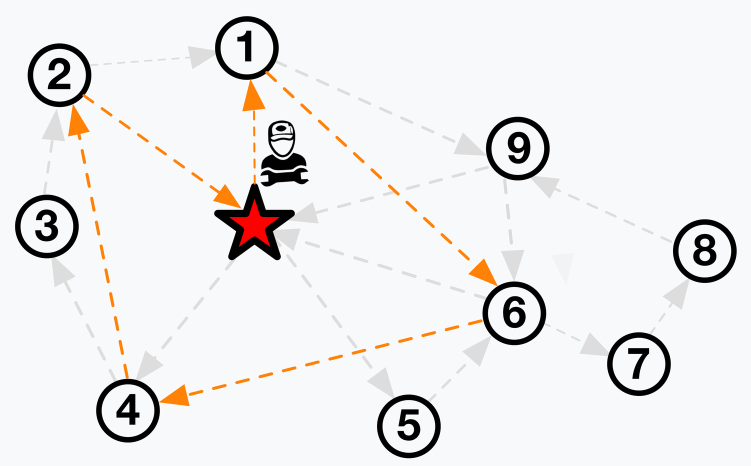
How can we **jointly optimize** truck repositioning and **on-site bike repairs** to improve service quality?



Truck 1 route: 0-4-3-2-1-9-6-0



Truck 2 route: 0-5-6–7–8–9-0



Repairer route: 0-1-6-4-2-0

Our integrated system: Trucks (blue routes) relocate bikes while repairers (dashed routes) fix bikes on-site.

Our Approach: Hybrid Genetic Search Algorithm

Two-Part Problem Framework:

Part 1: Route Decision
Local Search Operators

- ▶ Crossover generates offspring routes
- ▶ Education via local search
- ▶ Improves route quality

Part 2: Quantity Assignment
SBC Allocation

- ▶ Station-Budget-Constrained
- ▶ Loading/unloading/repairing
- ▶ Completes solution evaluation

Key Highlight: Maintains both **feasible** and **infeasible** solutions in the population
Infeasible solutions explored → repaired → may become feasible

Algorithm 1: HGSADC-SBC

Initialization: Generate a population *Pop* consisting of a feasible subpopulation Pop_f and an infeasible subpopulation Pop_i
Set *iter_no_imp* = 0 and *current_best_value* = +∞
while *iter_no_imp* < It_{NI} and *cpu_time* < T_{max} **do**
Randomly select two individuals Ind_1 and Ind_2 from *Pop*
Generate the routes of offspring C_1 and C_2 using ordered crossover on the routes of Ind_1 and Ind_2, then apply the SBC heuristic on the routes to determine the (un)loading and repairing quantities for C_1 and C_2
Educate on both C_1 and C_2 by applying local search procedures to generate new routes and using the SBC heuristic to determine the (un)loading and repairing quantities
if C_i ($i = 1, 2$) is feasible **then**
  Add C_i to the feasible subpopulation Pop_f
else
  Add C_i to the infeasible subpopulation Pop_i and repair C_i with the probability P_{repair} using local search procedures and the SBC heuristic
end if
if the individuals become feasible after repairs **then**
  The repaired individuals are added to the feasible subpopulation Pop_f
else
  The repaired individuals are discarded, and the original infeasible solution is kept in the infeasible subpopulation Pop_i
end if
if the size of Pop_f or Pop_i reaches the maximum subpopulation size **then**
  Select survivors from the corresponding subpopulation
end if
Set *bs* = the solution with the best fitness value from Pop_f and set *bv* = the fitness value of *bs*
if $bv \geq$ *current_best_value* **then**
  Set *iter_no_imp* = *iter_no_imp* + 1
Else
  Set *current_best_value* = *bv*, *current_best_sol* = *bs*, and *iter_no_imp* = 0
end if
if *iter_no_imp* ≥ It_{div} **then**
  Diversify the population consisting of Pop_f and Pop_i and adjust the penalty parameter
end if
end while
return *current_best_sol*, *current_best_value*

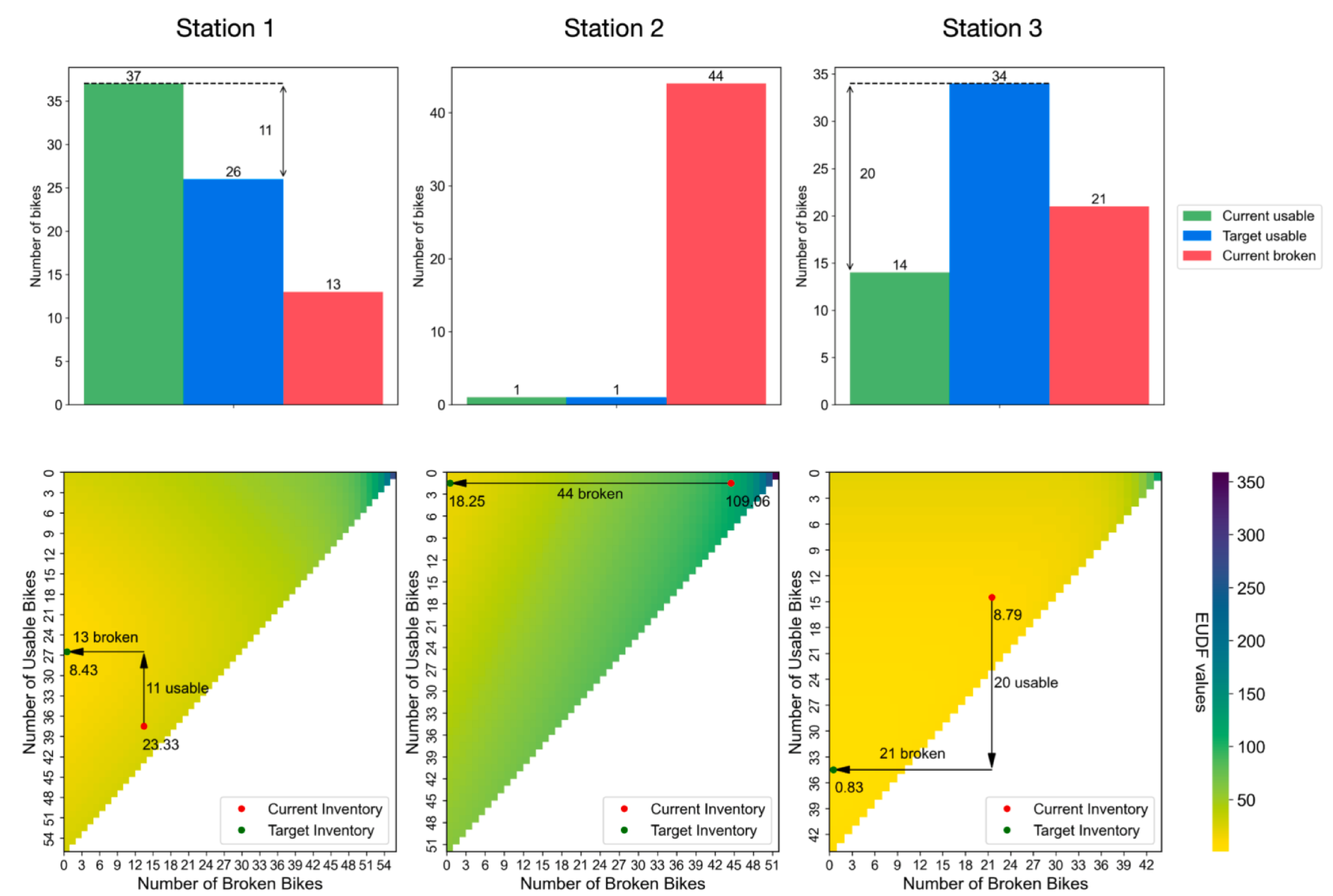
HGSADC-SBC: Routes optimized via local search, then quantities assigned using SBC allocation.

Key Innovation: Smart Time Allocation

The Problem: Spending too much time at early stations leaves no time for later ones.

Our Solution: Station Budget Constraint

- ▶ Each station gets a **time budget** based on its potential benefit
- ▶ Stations with more broken bikes get more time
- ▶ Unused time **redistributed** to later stations



$$BCRF_1^s = \frac{23.33 - 8.43}{(11+13) \times 60} = 0.0103 \quad BCRF_2^s = \frac{109.06 - 18.25}{44 \times 60} = 0.0344 \quad BCRF_3^s = \frac{8.79 - 0.83}{(20+21) \times 60} = 0.0032$$

Prioritizing stations: Heatmaps show dissatisfaction levels. Darker colors indicate higher priority for service.

Result: Better service across all stations, not just the first few.

Results: Algorithm Performance

Comparison with Commercial Solver (Gurobi):

Table 5.4
A comparison between the results of Gurobi and HGSDAC-SBC on larger networks (Repositioning time budget $T = 2$ h, $\tau = 600$ s).

$\lvert S \rvert$	$\lvert K \rvert$	$\lvert R \rvert$	Gurobi			HGSDAC-SBC				Gap (%)[5]
			UB	LB	CPU (s)	Obj. Avg.[1]	Obj. Best.[2]	Obj. Std.[3]	CPU (s)[4]	
60	1	1	2604.63	2517.12	7200.00	2581.29	2579.82	0.86	6.37	0.90
	1	2	2533.44	2420.32	7200.00	2494.24	2486.32	4.09	9.20	1.55
	2	1	3581.80	2284.28	7200.00	2400.65	2379.21	10.18	9.87	32.98
	2	2	3517.71	2202.55	7200.00	2324.50	2302.98	11.76	10.90	33.92
90	1	1	4510.58	3893.04	7200.00	4038.28	4014.71	13.59	8.81	10.47
	1	2	4159.43	3782.32	7200.00	3941.78	3915.27	14.03	13.40	5.23
	2	1	4510.58	3656.81	7200.00	3862.44	3829.09	19.77	9.56	14.37
	2	2	4488.64	3557.92	7200.00	3768.58	3733.90	21.59	13.28	16.04
120	1	1	5113.72	4247.68	7200.00	4303.05	4293.75	3.58	9.13	15.85
	1	2	5103.87	4169.64	7200.00	4237.52	4226.39	4.17	14.96	16.97
	2	1	5113.72	4024.31	7200.00	4167.75	4129.41	16.19	11.45	18.50
	2	2	5103.87	3951.26	7200.00	4097.19	4071.74	11.87	15.98	19.72
200	1	1	8657.30	2919.70	7200.00	7821.12	7793.53	19.35	12.36	9.66
	1	2	8657.30	7563.69	7200.00	7719.58	7694.50	12.08	23.96	10.83
	2	1	8657.30	7378.11	7200.00	7641.82	7573.84	28.23	14.19	11.73
	2	2	8657.30	7286.86	7200.00	7561.16	7511.34	22.96	23.90	12.66
300	1	1	11717.13	10898.80	7200.00	11169.22	11134.10	15.16	16.55	4.68
	1	2	11716.99	10757.43	7200.00	11057.76	10990.90	19.38	25.67	5.63
	2	1	11717.13	10561.75	7200.00	10989.39	10953.90	19.84	19.59	6.21
	2	2	11716.99	10442.63	7200.00	10888.38	10848.40	20.63	33.70	7.07
400	1	1	15068.74	6026.88	7200.00	14280.67	14267.10	5.89	25.27	5.23
	1	2	15060.58	6026.88	7200.00	14178.63	14149.00	12.33	34.05	5.86
	2	1	15068.74	6026.59	7200.00	14169.14	14104.30	21.84	25.92	5.97
	2	2	15060.58	6026.59	7200.00	14059.18	14033.60	15.29	43.77	6.65
500	1	1	19793.68	7716.08	7200.00	18961.58	18927.40	18.77	27.36	4.20
	1	2	19785.07	7716.08	7200.00	18793.14	18734.00	22.68	41.60	5.01
	2	1	19793.68	7715.69	N/A	18752.12	18618.10	50.56	33.65	5.26
	2	2	19785.07	N/A	N/A	18612.62	18505.40	37.57	51.13	5.93

[1] the average objective value for each instance in 20 runs.
[2] the best objective value for each instance in 20 runs.
[3] the standard deviation of the objective value for each instance in 20 runs.
[4] average computational time for each instance in 20 runs in seconds.
[5] (UB – Obj. Avg) / UB.

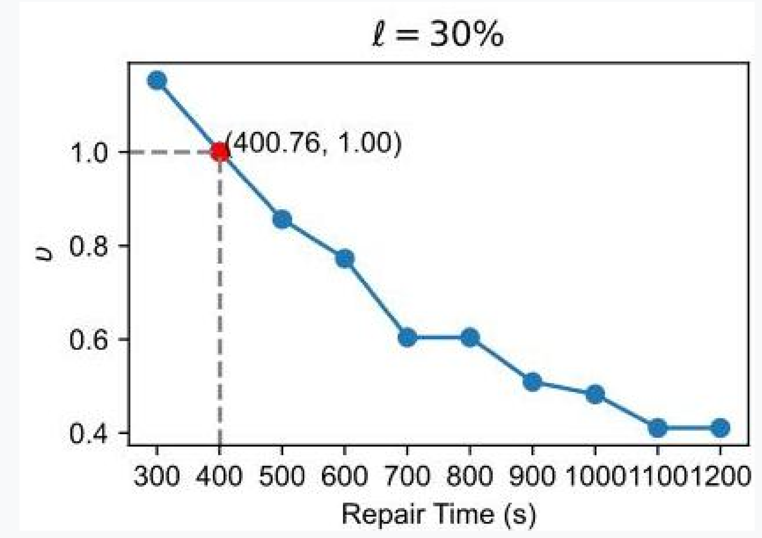
4–33%
Better solution quality
(lower user dissatisfaction)

6–51 sec
vs. over 2 hours
Computation time

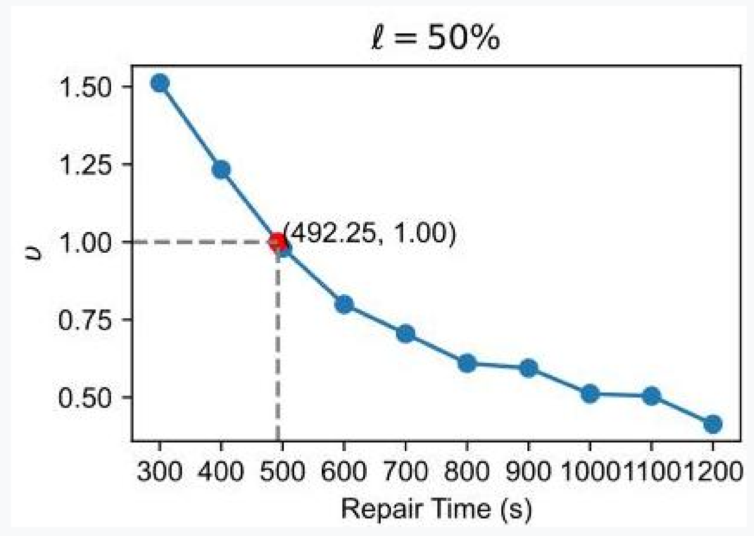
Scales to real cities: Tested on networks with up to **500 stations**
(Gurobi fails for networks larger than 200 stations)

Practical Takeaways for Operators

When should you deploy on-site repairers?



30% broken bikes



50% broken bikes

Key Decision Rules:

- ▶ **< 20% broken:** Trucks alone sufficient
- ▶ **> 30% broken:** On-site repairers cost-effective
- ▶ **Repair time > 11 min:** Never cost-effective

Bottom Line: Invest in preventive maintenance to keep repair times low.

Reference: Hu, R., Szeto, W.Y., & Ho, S.C. (2025). Repositioning in bike sharing systems with broken bikes considering on-site repairs. *Transportation Research Part E*, 195, 104155.

Acknowledgments: NSFC (71771194), RGC Hong Kong (17206322)

[QR Code]
Scan for paper

DOI: 10.1016/j.tre.2025.104155