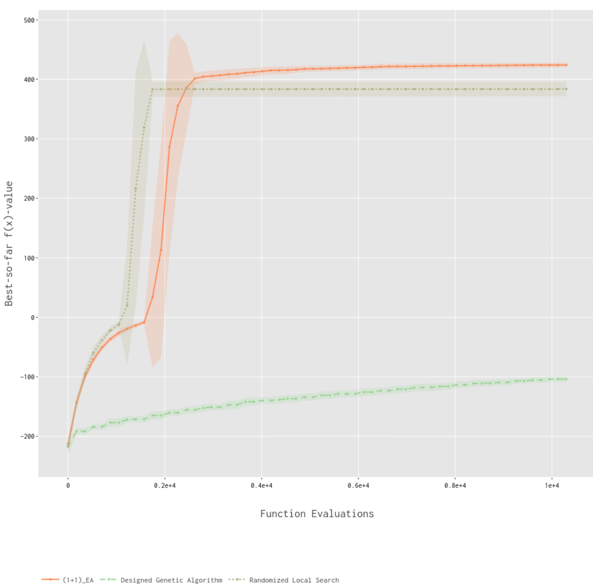
COMP SCI 3316: Assignment 3's Plots, Comparisons and Analysis

Tatiana, Miki, Hai, An and Seth

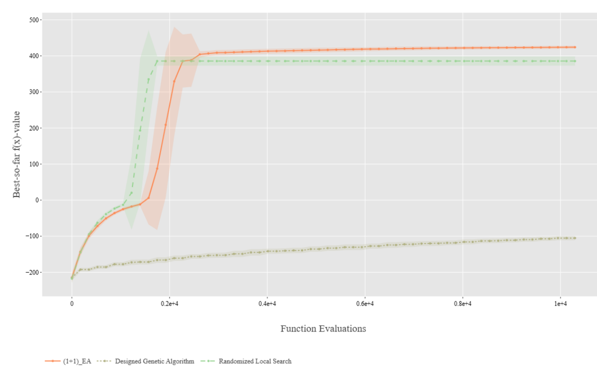
November 5, 2025

Exercise 1

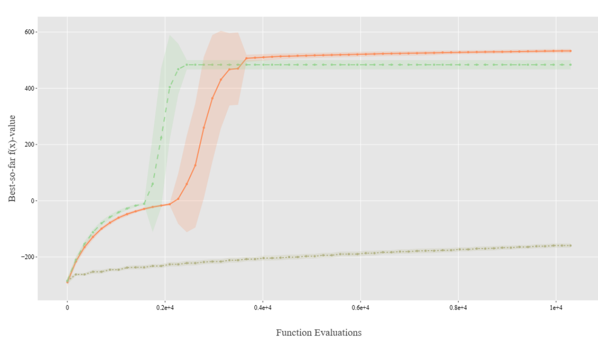
1. Fixed-budget Plots and Analysis



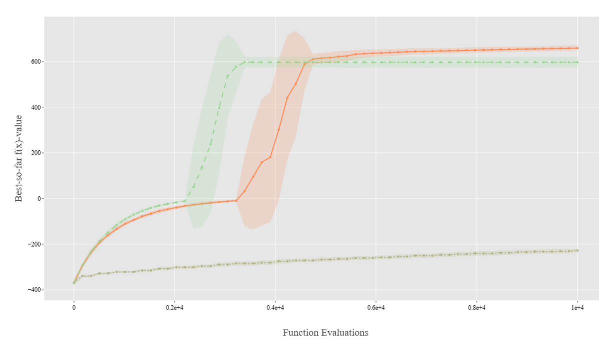
(a) Instance 2100



(b) Instance 2101



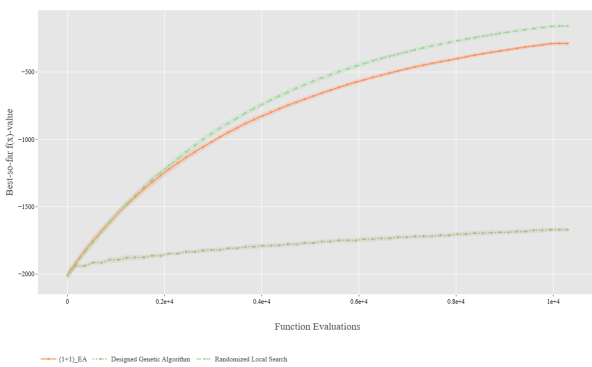
(c) Instance 2102



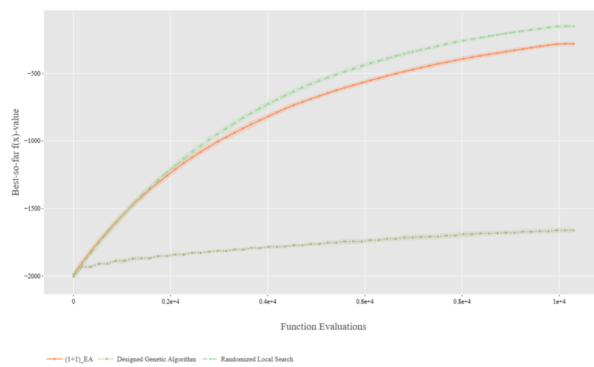
(d) Instance 2103

Figure 1: Fixed budget plots for Exercise 1 on Max Coverage instances.

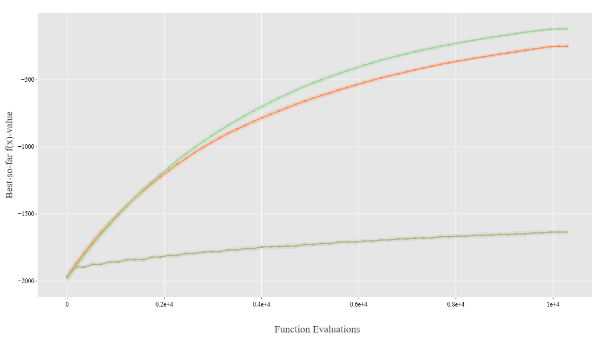
Analysis: The (1+1) EA algorithm shows rapid initial progress across all problem instances, followed by a plateau as it converges to local optima. For Max Coverage, all graphs show an initial exploration phase in the infeasible region. This phase is curved, showing a drop off in efficiency as the mutations continue. After the algorithm escapes the infeasible region, it rapidly converges upward and shortly after plateaus, signifying that it has converged to a local optimum. These three phases indicate that the algorithm has strong local exploitation abilities after feasibility is achieved, but weak exploration abilities as it tries to find feasible solutions to the problem. The plateau indicates that the algorithm tends to stagnate near a local optimum. For Max Influence, the algorithm behaves differently. The mean performance curve rises gradually but remains largely within the infeasible region, suggesting that the algorithm spends a significant portion of its budget exploring unproductive areas. The high standard deviation between runs confirms inconsistent performance, with some runs escaping the infeasible region later than others. This indicates low adaptability when facing complex, constraint-heavy fitness landscapes.. For Pack While Travel, two of the problem instances (Figures 3 - a, b) exhibit a similar pattern as Max Coverage, rapid ascent followed by early plateau. However, one instance shows more gradual improvement (Figure 3 - c), indicating that the algorithm is capable of finding a productive search path, but will generally become trapped in infeasible or suboptimal regions. Overall, the mutation-only approach allows efficient local exploitation but lacks the global search capability to consistently escape infeasible regions, making it unsuitable for complex or constrained problems due to its inconsistency.



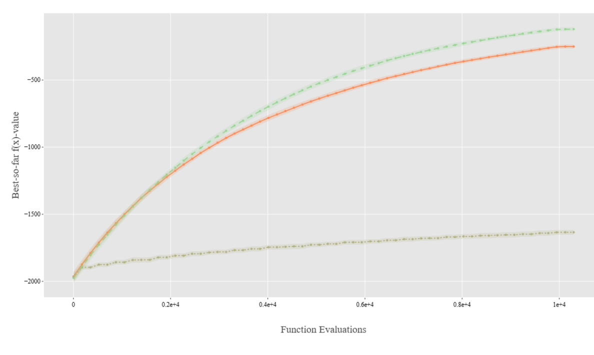
(a) Instance 2100



(b) Instance 2101



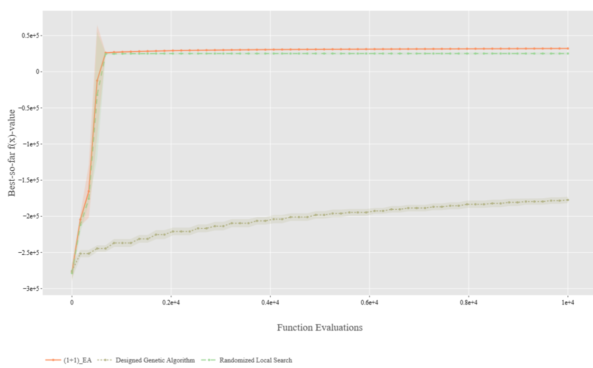
(c) Instance 2102



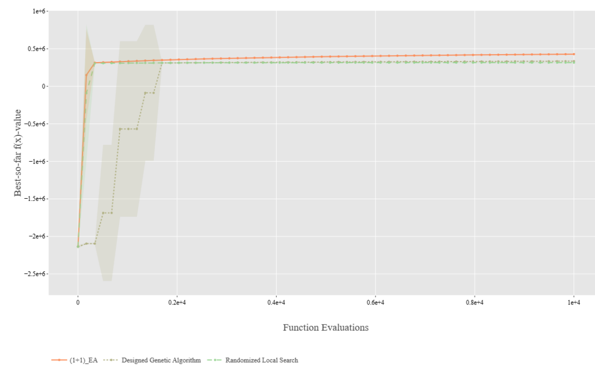
(d) Instance 2103

Figure 2: Fixed budget plots for Exercise 1 on Max Influence instances.

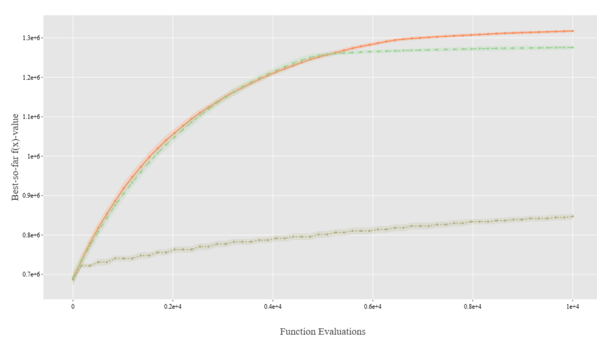
Analysis: The designed GA algorithm demonstrates slow and steady improvement across all three problems, with fluctuating yet progressive behaviour. The stochastic, near-linear growth pattern reflects the algorithm's population diversity and the combined effect of crossover and mutation operators. Unlike other algorithms discussed, it does not plateau early. However, it spends a prolonged period within the infeasible region, particularly in Max Influence and Pack While Travel. Based on the observed trend, it is likely that the algorithm would eventually converge towards feasible solutions given a larger evaluation budget In the case demonstrated by Figure 1 (all plots), the algorithm displays rapid ascent followed by a plateau. This could indicate the algorithm's attempt to stabilize after escaping the infeasible region rather than converging to a local optimum. To summarise, the Designed GA's ability to maintain diversity through crossover allows for continued exploration, but at the cost of slower convergence and increased overhead. This means that it performs more effectively on complex, multimodal problems with large infeasible spaces, such as Max Influence, while losing efficiency for simpler, unimodal problems, such as Max Coverage.



(a) Instance 2300



(b) Instance 2301



(c) Instance 2302

Figure 3: Fixed budget plots for Pack While Travel instances.

Analysis: RLS exhibits very similar behaviour to (1+1) EA, with three behavioural phases: a struggle to escape the infeasible region, sharp upwards progress and then a plateau. This behaviour results from its simple one-bit mutation mechanism, allowing for efficient local optimisation but limited exploration. Across all problem instances, RLS tends to remain trapped within the infeasible region for long stretches, most noticeably seen in Max Influence and Pack While Travel, where the landscape is complex and constrained. Although it can efficiently exploit local regions of the search space, it lacks diversity mechanics that are required to navigate toward feasible or globally optimal areas. Consequently, it performs well on simple, unimodal problems and struggles to maintain progress or adapt efficiently on more complex, multimodal problems.

2. Comparison of Algorithms

Comparing all three algorithms, there is a clear distinction between exploration and exploitation abilities. Both the (1+1) EA and RLS display rapid early progress, showing strong local exploitation once feasibility is achieved. However, they also show frequent stagnation and prolonged periods within the infeasible region, showing limited ability to recover or explore new areas of the search space. The deterministic mutation-based search strategies favour exploitation over exploration, making them suitable for simple, unimodal problems such as Max Coverage, where feasible regions are easier to locate and optimise. In contrast, their performance deteriorates on complex or constrained problems such as Max Influence or Pack While Travel, where escaping infeasibility and navigating multimodal landscapes require greater diversity and adaptibility The Designed GA algorithm demonstrates greater exploration capacity due to its population-based approach. It maintains progress for longer, though at the cost of slower convergence and higher computational complexity. While it also suffers from extended periods stuck in the infeasible regions, its stochastic and diverse population dynamics allow it to eventually transition towards feasible solutions, making it more suitable for multimodal and constraint-heavy problems. To conclude, none of the algorithms consistently achieve strong results due to frequent stagnation in infeasible zones. However, the Designed GA algorithm offers the best balance between exploration and exploitation in complex landscapes, while the (1+1) EA and RLS algorithms remain efficient but narrow in scope, making them better suited for simpler, more defined search spaces.

Exercise 2

1. Recalling Basics of Submodularity

We can define the *submodular and monotone* using the notation from graph theory. Given a graph $G = (V, E)$. Our search space is the vertex space V. The submodular function can be defined as:

$$f : 2^V \to \mathbb{R}$$

$A \subseteq B \subseteq 2^V$ are sets of vertices, it is monotonic when:

$$A \subseteq B \implies f(A) \leq f(B).$$

it is *submodular*, given that $v \in V \setminus B$ when:

$$f(B \cup \{v\}) - f(B) \leq f(A \cup \{v\}) - f(A).$$

Notations

We denote $\boldsymbol{v} \subseteq V$ be a set of vertices, the relevant notation will be written in bold with a lowercase letter. The bold, lowercase letter indicate a set of values.

Uniform k-constraint cost

The *uniform cost* function $c : 2^V \to \mathbb{R}_{\geq 0}$ is a linear function with weight $\boldsymbol{w}$ and defined as:

$$c(\boldsymbol{v}) = \sum_{i}^{n} w_i \cdot c(v_i) \leq k \in \mathbb{R},$$
$$\text{where } w_i = 1,\ c(v_i) = 1$$

Notice that $c(v_i)$ is the cost of the vertex being chosen in a set not the cost of its out-going edge. The value for $\boldsymbol{w} = \mathbf{1}$ and $c(v_i) = 1$ is given.

Overall, the fitness of a search point is a 2D vector given by. Therefore this is a bi-objective optimisation problem, defined as

$$\begin{aligned}
&\arg\max_{\boldsymbol{v} \subseteq V} \mathsf{F}(\boldsymbol{v}) = (f'(\boldsymbol{v}), -c(\boldsymbol{v})) \\
&\qquad \text{subject to} \\
&\qquad\quad c(\boldsymbol{v}) \leq 10 \\
&\qquad\quad f'(\boldsymbol{v}) = \begin{cases} f(\boldsymbol{v}), & c(\boldsymbol{v}) \leq k = 10 \\ (k = 10) - f(\boldsymbol{v}) \quad , & c(\boldsymbol{v}) > k = 10 \end{cases}
\end{aligned}$$

Dominance Formulation Used

For any two given solution sets $\boldsymbol{v} \succ \boldsymbol{u}$. We have that

$$\begin{aligned}
&\mathsf{F}(\boldsymbol{v}) \succ \mathsf{F}(\boldsymbol{u}) \\
\iff &\begin{cases} f'(\boldsymbol{v}) \geq f'(\boldsymbol{u}) \\ c(\boldsymbol{v}) \leq c(\boldsymbol{u}) \\ \mathsf{F}(\boldsymbol{v}) \neq \mathsf{F}(\boldsymbol{u}) \end{cases}
\end{aligned}$$

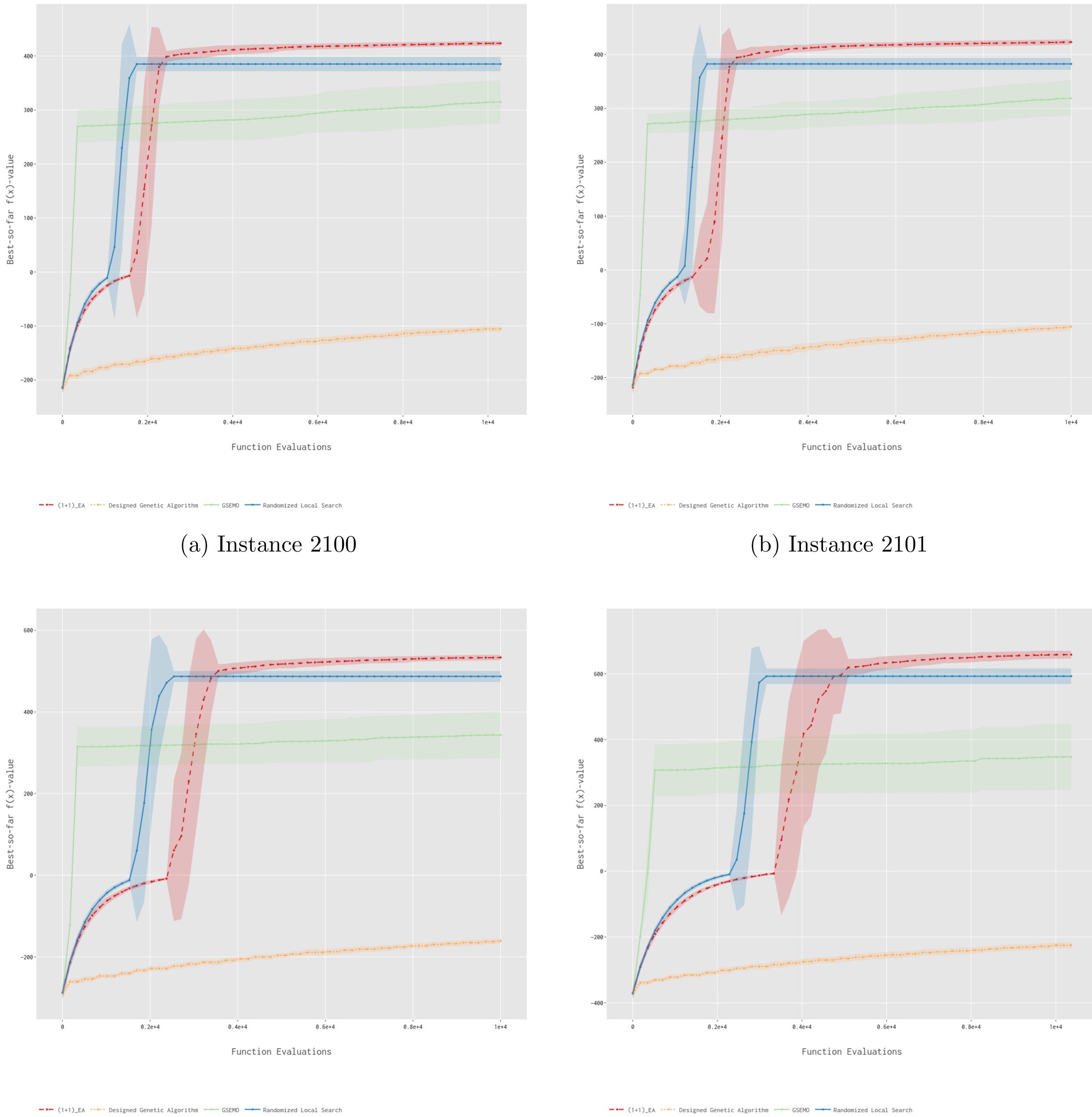
Thus, we can say that

$$\mathsf{F}(\boldsymbol{v}) \succ \mathsf{F}(\boldsymbol{u}) \iff \begin{cases} f'(\boldsymbol{v}) \geq f'(\boldsymbol{u}) \\ c(\boldsymbol{v}) < c(\boldsymbol{u}) \end{cases}$$

And also,

$$\mathsf{F}(\boldsymbol{v}) \succeq \mathsf{F}(\boldsymbol{u}) \iff \begin{cases} f'(\boldsymbol{v}) \geq f'(\boldsymbol{u}) \\ c(\boldsymbol{v}) \leq c(\boldsymbol{u}) \end{cases}$$

2. Fixed-Budget Plots and Analysis



(a) Instance 2100

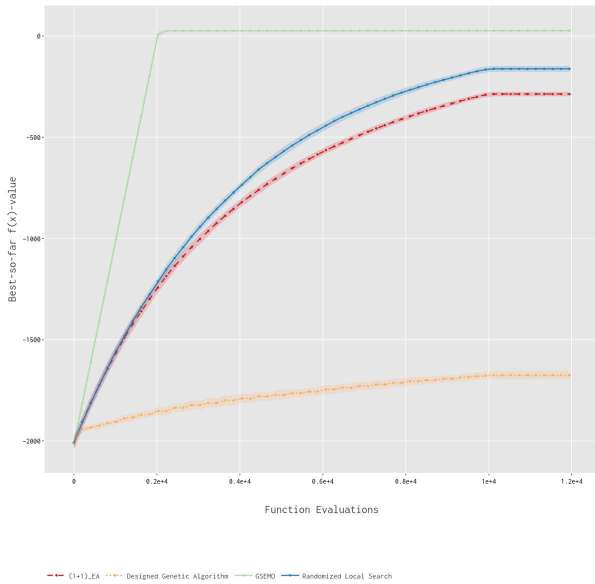
(b) Instance 2101

(c) Instance 2102

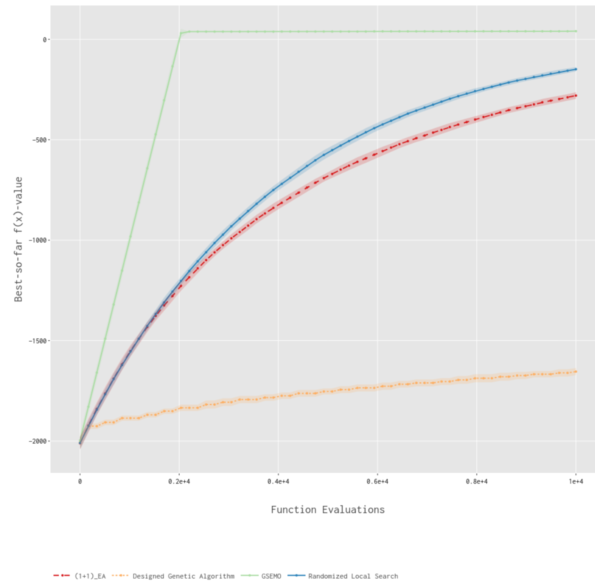
(d) Instance 2103

Figure 4: Fixed budget plots for Max Coverage instances.

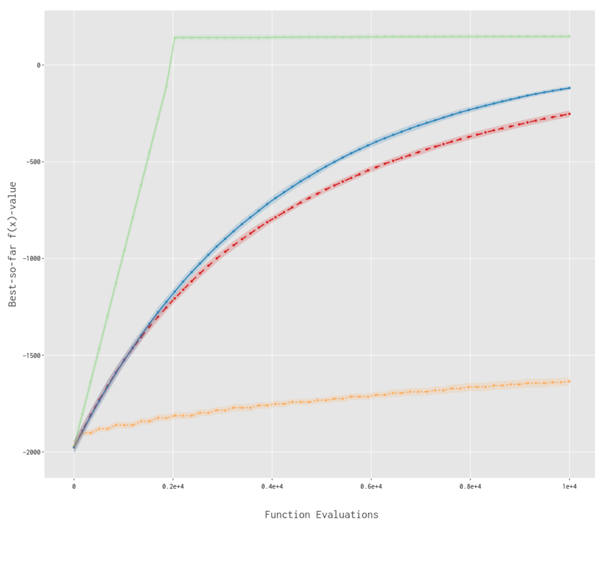
Analysis: When solving the Max Coverage problem instances, It is consistently true that the (1+1) EA, RLS, then GSEMO perform the best in terms of fitness. The GA is consistently outperformed, unable to find a feasible positive solution. The GSEMO is initially the fastest to reach high fitness values, likely due to the highly diverse set of non-dominated Pareto solutions it considers. However, it reaches a sharp plateau at an early stage after approximately 350 function evaluations. This, with the sudden height of standard deviation indicates early exhaustion of the mutation mechanism for diversity, at which point the algorithm prematurely converges and becomes trapped in varying local optima. The continual attempt to explore these regions yields a small approximately 30 to 50 point change in fitness after the initial plateau. In contrast, the (1+1) EA and RLS reach their high valued solutions with slower but productive convergences. Both algorithms gradually progress before stagnating briefly with a sudden increase in standard deviation, where they then steeply reach high fitness as standard deviation decreases. Standard deviation then increases as both algorithms plateau and find global convergence. The steep increase marks rapid progress which likely results from the eventual accumulation then exploitation of high fitness individuals after the gradual increase of random unguided exploration. While the trends between the (1+1) EA and RLS algorithms are similar, RLS is consistently slower, also achieving a lower final fitness. This may be attributed to the smaller marginal gains RLS makes with its single bit-flips compared to (1+1) EA's ability to flip multiple bits. The trend for the (1+1) EA appears with inconsistencies for the F2103 problem instance. Here, during the steep fitness increase, multiple stagnations occur as the plateau is approached with high standard deviation in the region. It is likely the rare flipping of multiple-bits was required to allow some infrequent escapes from the multiple local optima present at that stage. The designed GA exhibits a gradual asymptotic approach to its global optimum. The GA implemented is ineffective for the given problem. Its selection pressure does not provide sufficient diversity for exploration of the search space, and its mutation rate is too low to allow exploitation of existing solutions. This manifests as the early convergence with no trend change throughout function evaluations. Overall, the Max Coverage problem is best solved using bit-flipping mechanisms and is likely not receptive to population-based algorithms. The GSEMO, while achieving fast early progress, was unable to escape local optima to reach the greater fitness values the (1+1) EA and RLS algorithms were able to achieve.



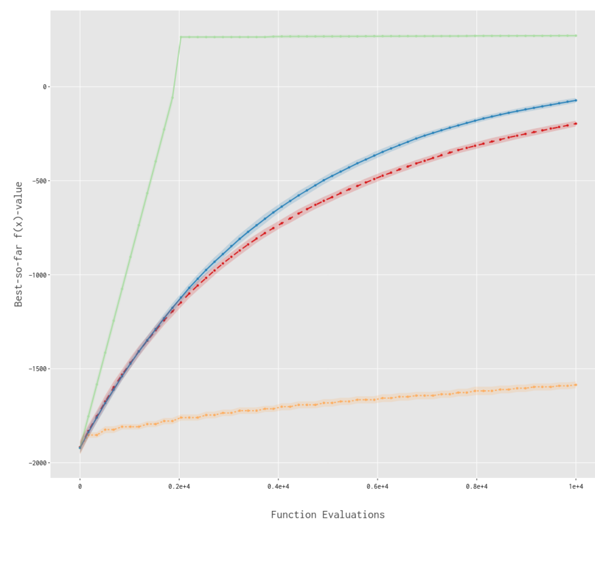
(a) Instance 2100



(b) Instance 2101



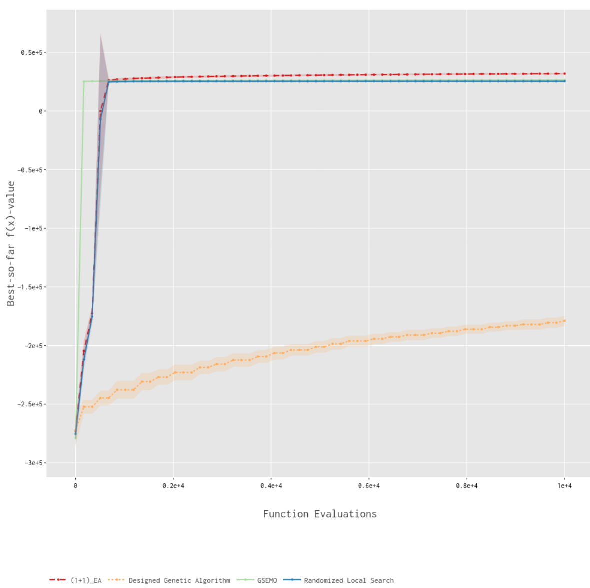
(c) Instance 2102



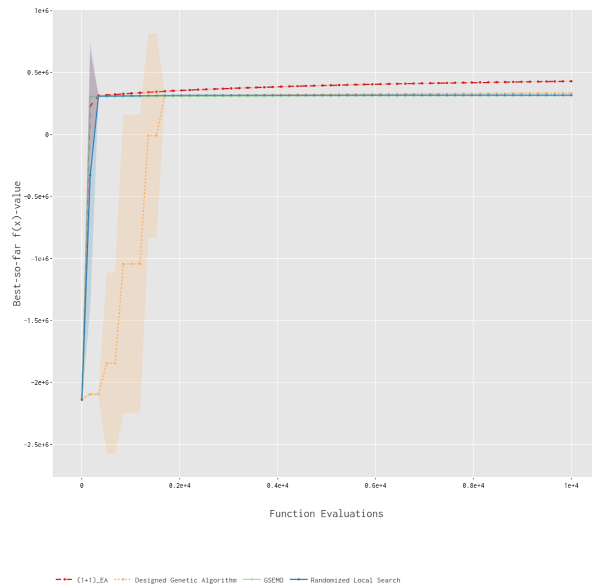
(d) Instance 2103

Figure 5: Fixed budget plots for Max Influence.

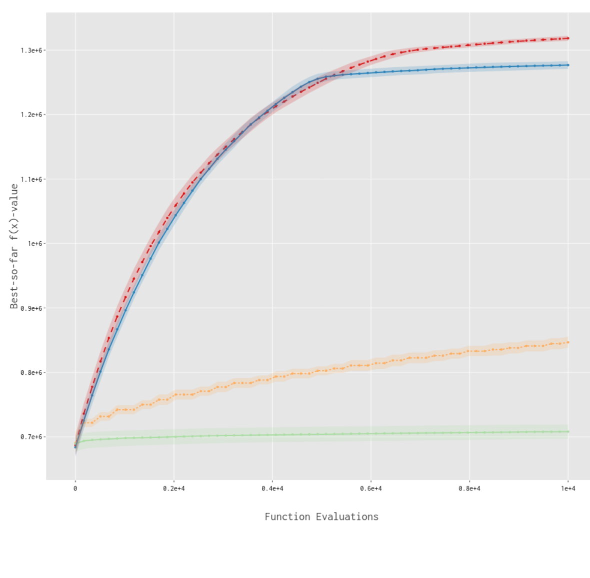
Analysis: Solving the Max Influence problem, the GA is again unable to achieve a feasible positive fitness value. However, for this problem, RLS and (1+1) EA also do not achieve feasible fitness values; only GSEMO achieves this, acting as the most effective algorithm for this problem set. The GSEMO follows a similar trend as with the Max Coverage problems. It exhibits a sharp linear progress, quickly achieving the highest fitness before its algorithmic counterparts. It achieves a constant value, at which point it plateaus at its final optimum. The steep increase can be largely attributed to the diversity achieved in mutation steps early on in the function evaluations. Also, the greedy construction of the Pareto set, and storage and use of partial solutions in this set are advantageous for allowing search space exploration. This is, as opposed to the immediate discarding of solutions for better ones as in the other algorithms. The GSEMO algorithm's consideration of both objectives when constructing solutions is also more suitable practice for the given multi-objective problem. However, while this algorithm efficiently reaches a plateau, this early convergence signals an inability to exploit solutions for higher fitness values. The (1+1) EA, RLS, and GA follow a similar trend of concave asymptotic increase towards their global convergence. However, the (1+1) EA and RLS achieve much higher fitness than the GA. The graphs each smoothly increase with a decreasing slope that eventuates towards a plateau at some constant fitness optimum; this is seen within the 10,000 function evaluation budget for problem instance F2200. The initial rapid improvement in fitness is due to achievements by each algorithm's respective diversity operators. However, for the GA, the high slope or high fitness growth region is short-lived, indicating an inability to accommodate exploration. Ultimately, being unsuccessful for any future exploitation attempts. The nature of the Max Influence problem influences these trends. Since it finds its solution using a probabilistic influence spread model, the problem generates smooth graphs whose rate of growth decreases and eventually reaches a saturation point. That is, its submodular nature causes the decrease in marginal gains made as the evaluations progress and, as gain approaches 0, a plateau forms. To contrast, the Max Coverage problem earlier applies discrete additions for the solution spread, resulting in the often stepwise, frequently stagnating nature of its graphs. Overall, the GSEMO is highly efficient and effective as its properties make it highly suited for a bi-objective submodular problem such as Max Influence. The other three algorithms were not able to perform as successfully. However, their graphs clearly exhibit the behaviours of the Max Influence function as it forms a solution, and, generally, that of a submodular problem.



(a) Instance 2100



(b) Instance 2101



(c) Instance 2102

Figure 6: Fixed budget plots for Pack While Travel Instances.

Analysis: Unlike the Max Coverage and Max Influence problem instances, the graphs generated by each algorithm for each Pack While Travel problem instance vary greatly. For instance F2300, the (1+1) EA, GSEMO and RLS reach near identical optima, while the GA maintains low performance. The first three algorithms form graphs of similar trends: they progress quickly towards a high fitness value after less than 700 fitness evaluations, before stagnating to a plateau. The GSEMO is the fastest to reach a plateau, but the (1+1) EA achieves a higher final fitness value. The GSEMO, as noted for Max Influence, is able to effectively explore diverse solutions through its mutation operation and use of non-dominated solutions. For the RLS and (1+1) EA during their steep rise towards the plateau, there is brief stagnation. This stagnation is a likely result of initial accumulation of fit solutions reaching saturation, before exploitation via bit-flip mutations begins to become effective; thus, causing the continued rise to the plateau. Standard deviations for these graphs coincide and mark brief divergence into various local optima, at which point the algorithms find global convergence, thus initiating the plateaus. As previously observed with other problem instances, the GA's graph asymptotically approaches a plateau. The algorithm's attempt at exploration is evident in the inconsistent stepwise stagnations at the curve start. However, it is unable to meaningfully escape towards more optimal solutions, resulting in its poor performance. For instance F2301, all the algorithms achieve high fitness solutions, and do so following similar trends. The GSEMO, RLS and (1+1) EA follow the same trends as discussed for F2300. However, they each reach the plateau after a similar number of function evaluations, with the GSEMO being marginally faster. The GA follows the general trend of a fast rise to its plateau, however, its highly stochastic nature in this process is also evident. It is relatively slower to reach its plateau due to a set of stepwise stagnations during the rise, at which standard deviation is highest before a sharp decrease along the plateau. The high standard deviation demarcates both high stochasticity and the diverse local optima found and explored before global convergence. For instance F2302, every graph appears as a concave down graph. This is similar to the Max Influence graphs: the graphs smoothly increase with a decreasing slope, marking a slowing rate of fitness increase per gain, eventuating towards a plateau. The (1+1) EA and RLS perform best, and the GA exhibits a slow unproductive increase. However, the GSEMO performs the worst, forming a nearly horizontal graph with insignificant gains. It is likely that, for the non-monotone Pack While Travel problem, the dominance relation is incorrectly defined, causing the addition of low-quality solutions to the Pareto set. Also, the bits for a Pack While Travel problem are considered highly epistatic, of which the local bit-flip mutations performed may disturb; thus, leading to lower quality solutions and stagnation. Observing the trade-off plots generated for this instance, the GSEMO exists with a heavily populated Pareto front. It is likely a large number of low-quality solutions were added, the application of this algorithm to a non-monotone function preventing sufficient exploration and discovery of good solutions. The frequent function evaluations on the larger Pareto set can slow the solution's convergence time, resulting in the poor performance. Overall, each algorithm's behaviour changes across problem instances for the Pack While Travel problem, indicating each algorithm is highly sensitive to changes in landscapes. However, the (1+1) EA consistently outperforms the other algorithms.

Conclusion: The GSEMO algorithm is an efficient algorithm for monotone submodular problems with an ability to achieve an optimum with relatively few function evaluations. For the Max Coverage problem, the GSEMO is outperformed by the (1+1) EA and RLS in terms of optimal fitness value but it demonstrates its affinity for fast and effective search space exploration. GSEMO also exhibits its clear superiority at solving dynamic problems when optimising the Max Influence problem. However, for the Pack While Travel problem instances, discrepancies concerning non-monotone submodular problems with varying landscapes were apparent.

3. Trade-off plots (Pareto fronts) of the first run of each problem instance

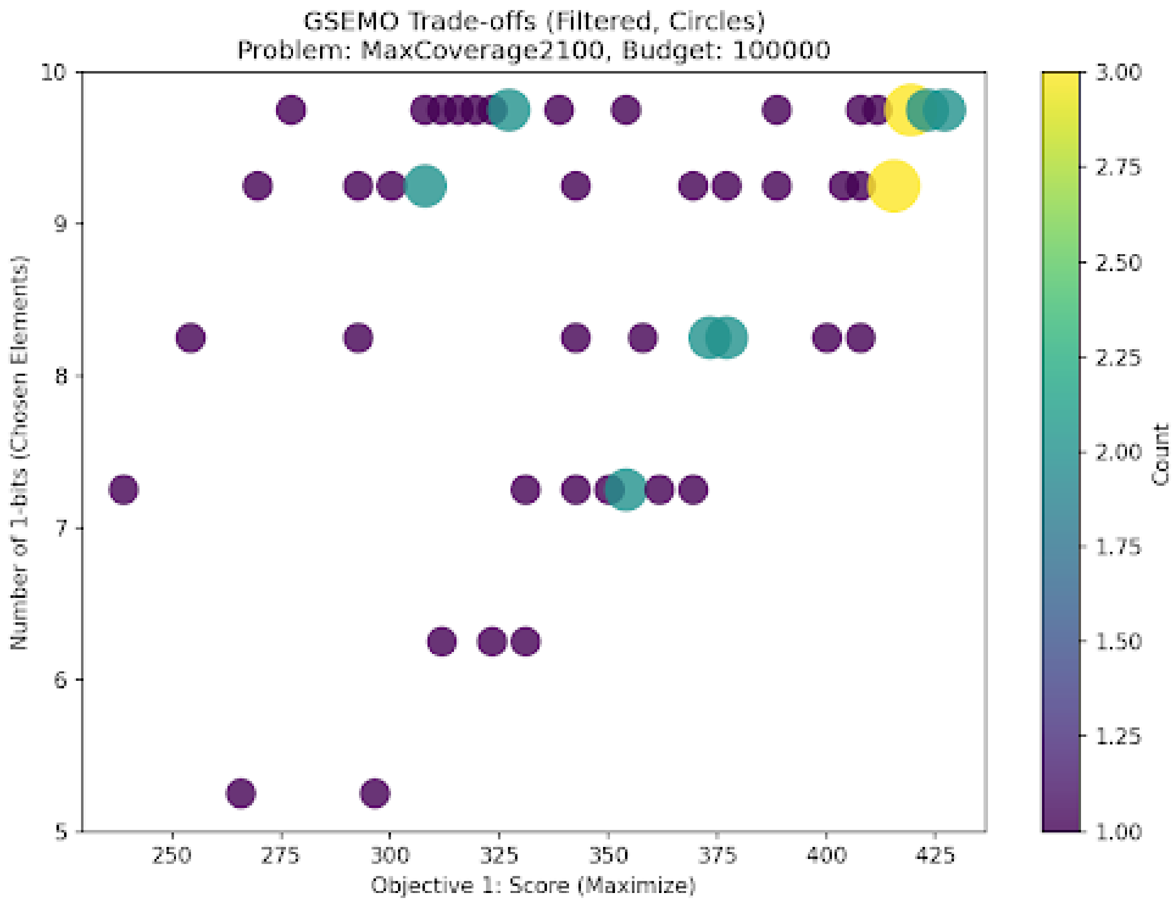


Figure 7: GSEMO tradeoff on Max Coverage 2100

The trade-off plot for GSEMO on the Max Coverage 2100 instance shows that the algorithm attains high fitness. The densest region of non-dominated solutions is in the upper-right of the plot (9–10 1-bits and fitness around 425), indicating many solutions use a large number of 1-bits. This suggests the algorithm tends to exploit the cost constraint to increase coverage. There is an overall positive trend between bit-count and fitness, fitness generally increases as the number of 1-bits grows up to the observed maximum. However, there are notable exceptions in the 7–8 1-bit region where some lower cost solutions achieve higher fitness than some more expensive solutions. Thus, while GSEMO appears biased toward high bit-counts where ranges near the constraint are more populated, some high-bit solutions still perform poorly and many low-cost solutions can deliver strong coverage. In summary, the Pareto approximation behaves as expected, maximising coverage typically requires more 1-bits, so feasible costly solutions cluster in the upper right.

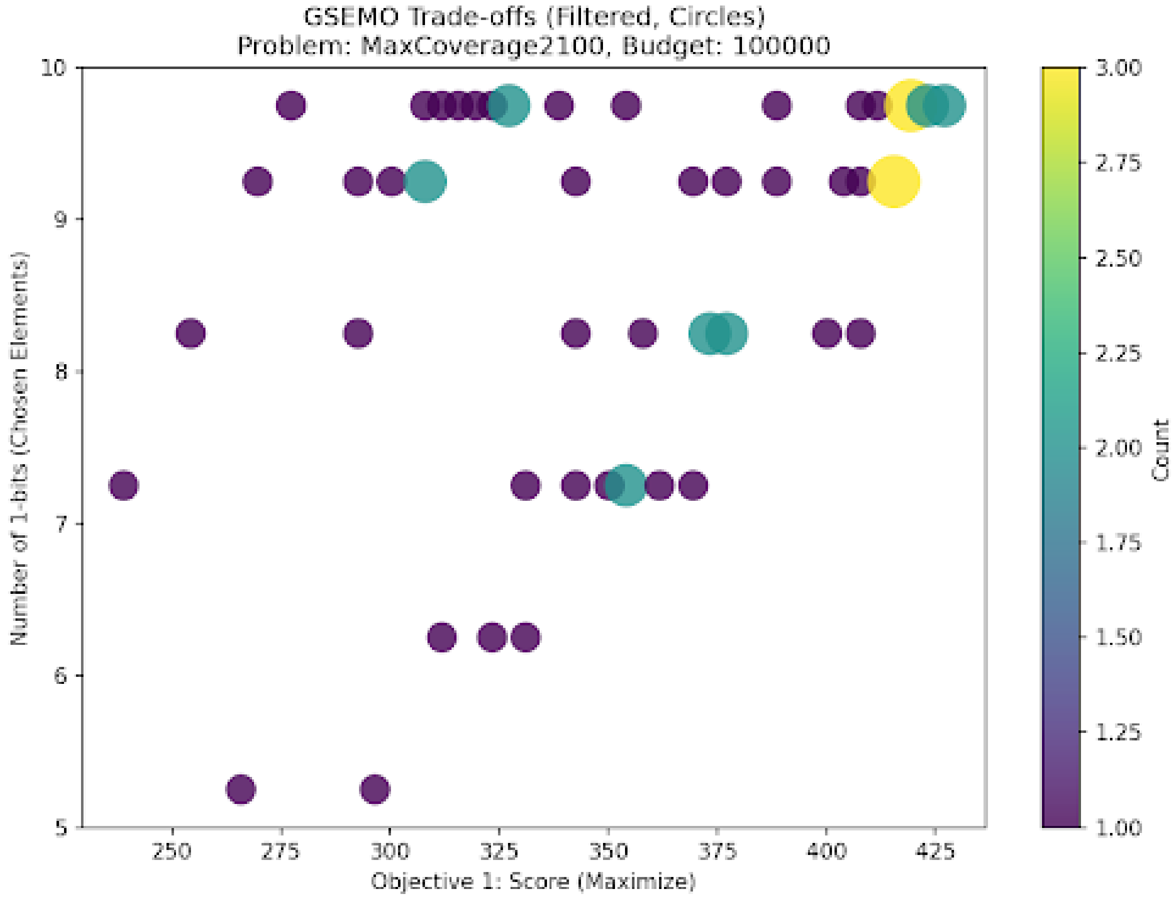


Figure 8: GSEMO tradeoff on Max Coverage 2101

The GSEMO performs very well on the Max Coverage 2101 instance. The densest region of solutions is at the highest fitness values. The plot shows a positive relationship between the number of 1-bits and fitness whereas fitness increases, the number of 1-bits generally increases. There is, however, a small region around 7–8 1-bits that performs slightly worse than the neighbouring regions, on average that region still reaches fitness values above around 325. The 9–10 1-bit region contains many tailing solutions that drop below around 325. This pattern suggests GSEMO is biased toward expensive solutions and therefore explores the cheaper region less thoroughly. The most densely populated region is between 400–425 fitness, indicating that costly solutions often pay off with higher fitness. Overall, the result matches expectations, by achieving higher coverage typically requires more 1-bits, so GSEMO tends to search and sample many high-cost solutions in this problem.

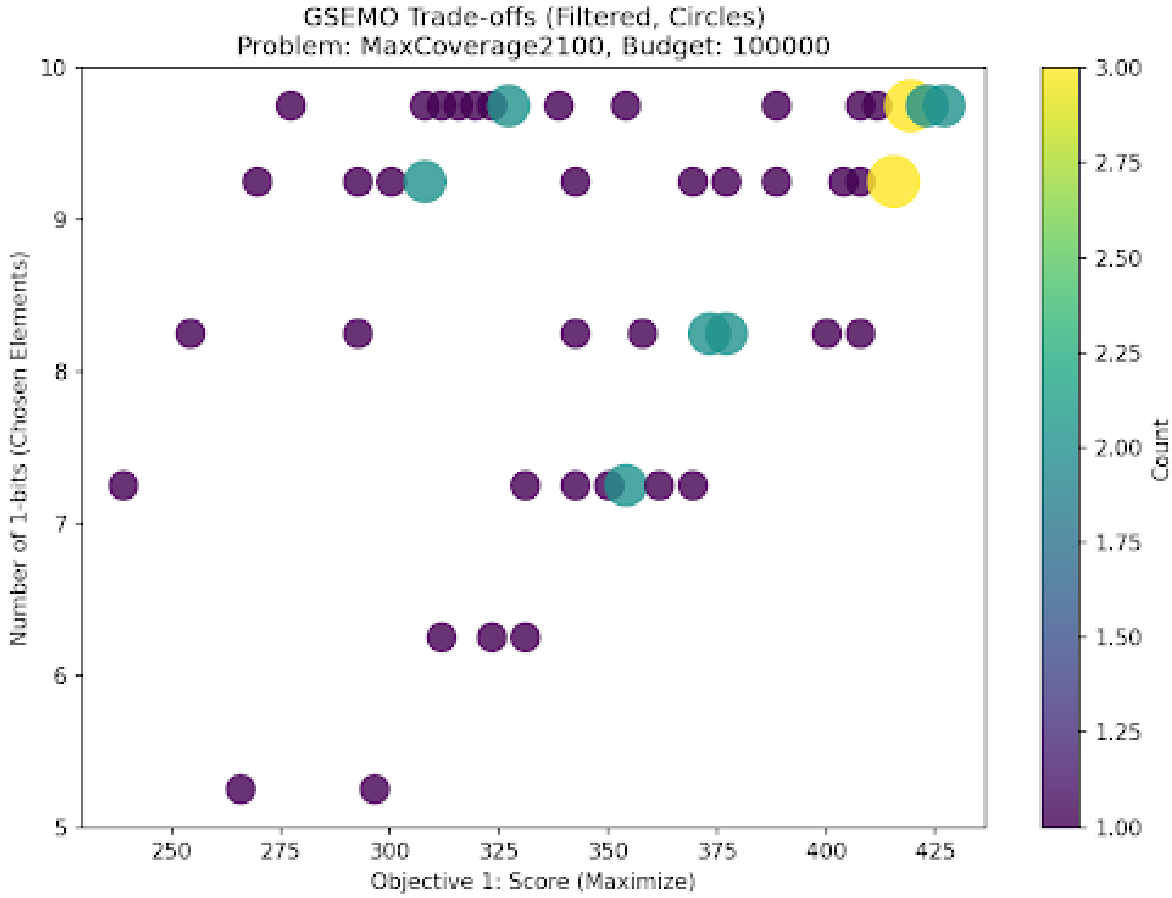


Figure 9: GSEMO tradeoff on Max Coverage 2102

The GSEMO performs well on the Max Coverage 2102 problem. The densest region is the one with the highest fitness and highest cost, in the top-right of the graph. The plot shows a positive relationship between number of 1-bits and fitness, on average, more 1-bits yield higher fitness. In this problem that relationship is particularly consistent, more costly solutions tend to perform better. The 7–8 1-bit region gives poor results, with most fitness values below 400. By contrast, the 9 1-bit region attains values around 450, while the 10-bit cluster lies roughly in the 450–500 range. If GSEMO is biased toward expensive solutions, that bias is advantageous here because this instance strongly rewards higher cost.Specifically, the 9-bit region is only about 50 fitness points worse than the 10-bit region, whose densest part is around 470. This indicates that the 9-bit region can also yield very good results. Overall, the Pareto approximation for this instance rewards costly solutions, and GSEMO correctly explores these regions and yields good results while avoiding the cheap region that produces poorer fitness.

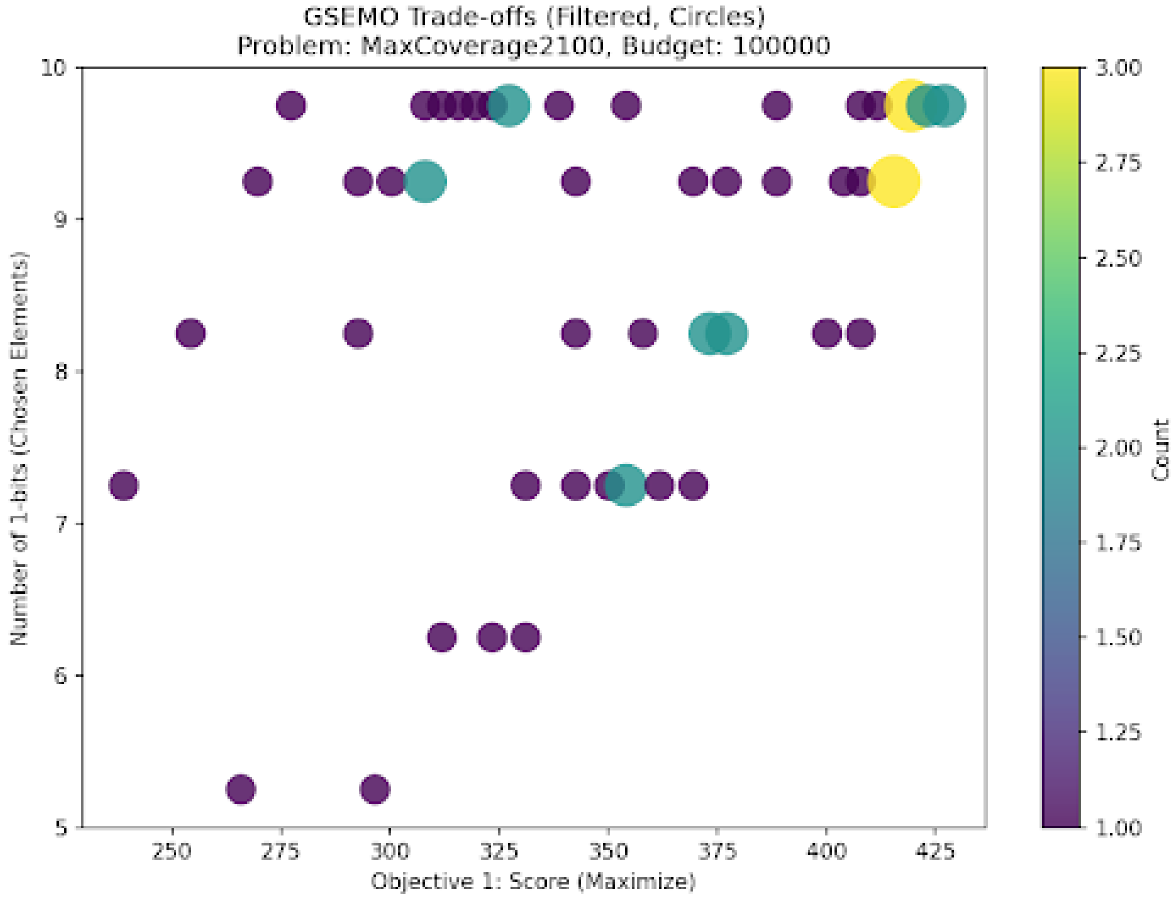


Figure 10: GSEMO tradeoff on Max Coverage 2103

The graph shows an interesting inverted triangle shape in objective space. As the number of 1-bits increases, fitness also increases, but only gradually. The solution sets for each bit-count from 6 to 10 become denser and fan outwards. This implies that higher costs tend to be associated with higher fitness values. However, the number of solutions that achieve strictly higher fitness at higher cost is small once we compare with the previous (cheaper) regions. For example, in the 10-bit region, solutions that clearly outperform the 9-bit region appear only after about fitness = 550, below that threshold the 9 and 10 1-bits solutions largely overlap. Thus, for this instance, if GSEMO is biased toward expensive solutions it may struggle to find substantially better solutions, because cheaper solutions are only marginally worse than the more expensive ones. Overall, the fan shape shows an approximately linear increase between number of 1-bits and fitness, the more 1-bits generally give higher fitness, but the improvement is small as you move to higher costs.

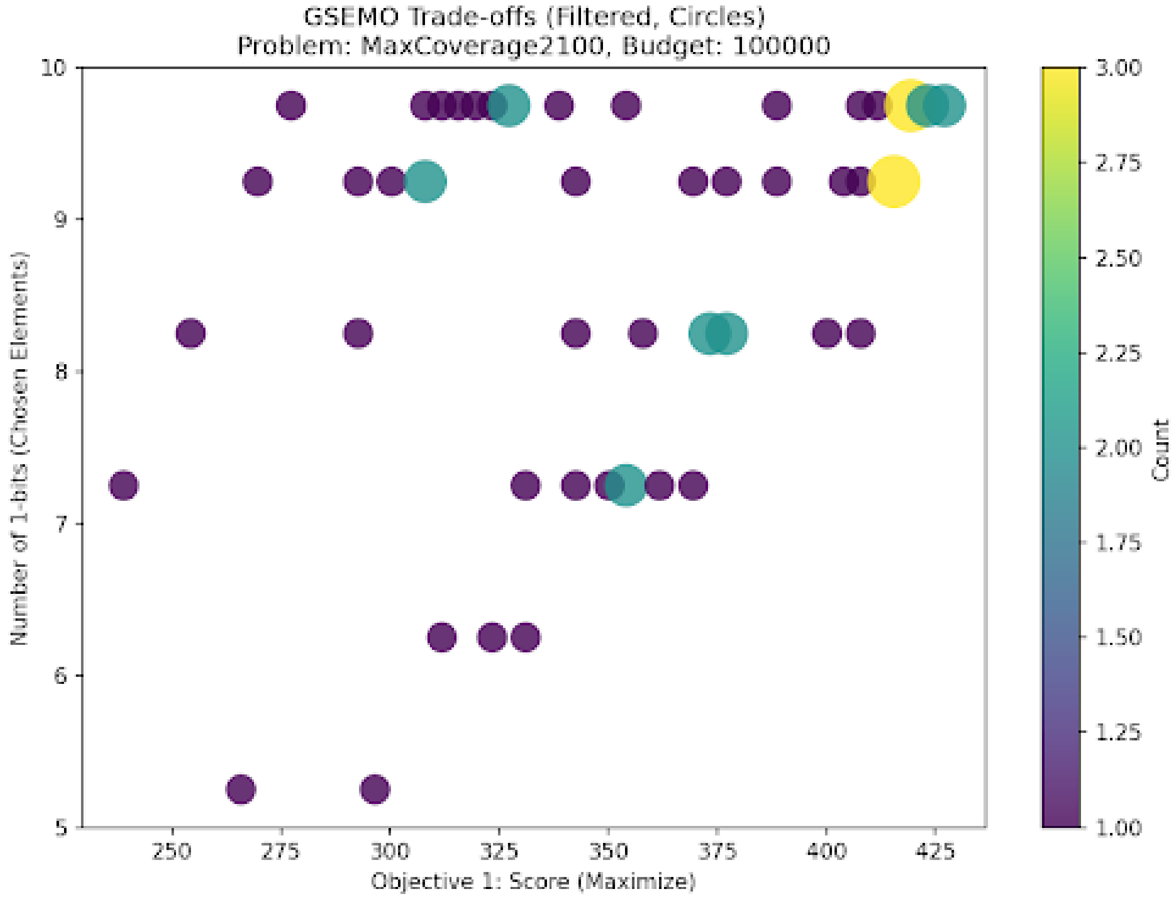


Figure 11: GSEMO tradeoff on Max Coverage 2200

The tradeoff plot for GSEMO shows that the algorithm does not perform well on Max Influence 2200. Many expensive solutions still yield very low fitness and cluster at the left side of the fitness axis. This highlights a weakness of the vanilla GSEMO even with our simple single-bit repair operator. GSEMO struggles in this instance because the weak repair fails to recover feasible, high-quality solutions, so the Pareto set remains dominated by low-fitness points. The trade-off does not show a clear correlation between the number of 1-bits and fitness. Without a stronger repair mechanism, many solutions only start to leave the infeasible region and slowly increase in fitness once they become feasible. Overall, this indicates that GSEMO does not have enough evaluations to escape the low-fitness zone.

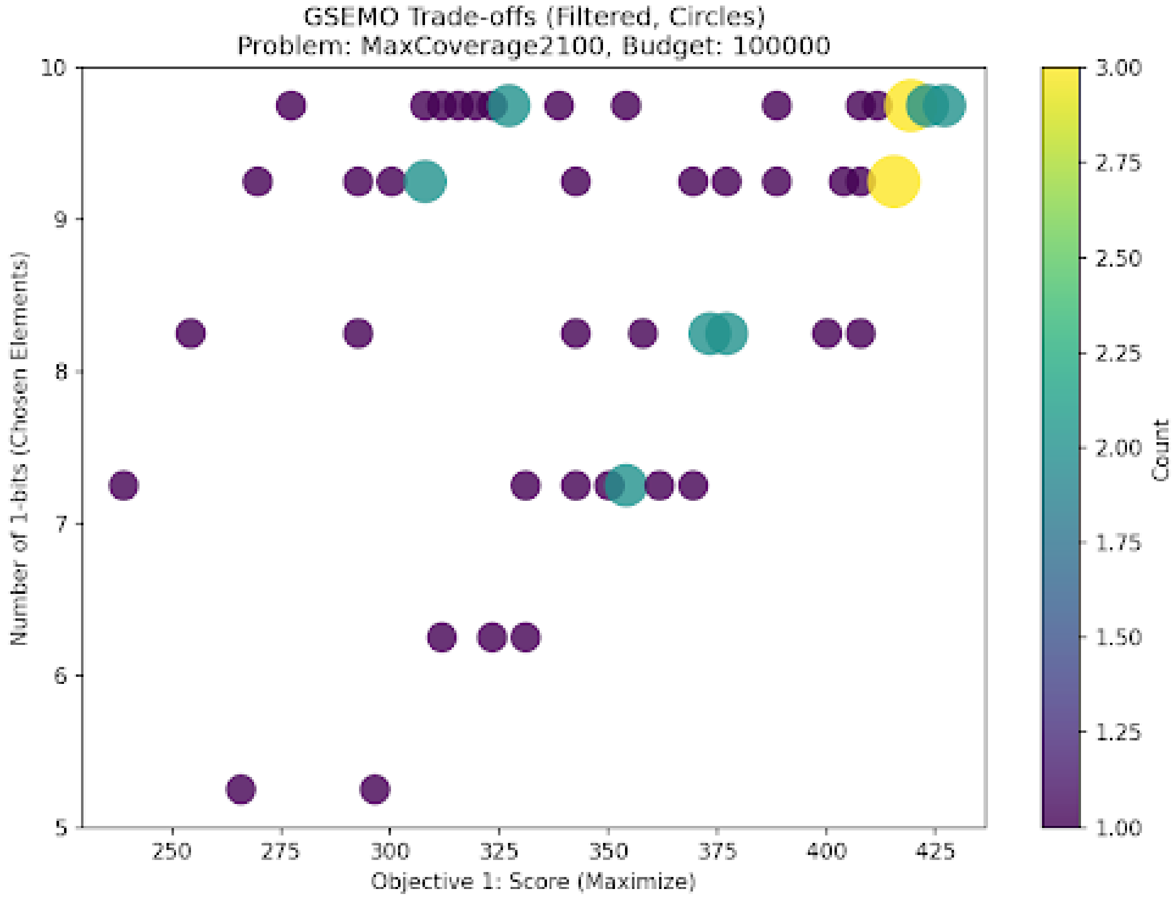


Figure 12: GSEMO tradeoff on Max Coverage 2201

GSEMO found no feasible solutions on the Max Influence 2201 instance. Fitness values are more spread out, but there is a dense cluster around 35–40. This pattern indicates that GSEMO is improving objective value while remaining infeasible by attempting to raise fitness under the cost bound of 20. There is little evidence that the algorithm explores the feasible region: solutions with cost 19 still have low fitness around 33 with just a single solution. Overall, GSEMO appears stuck in the infeasible region and has not yet escaped it.

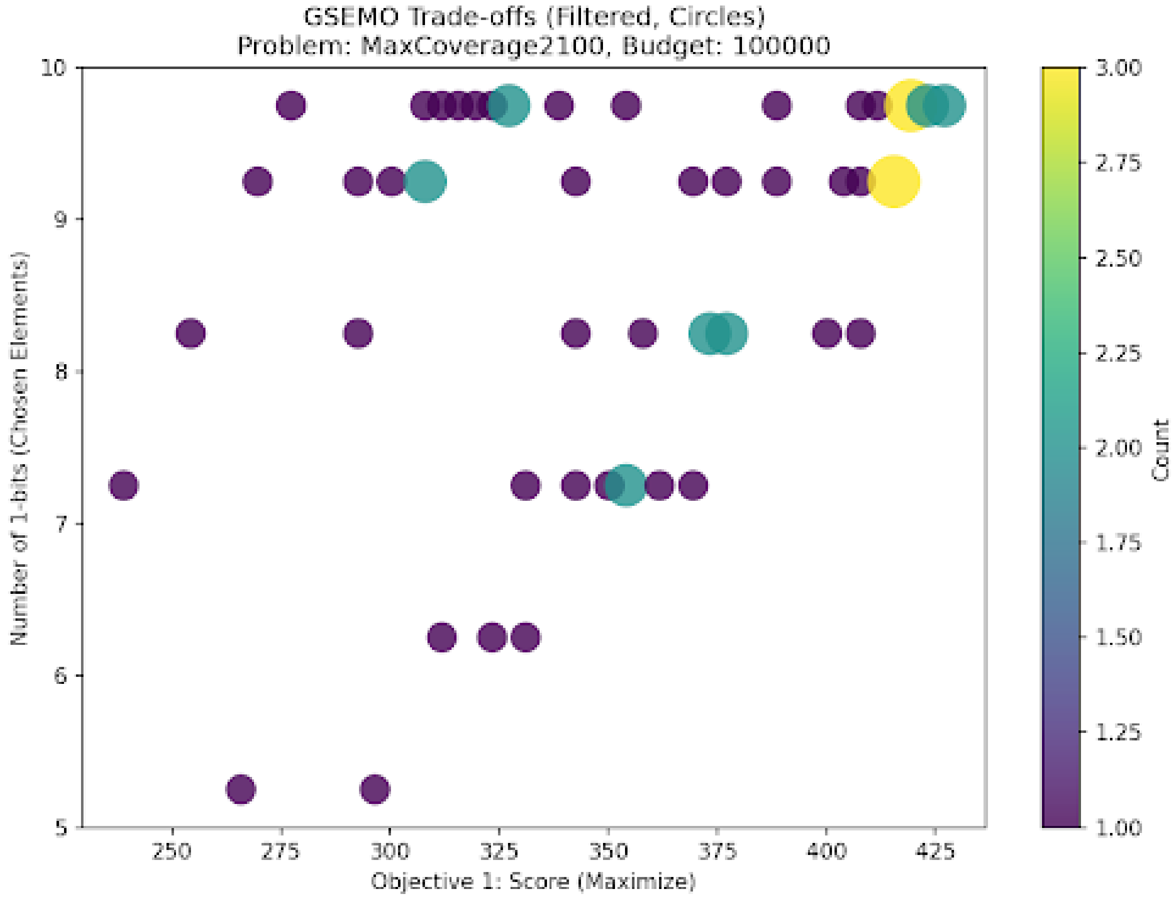


Figure 13: GSEMO tradeoff on Max Coverage 2202

GSEMO on the Max Influence 2202 instance shows interesting behaviour. It is clearly stuck in the infeasible zone. However, GSEMO still discovers high fitness values across different cost regions, and it produces very dense sets of solutions at low fitness values. This suggests the algorithm is attempting to validate itself and escape the infeasible zone, since observed costs are gradually decreasing from 50 down to 48. For instance, at cost 50 the solutions span roughly 130–188 fitness. The high density in the low-fitness infeasible region therefore indicates that GSEMO is trying to correct itself.

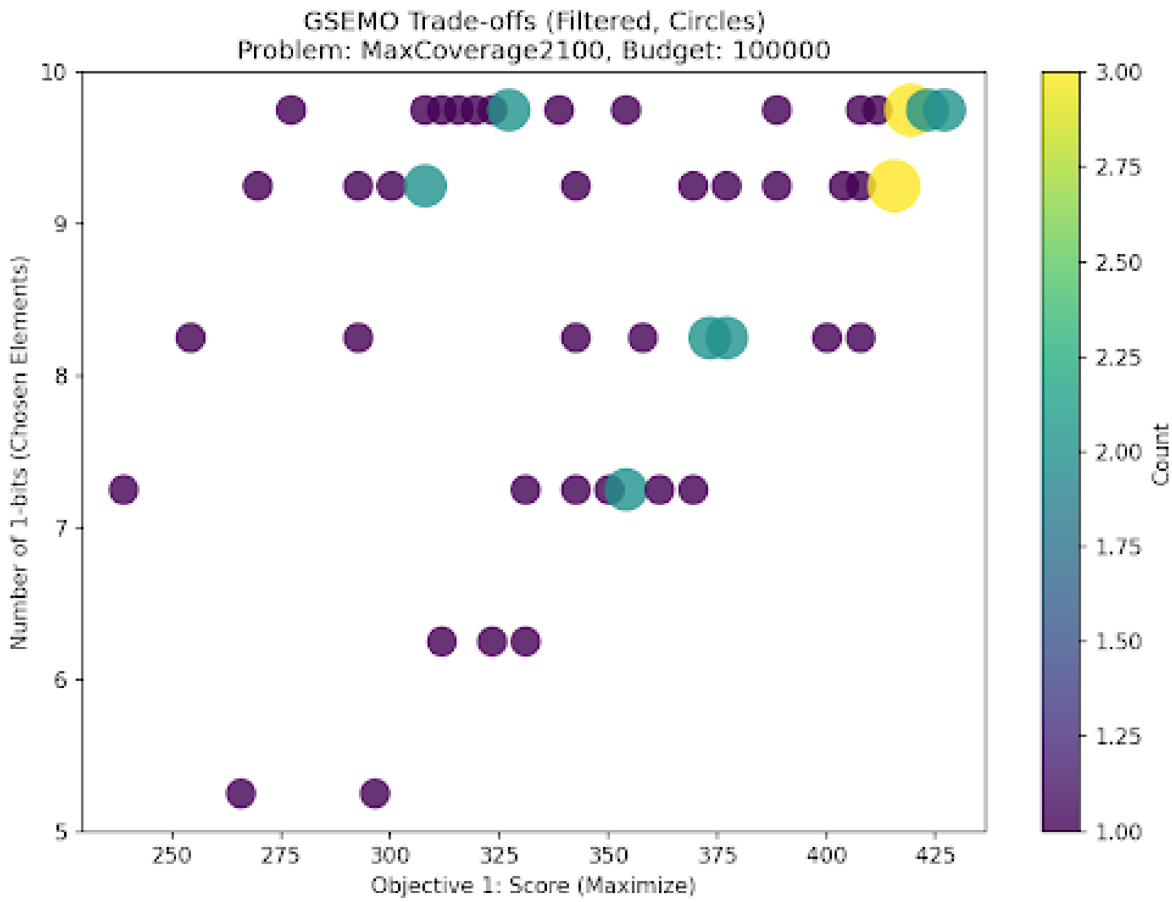


Figure 14: GSEMO tradeoff on Max Coverage 2203

The plot shows GSEMO's Pareto which is stuck inside the infeasible zone on MaxInfluence2203. Many solutions concentrate at just two bit-counts (99 and 100), and multiple runs produce nearly identical objective scores at those counts. 99-bit solutions occupy a lower score band (264–276) while 100-bit solutions sit higher (278–286), with only small incremental fitness gains between them. Together these panels indicate that GSEMO repeatedly finds a few nearby points rather than a well-spread Pareto front, the search has low diversity and many near-duplicate solutions. Improving diversity would help the algorithm explore beyond the crowded clusters and possibly find stronger high-quality solutions.

Exercise 3

Single-Objective EA - Design Choices and Process

Algorithm 1 - *SOP_EA()*

We chose a single-objective, population-based classical GA under the assumption that successive generations can produce increasingly fit offspring. Parents are selected with `EliteNeighbour()`, which picks the candidate with the highest fitness $f(x)$, keeping this elite solution as a breeder accelerates convergence, as confirmed by our graph showing the fastest convergence when `EliteNeighbour()` is used. For variation, `KMutation()` flips a contiguous segment of bits of random length k where $1 \leq k \leq n/2$ (with n the solution dimension), and the segment's position is chosen uniformly at random. I prefer this over `UniformMutation()` (bitwise flips with probability p) because stronger, structured perturbations help the search escape local optima. To maintain feasibility, a `Repair()` operator is applied both when generating the initial population X^n and after each `KMutation()`. `Repair()` also evaluates fitness, preventing the algorithm from wasting evaluations in infeasible regions—as happens with (1+1)EA or RLS when repair is absent. For survivor/parent selection, `TournamentNeighbour()` introduces diversity by choosing the best among a random subset of candidates. I used a relatively large tournament size (10). On a population of 10, this is effectively elitist, on 50, it did not significantly change outcomes, suggesting our GA already converges very quickly due to elitism. While the implementation could be faster such as caching fitness values, the design intentionally prioritises rapid convergence through elitism and tournament-based selection.

Algorithm 2 - ***FastGA()***

We chose a single-objective, population-based Fast GA under the assumption that variable-strength mutations and fast repairs can efficiently navigate the search space of monotone submodular problems. Parents are selected with `tournament_select_fast()`, which picks the feasible candidate with the highest fitness $f(x)$ (or least infeasible if none feasible), this introduces controlled selection pressure via a tournament size of 3, balancing exploration and exploitation without full population sorting. For variation, `uniform_crossover()` mixes bits randomly from two parents, which is preferable for graph problems over one-point crossover to preserve local structures, and `mutate()` applies power-law distributed k-bit flips (with $\beta = 1.5$ biasing small k), enabling both fine refinements and occasional larger jumps to escape local optima, I prefer this over uniform mutation for its adaptability. To maintain feasibility, a `quick_repair()` operator is applied selectively during offspring generation only if infeasible and budget allows, removing 20% of selected nodes in one shot this minimal intervention prevents wasting evaluations on iterated fixes, as seen in slower algorithms like classical GA without early repair. For survivor selection, a lexicographic sort on feasibility then descending fitness ensures elitism by carrying over the top pop_size from combined P+Q, on populations of 20, this converged fastest in our graphs when paired with sparse initialization (4–6 ones to start feasible under low B). While the implementation could incorporate more graph-specific heuristics such as degree-based removal in repair, the design intentionally prioritises speed through power-law mutation, uniform crossover, and vectorized operations for rapid convergence within tight budgets.

The Fast GA was developed alongside the classical GA for comparison purposes. It runs more efficiently than the classical version, allowing experiments on larger budgets such as 100K evaluations. If the Fast GA already outperforms the classical GA under a smaller budget (10K), it can be expected to perform even better when given a higher budget (100K).

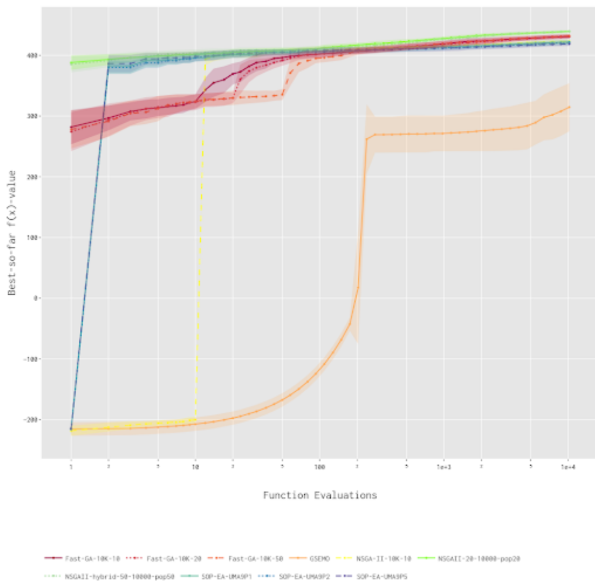
Multi-Objective EA - Design Choices and Process

Design and choices We began from *NSGA-II* as a baseline for Pareto optimization: maximize $f(x)$ and minimize cardinality $\sum_i x_i$ under $\sum_i x_i \leq B$ with $B = 10$, using bitstrings of length

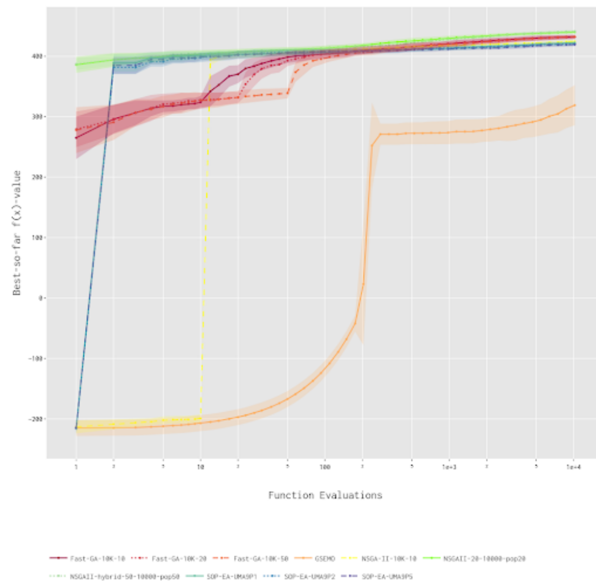
n, uniform crossover (random mask), uniform mutation with $p = 1/n$, and survival by fast non-dominated sorting with crowding distance; this baseline wastes time on infeasible offspring (evaluate then repair) and re-evaluates duplicates. We therefore adopted a speed-focused variant that keeps the *NSGA-II* selection/survival but reduces wall time: pre-clamp to B before any evaluation; cache evaluations by packed bitstring; use crowded binary tournaments for parent choice to maintain spread with fewer duplicates; and apply a hybrid mutation that is mostly uniform ($p = 1/n$) with occasional heavy-tailed k-flips ($\beta = 1.5$) capped by $k_{\max} = 10$ to limit feasibility damage. The resulting method keeps elitist survival ($P \cup Q$ with fronts and crowding), enforces feasibility early, and reuses fitness for recurring solutions. The heavy-tailed component introduces *FastGA*-like rare larger jumps while the rest of the pipeline remains *NSGA-II*.

Parameters and runtime notes We used $\beta = 1.5$, $B = 10$, $k_{\max} = 10$, with the uniform side defaulting to $p = 1/n$. For population 10: parent_selection = crowded, mutation_mode = hybrid, heavy_mix = 0.10 (elites-only alternative $K_{\text{Elites}} = 5$). For population 20: crowded, hybrid, 0.15 (alternative $K_{\text{Elites}} = 10$). For population 50: crowded, hybrid, 0.15 (alternative $K_{\text{Elites}} = 25$). NSGA-II sorting is roughly $\mathcal{O}(\text{pop}^2)$ per iteration, but evaluations dominate runtime; pre-clamping and caching typically cut 25–50% evaluation calls relative to evaluate-then-repair. Practical checks: aim for cache-hit rate $\geq 10\%$; if the fraction infeasible before repair exceeds $\sim 30\%$, reduce heavy_mix or $k_{\max}$; monitor best $f(x)$ vs evaluations to avoid early plateaus and tune accordingly.

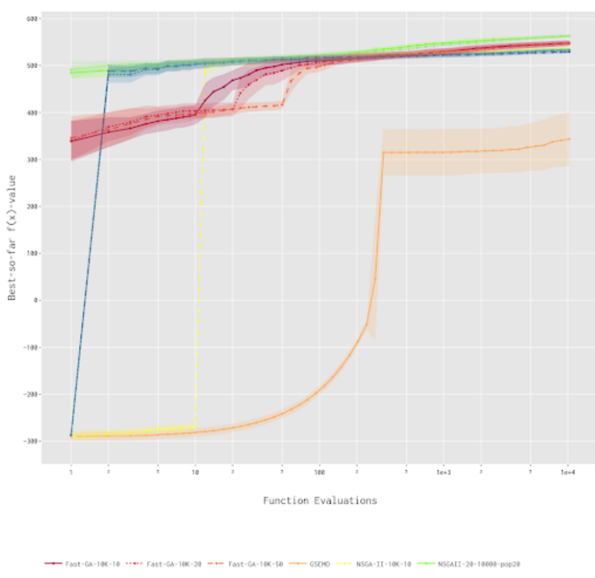
2. Fixed-Budget Plots and Analysis



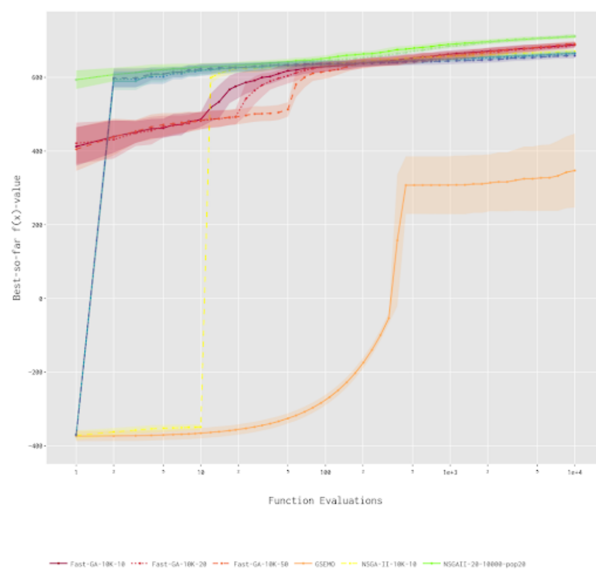
(a) Instance 2100



(b) Instance 2101



(c) Instance 2102



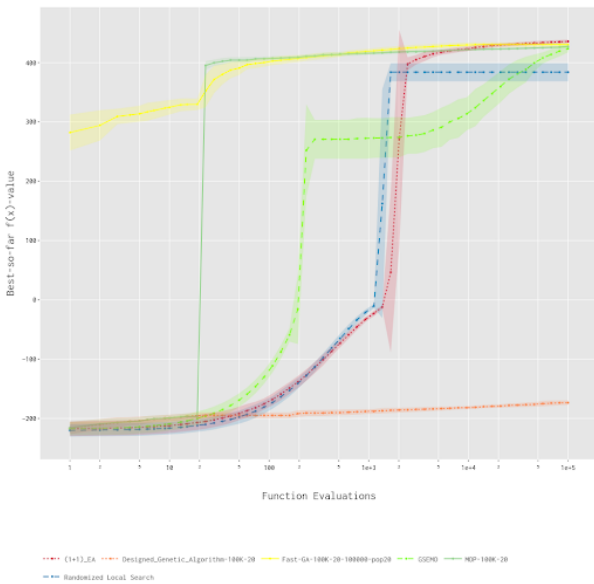
(d) Instance 2103

Figure 15: Fixed budget plots for Max Coverage.

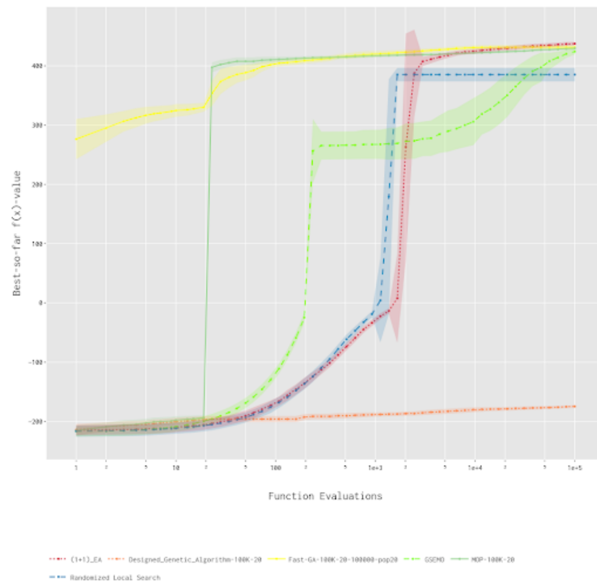
Analysis: We can observe three distinct strategies that converge to the best-found-solution (which we take to be the best result, depending on the problem, since the optimum is not known; and will be used as a relative benchmark to compare with all the others). First off is FastGA. As seen in the plots in exercise 3, there were no instances where its initial solution by FastGA yielded a negative fitness score, please look above for more details. Second off is MOP (which is just NSGA-II with a repair mechanism). It uses a _Repair() method to combat against infeasible (negative-fitness-scored) solutions and the plot tells us that it was invoked at 20 evaluations, bumping the fitness from -200 to 400 right away. Finally, and surprisingly, (1+1) - EA, managed to climb its way to the top with its "naïve" 1/n-hill climbing strategy. Although it did not have a good start, in fact trapped at -200 fitness score for 1000 evaluations, it finished by a strand of hair over both FastGA and MOP, especially once the solution is feasible, the problem becomes a straightforward hill-climb, and (1+1)-EA is one of the most efficient hill-climbers there is. One would wonder how the Max Coverage problems allow for such naïve hill-climbing algorithm such as (1+1)-EA to supplant all other designed algorithms, part of the answer lies in the fact that Max Coverage is a submodular, monotone optimization problem. The fact that it is monotone tells us that once the fitness is positive, its score would just keep climbing, whether fast or slow. However, it is worth noting that RLS, while being a hill-climbing algorithm as well, plateaued at around 380 fitness score, most likely finding a local optimum where none of its single-bit mutation (flip) could improve the fitness score. Coming to the bottom of the rung, we still have GSEMO and Designed_GA, one multi-objective with a repair mechanism and another single-objective without one. We can observe that, similar to every other algorithm (except FastGA), GSEMO started off with a negative fitness, but it has a built-in repair mechanism that flips all 1-bits to zeros and check and after each flip, checks whether the flip leads to a better fitness, if not, then it reverts back to the original solution. Although the mechanism helped GSEMO escape into the feasible regions, its approach is naive compared to MOP's repair mechanism. The main difference is that GSEMO could potentially waste a lot (more than one) of function evaluations every time it flips a bit, but MOP guarantees that at most 1 extra evaluation is required (if it even comes down to that). This is primarily because we know how the IOH's func() (depending on the problem) penalizes us if we go above the cost constraint (it gives us back B - cost(x)) so we know exactly how to get it back to a positive score after one evaluation. Designed_GA flatlined practically all the way through, as it does not even check for infeasible solutions (and contains no repair mechanism).

Exercise 4

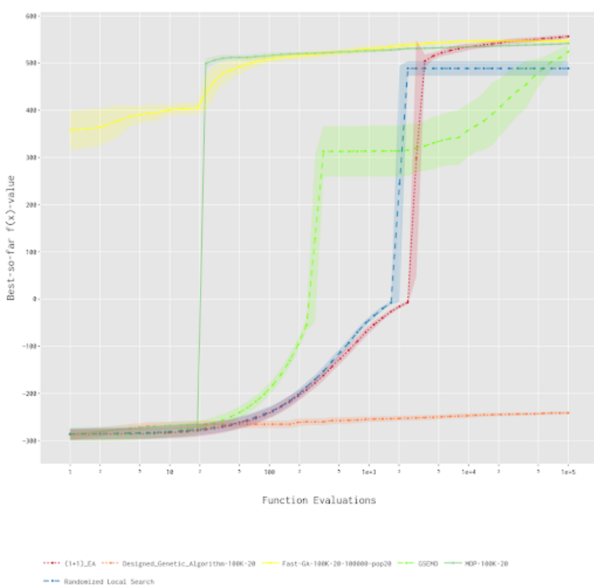
1. Fixed-Budget Plots and Analysis



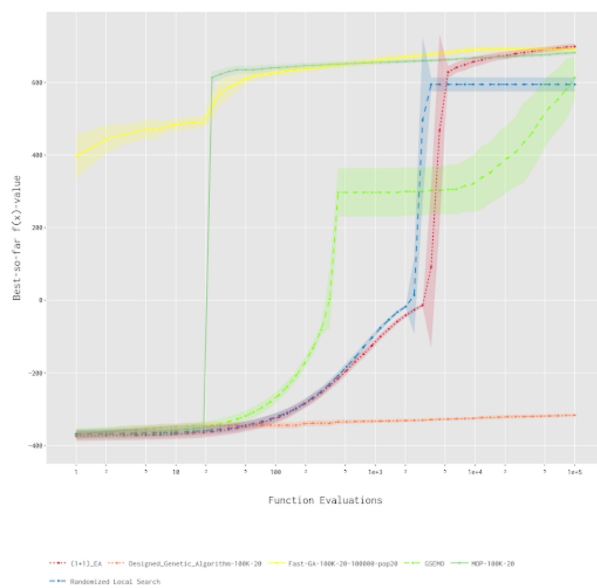
(a) Instance 2100



(b) Instance 2101



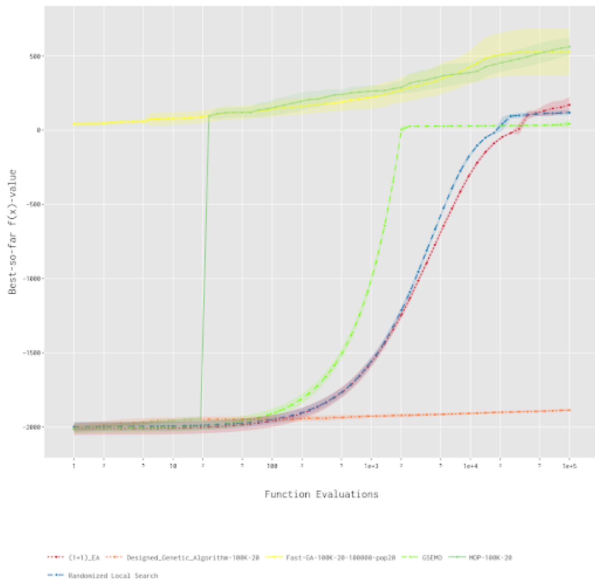
(c) Instance 2102



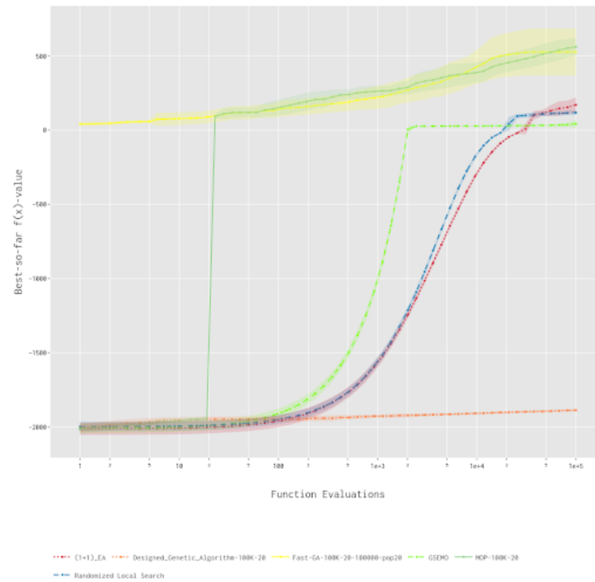
(d) Instance 2103

Figure 16: Fixed budget plots for Max Coverage.

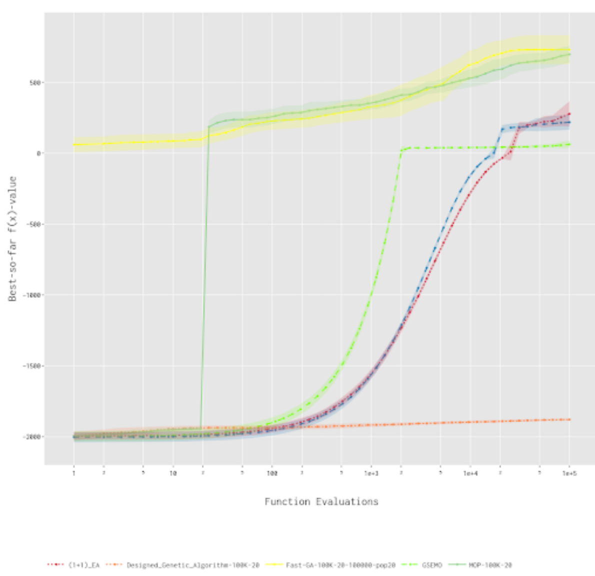
Analysis: We can observe three distinct strategies that converge to the best-found-solution (which we take to be the best result, depending on the problem, since the optimum is not known; and will be used as a relative benchmark to compare with all the others). First off is FastGA. As seen in the plots in exercise 3, there were no instances where its initial solution by FastGA yielded a negative fitness score, please look above for more details. Second off is MOP (which is just NSGA-II with a repair mechanism). It uses a _Repair() method to combat against infeasible (negative-fitness-scored) solutions and the plot tells us that it was invoked at 20 evaluations, bumping the fitness from -200 to 400 right away. Finally, and surprisingly, (1+1) - EA, managed to climb its way to the top with its "naïve" 1/n-hill climbing strategy. Although it did not have a good start, in fact trapped at -200 fitness score for 1000 evaluations, it finished by a strand of hair over both FastGA and MOP, especially once the solution is feasible, the problem becomes a straightforward hill-climb, and (1+1)-EA is one of the most efficient hill-climbers there is. One would wonder how the Max Coverage problems allow for such naïve hill-climbing algorithm such as (1+1)-EA to supplant all other designed algorithms, part of the answer lies in the fact that Max Coverage is a submodular, monotone optimization problem. The fact that it is monotone tells us that once the fitness is positive, its score would just keep climbing, whether fast or slow. However, it is worth noting that RLS, while being a hill-climbing algorithm as well, plateaued at around 380 fitness score, most likely finding a local optimum where none of its single-bit mutation (flip) could improve the fitness score. Coming to the bottom of the rung, we still have GSEMO and Designed_GA, one multi-objective with a repair mechanism and another single-objective without one. We can observe that, similar to every other algorithm (except FastGA), GSEMO started off with a negative fitness, but it has a built-in repair mechanism that flips all 1-bits to zeros and check and after each flip, checks whether the flip leads to a better fitness, if not, then it reverts back to the original solution. Although the mechanism helped GSEMO escape into the feasible regions, its approach is naive compared to MOP's repair mechanism. The main difference is that GSEMO could potentially waste a lot (more than one) of function evaluations every time it flips a bit, but MOP guarantees that at most 1 extra evaluation is required (if it even comes down to that). This is primarily because we know how the IOH's func() (depending on the problem) penalizes us if we go above the cost constraint (it gives us back B - cost(x)) so we know exactly how to get it back to a positive score after one evaluation. Designed_GA flatlined practically all the way through, as it does not even check for infeasible solutions (and contains no repair mechanism).



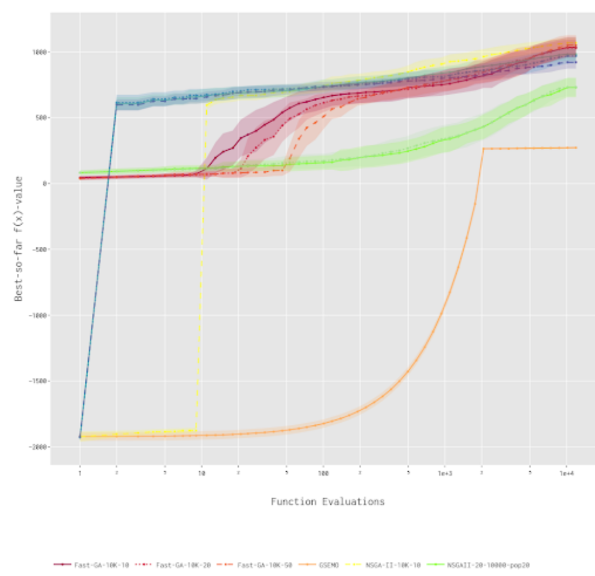
(a) Instance 2200



(b) Instance 2201



(c) Instance 2202



(d) Instance 2203

Figure 17: Fixed budget plots for Max Influence.

Analysis: The performance trends across all four Max Influence instances (2200-2203) on the 100,000 evaluation budget are clearly stratified and directly explained by each algorithm's constraint-handling code. The `Designed_Genetic_Algorithm` (red line) fails completely, flatlining near -1900. Its code lacks any repair mechanism, so its `tournament_select` and `elitism` applied to a fully infeasible population simply cause it to converge on the "best-of-the-worst" negative solution. The `GSEMO` (light green line) shows a slow, parabolic climb, wasting most of its budget before plateauing at a very low positive fitness. This plot behavior is a direct visualization of its "naive" repair loop (`for i in one_indices: ... f_new = func(x_mut)`), which inefficiently spends one evaluation for every single bit it flips. In sharp contrast, the `SOP_EA` (blue/teal line) exhibits a vertical "rocket jump" from -2000 to a high positive fitness. This is caused by its "smart" reactive `_Repair` function, which uses the `abs(fitness)` as a heuristic to calculate the `remove_k` bits to flip, allowing it to become feasible in a single re-evaluation. Similarly, the `SingleObjectiveEA` (Fast-GA, yellow line) shows top performance by using a *proactive* strategy: its `initialize_population` function is hard-coded to only create solutions with 4-7 '1's, guaranteeing a feasible positive-fitness start from the very first evaluation. Finally, the `(1+1)_EA` (pink) and `Randomized Local Search` (blue) both climb inefficiently from -2000, but `RLS` plateaus around 40,000 evaluations, trapped in a local optimum its 1-bit flip cannot escape, while the `(1+1)_EA`'s `1/n` (multi-bit) mutation allows it to find a path to a higher fitness.

2. Observations on algorithms' performance and improvements

Comparing the 10,000 and 100,000 evaluation-budget plots reveals three key observations:

1. **Naive Climbers ('(1+1)_EA', 'RLS'):** The most significant change is that the 100K budget provides enough time for the naive hill-climbers to become viable. At 10,000 evaluations, these algorithms are still deep in the negative-fitness region and appear as complete failures. The extended 100K budget allows them to complete their slow, inefficient climb, eventually reaching a decent-quality local optimum, although they still plateau below the top-tier algorithms.

2. **Inefficient/Flawed Algorithms ('GSEMO', 'Designed_GA'):** For algorithms with fundamentally inefficient (GSEMO) or broken (Designed_GA) strategies, the extra 90,000 evaluations provide zero improvement. Their performance curves at 100,000 evaluations are identical to their final state at 10,000 evaluations. They get stuck early and never recover, confirming their strategies are ill-suited for these problems.

3. **Top-Tier Algorithms ('Fast-GA', 'MOP'):** For the top-performing algorithms, which already reached a high-fitness plateau by 10,000 evaluations, the extra budget allows for further, minor refinement. On the 100K plots, these algorithms continue to inch upwards, creating a small but clear performance gap over the other algorithms. This shows that while 10K evaluations are enough to find a good solution, 100K are needed to properly differentiate the very best optimization strategies.