A GPU-based Genetic Algorithm for the Set Cover Problem

CS 7387: Research in Computer Science

Andres Gonzalez
Texas State University
andres.gonzalez@txstate.edu

Martin Burtscher*
Texas State University
burtscher@txstate.edu

December 2025

Abstract

The Set Cover Problem (SCP) involves choosing a subset of sets (or hyperedges), each containing vertices of a graph, such that the union of the chosen subsets "cover" all vertices of that graph. Additionally, each set has an associated cost (or weight), and the objective of SCP is to find a cover with the minimum weight. Since SCP is NP-hard, there are a variety of implementations, including exact solvers, heuristics, genetic algorithms, and other algorithms inspired by nature. Most of the literature focuses on Linear Programming approaches, framing the SCP as a linear optimization problem.

In this study, I implemented serial and parallel exhaustive exact solvers on both the CPU and GPU. Compared to third-party solvers, these implementations are slower for instances consisting of at least 30 vertices and 30 sets, mostly due to the third-party codes exploiting mathematical concepts such as the Simplex method. Third-party solvers are able to provide optimal solutions in less than 10 seconds for inputs consisting of fewer than 400 vertices and 4,000 sets, but instances that are larger than 2,500 vertices and 1,000,000 sets can take longer than 1,000 seconds, with some instances timing out at 2 hours. For the larger inputs (e.g., inputs consisting of 1,000 vertices and 10,000 sets), matrix-based approaches start becoming intractable, and this has led me to design a heuristic approach that combines finding a maximal independent set with a genetic algorithm. My heuristics yield solutions that differ from the optimal (or best-known solution) by as little as 2.3% in under 20 seconds, showcasing the usefulness and potential of heuristic-based algorithms to process large SCPs.

1 Introduction

When we think of a graph or a network, we normally think of the graph in terms of vertices and the edges that connect those vertices, where each edge has incident on it a vertex at either of its ends. That is, a graph G is defined by its vertices V and edges E, such that $G = (V, E)$. The "Set Cover Problem" (SCP) involves graphs that instead contain hyperedges—edges that connect more than two vertices—where the objective is to select a set of hyperedges such that the union includes (or "covers") all the vertices of the graph [8, 15]. Furthermore, like graphs with edges, every hyperedge (henceforth called a "set") is associated with a weight or "cost", and the secondary objective of the SCP is to minimize the total cost of the chosen sets. In this work, we use the terms "weight" and "cost" interchangeably. A *solution* to this problem that includes the chosen sets is considered *feasible* when all vertices in the graph have been accounted for, and vice versa, a solution is considered *infeasible* if there are yet missing vertices in the union of the chosen sets.

Given that every set x has an associated cost c, the SCP can be formalized as follows:

*Advisor

minimize

$$\sum_{j=1}^{n} c_j x_j \tag{1}$$

subject to

$$x_j \in (0,1); \qquad j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1; \qquad i = 1, \ldots, m \tag{3}$$

Equation (2) denotes that every set x_j can only take on the values 0 or 1, and when applied to equation (1), this means that the summation across all of the chosen sets' costs, when multiplied by their inclusion or not (denoted by $x = 1$ or 0, respectively), reaches a final value that is the final "weight" of the solution. Again, the goal of SCP is to choose a set of sets whose cost is minimal and solution is *feasible*. Equation (3) denotes the constraint that, for all entries in the graph's incidence matrix a, the summation across the i'th row must be greater than or equal to 1. Stated another way, for every vertex, there must be at least one set in the solution that includes that vertex, and equation (3) can be read as, "for all j columns of the ith row, multiply that entry a_{ij} by whether the set x_j is included or not, and take the sum of that multiplication across all sets j for that ith row, where the sum must be at least 1". In this context, a solution is still considered feasible if a vertex is included more than once in the set of chosen sets.

The SCP is known to be NP-Hard [9, 15, 17]—that is, whereas it is easy to verify whether a solution is feasible by simply checking the union of the selected sets, it is indeed difficult to determine whether the solution is the optimal among the possibilities in the solution search space. As discussed below, exhaustively-brute-forcing every possible combination of sets is intractable for large-scale inputs as the computation necessary undergoes a combinatorial explosion as the inputs grow larger. Specifically, the growth rate is exponential, for every added set to the problem size doubles the number of combinations that need to be checked.

The SCP finds applications in several industries: it can determine facility location selection [1], where facility proximity fulfills certain emergency response or recycling requirements [29, 6]; it can be used in aircraft crew scheduling to determine where staffing is allocated to cover flights [35]; it can be used to determine truck path planning [27]; and it can be used to determine effective placement of gas detectors in chemical plants [6].

2 Related Work

SCP Variations

Unicost vs. Non-Unicost The SCP described in this study is "non-unicost" in that the sets of the problem instances are all of varying costs. The "unicost" variation of SCP implies that all sets have the same cost, where the costs are implied as having the value 1. Strictly speaking, the set costs for the unicost SCP can have any value so long as the value is the same for all sets [7, 28, 27].

Set Partitioning Problem The "Set Partitioning Problem" can be formalized by replacing the inequality in equation (3) by an equality, where the constraint is to include the vertex once and only once in the final solution. In this SCP variation, a solution would be considered infeasible if multiple sets in the final solution contain the same vertex [8].

Partial Set Cover The "Partial SCP" consists of finding a set of sets that cover, at minimum, a specific portion of vertices in the graph whilst leaving some other vertices to be covered as optional [9, 10, 26].

Minimum Set Cover Problem Whereas the objective of the SCP is to find a solution whose weight is minimal, the objective of the "Minimum SCP" is to find a solution whose only requirement is that the number of sets in the solution is minimal (and therefore optimal) [19]. In this scenario, a solution that includes fewer sets but weighs more is considered better than a solution that includes more sets but weighs less.

Conflicts on Sets The "Conflicts on Sets" variation of the SCP involves finding a solution that minimizes number of "conflicts" in the solution [15] as well as the costs. Here, a conflict is defined as a vertex showing up more than once in a solution, and each vertex's set-pairwise appearance constitutes a conflict k. The solution remains permissible so long as a penalty can be paid, and in the formulation of "Conflicts on Sets", k-many

conflicts may arise before the solution is discarded as infeasible. The authors in [32] call this variation "set covering problem with conflict constraints" (SCC).

Small Neighborhood Property The presented SCP is also known as the "minimum weight set cover" (MinWSC) [31], and the small neighborhood cover "SNC" furthers the formalization of SCP—a problem instance is considered as having the τ-SNC property if for every vertex v, there are τ-many sets that cover v and all of its neighbors. Here, vertices are considered neighbors if they are a part of the same set.

And More Further works involve the placement of security cameras to cover a facility (the "angular" set cover problem [6, 25]), updating the chosen set of sets as the vertices of a network are dynamically changed (the "dynamic greedy set cover" [34, 21]), ordering the vertices in the union of the chosen sets so as to minimize the sum of the number of times a set is covered (the "min sum set cover" [5, 4]), finding a cover such that the wildlife diversity of species is present in at least one reserved site (the "reserve set cover problem" [2]), finding a solution where every vertex is covered by at least k-many sets (the "set k-cover problem" [33, 30]), and the formulation even goes further into the domain of natural language processing, where it is described as "the Minimum Set Cover Problem with several constraints" [20]. An alternative formulation of SCP also involves the "maximum set k-cover problem", where the goal is to instead find a cover that maximizes the sizes of all the chosen sets [36], and other formulations include "the minor set cover" [23] and the "geometric set cover problem" [13, 16]. The reader is encouraged to formulate new possibilities and variations for further investigations into the SCP.

Linear Programming

The astute reader will have noticed that equations (1)-(3) correspond to a set of equations that regularly make an appearance in the field of mathematics known as "Linear Programming", often abbreviated as LP. Indeed, equation (1) is what's known as the *objective function* and equations (2) and (3) are the *constraints*, and these equations describe the SCP as a *linear optimization* problem.

The field of "Linear Programming" is more in the domain of Operations Research than it is in coding programs, and instead revolves around the notion of knowing how to model an optimization problem in a linear optimization framework. Once the problem is modeled with an objective function and a set of constraints in a standardized manner (e.g. see the "MPS" file format discussed below), the problem is then fed into a linear programming solver, and running the solver produces an "optimized" solution which can then be used to (e.g.) make business decisions. The field of Linear Programming also includes, but is not limited to, subdomains such as Mixed-Integer Programming (MIP), Integer Programming (IP), Binary Integer Programming (BIP), Mixed-Integer Non-Linear Programming (MINLP), Quadratic Programming (QP) and Convex Quadratic Programming (CQP), Fractional Linear Programming (FLP), and Second Order Cone Programming (SOCP).

The third-party solvers used in this study take on the task of solving linear optimization problems by implementations that derive from matrix-based approaches, and these approaches may involve the Simplex Method [14], interior point methods, branch and bound methods, or methods such as the "primal-dual hybrid gradient" approach. Unfortunately, methods involving matrices cannot be pragmatically applied to large-scale inputs such as graphs that contain millions of vertices and millions of edges, where an adjacency or incidence matrix of that graph would require a space complexity of at least $O(n^2)$. In the case of a graph of $1,000,000$ vertices and $1,000,000$ sets, this matrix would require $1,000,000 \times 1,000,000 = 1,000,000,000,000$ entries (or 4 TB of memory if using a 4-byte `int` per entry) for the incidence matrix alone. Thusly, it is important to find alternative means for addressing the SCP with large-scale graphs that do not take a matrix-based approach.

2.1 Solvers

It is clear that the field of research backing linear optimization problems (and its associated subdomains) has economic interest in the everyday business, both small and large. With companies such as Apple and Microsoft engaging in utilization of commercial software as a means of carrying out their business plans, it is evident that there is an undoubtable need to continue the efforts in finding more robust and practical solutions that bring along guarantees such as rigorously-proven optimalities.

2.1.1 Closed-Source Implementations

IBM ILOG CPLEX [1]

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio

IBM's ILOG CPLEX, named after the simplex method that was implemented in the C programming language[2], is IBM's solution for linear optimization problems. IBM describes their implementation as a method that enables "rapid development and deployment of decision optimization models that use mathematical and constraint programming".

Gurobi [3]

Gurobi Optimization, established in Oregon, is an enterprise solution to linear optimization problems. In addition to providing software for modeling and optimizing linear models, they also provide a platform of services that are geared towards making better business decisions in general (what Gurobi describes as "business intelligence"). Their customer base consists of major corporations such as Google, AT&T, T-Mobile, Microsoft, AMD, Apple, hp, SAP, Tesla, Uber, amongst others, and one could interpret this as an indication that Gurobi is a leading provider of optimizers.

FICO Xpress [4]

FICO Xpress, based in Montana, has history going back to the early 1980s, claiming titles such as being the first commercial software to run on PCs and being the first to provide the first parallel implementation of the parallel primal dual simplex method[5]. As of 2022, they also claim to have succeeded in the implementation of a global solver for mixed-integer nonlinear optimization problems that is also capable of proving optimality[6].

Mosek [7]

Mosek, headquartered in Copenhagen, Denmark, touts the adaptation of their academic license throughout many research institutions, and further claims that their interior-point optimizer for linear, quadratic, and conic problems is the state of the art. Their customers include MathWorks (the company behind MATLAB), Wolfram|Alpha, Sony, and Virgin Mobile as well as UC-Berkeley and Harvard University.

2.1.2 Open-Source Implementations

The results reported in this work utilized open-source implementations of solvers. Unfortunately, the End-User License Agreement for Gurobi Optimization, for example, forbids publishing information regarding performance studies comparing their implementation to others. They offer academic licenses, but further require permissions for publishing acquired data. In light of this, our preliminary results only include that of the open-source implementations, with future work potentially including results obtained from the other closed-source academic-license providers such as Mosek.

cuOpt [8]

cuOpt is NVIDIA's GPU solution to linear optimization. In addition to being capable of solving MIP and LP, as well as vehicle routing problems, their implementation is open-source. This provides an excellent opportunity to specifically compare their CUDA GPU implementation that derives from the perspective of linear optimization to our GPU graph-based implementations.

HiGHS [9]

HiGHS[24] stands out as the state-of-the-art open-source implementation that MathWorks uses for their MATLAB platform. MATLAB's `intlinprog` function call is a wrapper for the HiGHS implementation, where MATLAB output matches that of running HiGHS outside the MATLAB context. It can solve all of LP, CQP, and MIP problems.

CyLP / CBC [10]

The Computational Infrastructure for Operations Research organization (COIN-OR) is a scientific non-profit organization that implements various open-source solvers, specifically the COIN-OR Linear Programming solver (CLP), the COIN-OR Branch-and-Cut solver (CBC), and the COIN-OR Cut Generation Library (CGL).

[2] https://en.wikipedia.org/wiki/CPLEX
[3] https://www.gurobi.com/
[4] https://www.fico.com/en/products/fico-xpress-optimization
[5] https://en.wikipedia.org/wiki/FICO_Xpress
[6] https://community.fico.com/s/blog-post/a5Q4w000000D26XEAS/fico3785
[7] https://www.mosek.com/
[8] https://github.com/NVIDIA/cuopt
[9] https://github.com/ERGO-Code/HiGHS
[10] https://github.com/coin-or/CyLP

Table 1: Beasley OR-Library Benchmark Problem Sets

Problem Set	Vertices	Sets	Density
4	200	1000	2%
5	200	2000	2%
6	200	1000	5%
A	300	3000	2%
B	300	3000	5%
C	400	4000	2%
D	400	4000	5%
E	50	500	20%
NRE	500	5000	10%
NRF	500	5000	20%
NRG	1000	10000	2%
NRH	1000	10000	5%

Table 2: The Rail Benchmarks

Problem Instance	Vertices	Sets	Density	Max Cost
Rail507	507	63009	1.3%	2
Rail516	516	47311	1.3%	2
Rail582	582	55515	1.2%	2
Rail2536	2536	1081841	0.4%	2
Rail2586	2586	92683	0.3%	2
Rail4284	4284	1092610	0.2%	2
Rail4872	4872	968672	0.2%	2

"CyLP" is a Python interface to the underlying solvers. Here, we compare our GPU implementation against the CBC solver (which internally also uses the CLP implementation).

SCIP [11]

The Solving Constraint Integer Programming "SCIP" [11, 12] framework is developed by the Zuse Institute Berlin, the same organization that is also responsible (with associated partners) for developing the MIPLIB 2017 [22] datasets used in this study. They provide "one of the fastest non-commercial solvers for MIP and MINLP", and the platform is led by a team of various academics alongside various academic and commercial institutions such as the University of Twente in the Netherlands and the Technische Universitat Darmstadt in Germany, to the same previously-mentioned FICO and Gurobi, as well as SIEMENS.

3 Datasets

Beasley OR-Library The SCP benchmark datasets used in this study and throughout the literature were officially published by Beasley in 1987[8] and included benchmarks from Balas and Ho's 1980 set cover publication[3]. The Beasley dataset was named the "OR-Library", and it included problem sets "4-6" and "A-H" (see Table 1). Each of problem sets 4-6 include several variations of SCP problem instances that contain a varying number of vertices and sets, where the sets' costs were randomly-generated with an upper limit of 100. Problem set "E" contains unicost SCP instances, where all set costs are 1. There is also a varying degree of density in each problem instance, where non-zero values appear in either 2% or 5% for problem sets 4-6 and A-D, and problem set "E" contains more non-zero values at a density of 20%.

As part of Beasley's dataset, there are also several larger "Rail" SCP instances, with upwards of millions of sets and thousands of vertices, that reflect real-world set cover problems. They are named "Rail" due to the construction of the problem sets from Italian railways. These benchmarks are available at J. E. Beasley's homepage[12].

[11] https://github.com/scipopt/scip

[12] https://people.brunel.ac.uk/ mastjjb/jeb/orlib/scpinfo.html

MIPLIB 2017 The MIPLIB 2017[13] [22] benchmark collection was the result of research institutions requiring more datasets that largely reflected real-world problems. Originally created by Bixby, Boyd, and Indovina in 1992, the benchmark suite has continuously been on a seven-year release cycle since 2010, with releases in 2017 and 2024, though the 2024 benchmark collection is still taking submissions and has not been officially released at the time of this writing. Additionally, the 2017 collection contains problem instances that yet have a status of "open", denoting that these particular benchmarks have not had their optimal solution mathematically-proven, giving rise to opportunity for anyone willing to take on the challenge.

File Formats

The Example Input Shown in Listing 1 is an example of an SCP that can be fed into linear optimizers for solving. It consists of 5 sets with costs 1, 2, 3, 4, and 5, respectively. Lines 2-4 denote the incidence matrix of the graph. Vertex 1 (shown on line 3) makes an appearance in sets 1 and 2, Vertex 2 (shown on line 4) makes an appearance in sets 2, 3, and 4, and Vertex 3 (shown on line 5) makes an appearance only in set 5. This example is described in both the Beasley SCP file format and the MPS file format below.

Listing 1: The Example Input

```
1: Set Costs: [1 2 3 4 5]
2:
3: Vertex 1:  [1 1 0 0 0]
4: Vertex 2:  [0 1 1 1 0]
5: Vertex 3:  [0 0 0 0 1]
```

Beasley SCP Format Listing 2 shows the "simple" example from above of what an SCP problem input looks like. The format is fairly easy to grasp and is straightforward. On line 1, the first two numbers denote the number of rows (i.e. the number of vertices in the problem instance) and the number of columns (i.e. the number of sets in the problem instance). Line 3 then lists the costs of the columns (the costs of the sets) in their respective order. In the example, the first column (set) has a cost (weight) of 1, and the second set has a cost of 2, etc.—there should be as many costs as there are sets. Lines 5 and onward are shown in line pairs, where the first line denotes how many sets that row (vertex) shows up in, and the second line denotes specifically which sets those are—the listings are always in order of vertex, meaning the first pair of lines is vertex 1. The next pair of lines is vertex 2, and the next pair is vertex 3. It is not difficult to see that one can generate a graph compressed-sparse-row (CSR) representation of this graph quite easily. Note that the lines separating the digits need not be actual line feeds, as all that is needed between the numbers is whitespace. The semicolon characters and the comments following them are not a part of the benchmarks themselves and are included here to aid in explaining the file format.

Listing 2: The Beasley OR-Library Format

```
1: 3 5            ; NUM ROWS, NUM COLUMNS
2:
3: 1 2 3 4 5      ; COLUMN COSTS
4:
5: 2              ; VERTEX 1 appears in "2" sets
6: 1 2            ; The sets are "1" and "2"
7:
8: 3              ; VERTEX 2 appears in "3" sets
9: 2 3 4          ; The sets are "2", "3", and "4"
10:
11: 1             ; VERTEX 3 appears in "1" set
12: 5             ; The set "5"
```

MPS File Format Standardized by IBM for linear optimization problem descriptions, the "Mathematical Programming System" (MPS) file format is the industry standard for representing linear optimization problems. Codes like cuOpt and HiGHS (used in MATLAB) are built to ingest these representations, and these representations have a specification outlined in IBM's CPLEX documentation site[14]. The MPS format can be

[13] https://miplib.zib.de/

[14] https://www.ibm.com/docs/en/icos/22.1.2?topic=formats-working-mps-files

used to represent linear optimization problems that are not limited to SCP, such as the set partitioning, set packing, or invariant knapsack problems. Listing 3 shows the same "simple" example as Listing 2, but in the MPS file format.

In short, the specification consists of sections, with designations for each section. The `NAME` section simply states the name of the problem input, the `ROWS` section denotes the constraints that are a part of the problem. Here, "N" means that the label following is the label for the problem's objective function, and the "G"s indicate that those constraints are "greater than or equal to" constraints. The `COLUMNS` section details the costs of the sets and the participating vertices for each of those sets, given by the correspondence of the constraint labels after the variable labels (e.g. variable (set) "x1" costs 1 and contains vertex `COV_V1`, which is vertex 1 from the `ROWS` section). The `RHS` section denotes the right-hand-side of the constraints (the values after the inequalities/equalities), and the `BOUNDS` section states that every variable (set) is a binary variable, taking on the values 0 or 1. This formulates the SCP as a linear optimization problem using the MPS file format.

Listing 3: The MPS File Format

```
 1: NAME            SIMPLE      ; The name
 2: ROWS                        ; Section ROWS
 3:  N  COST                    ; Denotes the objective function
 4:  G  COV_V1                  ; <-- Denotes a '>=' constraint,
 5:  G  COV_V2                  ;     which is a vertex
 6:  G  COV_V3
 7: COLUMNS                     ; Section COLUMNS
 8:     x1     COST     1       ; Denotes the cost of set x1
 9:     x1     COV_V1   1       ; <-- Denotes that constraint
10:     x2     COST     2       ;     COV_V1 participates in
11:     x2     COV_V1   1       ;     set x1
12:     x2     COV_V2   1
13:     x3     COST     3
14:     x3     COV_V2   1
15:     x4     COST     4
16:     x4     COV_V2   1
17:     x5     COST     5
18:     x5     COV_V3   1
19: RHS                         ; Section RHS
20:     RHS1   COV_V1   1       ; All right hand side values
21:     RHS1   COV_V2   1       ;   should be 1
22:     RHS1   COV_V3   1
23: BOUNDS                      ; Section BOUNDS
24:  BV BND1 x1                 ; Variable "x1" is a
25:  BV BND1 x2                 ;     "Binary Variable"
26:  BV BND1 x3                 ;     "BND1" label is ignored
27:  BV BND1 x4
28:  BV BND1 x5
29: ENDATA                      ; End of data
```

Benchmark Compositions Figure 1 shows the vertex compositions of the "Rail" problem instances, where the number of times each vertex makes an appearance in the problem instance is depicted on the y-axis (the Vertex ID is on the x-axis), and this shows that every vertex varies in the number of times it makes an appearance throughout the sets. As shown in Table 1, instances of the "Rail" problem set have a varying number of vertices and sets, and one could think of each of these depictions as a type of "fingerprint" for the problem instance. This allows for variety in the benchmark compositions which can be used to verify flexibility in solver implementations.

Shown in Figure 2 are the set-size bar plots for the "Rail" instances. We can observe that the set size and compositions vary in the real-world benchmarks, where there is a distinct count of sets for each of the set sizes ranging from 2 to 12 vertices. For example, "Rail-4872" has sets of size 12 dominating the instance, whereas "Rail-582" has sets of size 6 dominating the instance.

Figure 1: Counting the number of times each vertex makes an appearance in each of the different Rail benchmarks.

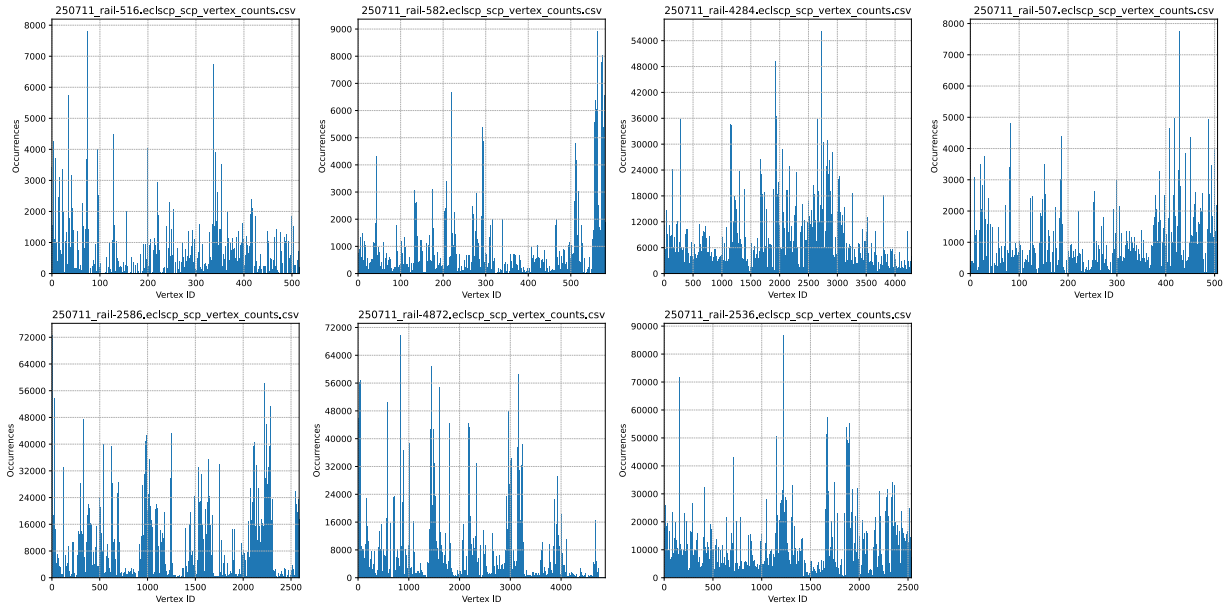
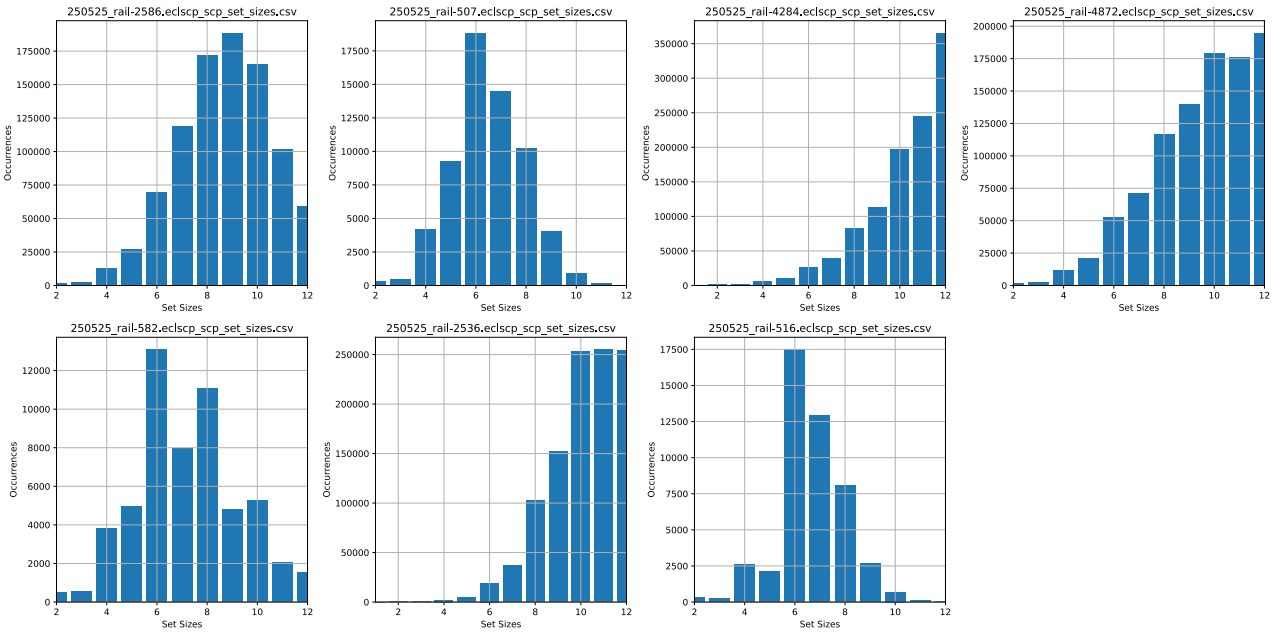


Figure 2: Set sizes and counts for the “Rail” problem instances.



4 Approach

4.1 Exhaustive Brute-Force

ECL-CPU-BF The naïve Brute-Force CPU implementation, after ingesting the SCP instance, simply checks the number of sets that are in the instance, and proceeds to check every possible combination of set inclusions and exclusions, keeping track of which solution has been both feasible and cheapest with regards to the total set costs along the way. The cheapest cost and set combination are reported. Note that there may be multiple optimal solutions, where the solutions cost the same but the solution set compositions may differ.

ECL-GPU-BF Much like the naïve Brute-Force CPU implementation, the Brute-Force GPU implementation carries out the same bit-centric checks on all possible solutions on the GPU. The SCP problem instance is ingested, sent to the GPU, the GPU processes all combinations, keeping track of feasible and cheapest combinations, and reports back to the CPU the optimal solution.

4.2 Our MIS/GA Implementation

Our fast Maximal Independent Set (MIS) and Genetic Algorithm (GA) implementation is a heuristic-based approach that aims to provide solution qualities that are close to the optimal (if not, optimal) in a practical amount of time. The algorithm is outlined in Algorithm Listings 1 and 2 below and proceeds in two phases.

MIS In `Phase I`, we begin by processing vertices that only show up once in the problem instance—that is, a vertex that uniquely shows up in only one set causes that set to automatically be required in the final solution, as no other sets contain that vertex.

Subsequently, the main focus of Phase I is the loop that follows the preprocessing step. For all vertices that have not been accounted for, find that vertex's cheapest set. After vertices have decided on their cheapest set, all sets then consider their vertices. If all vertices of a set have "chosen" that set as its cheapest, then mark that set for inclusion in the MIS result. After all sets have checked their vertex, all the vertices that are a part of the marked-for-inclusion sets are marked as accounted-for, and they are further removed from consideration. This process then repeats, where (1) vertices that have not been accounted for check-for and decide-on their cheapest set, (2) the sets that have their vertices chosen are marked for inclusion, and (3) the sets that are now included have their vertices removed from further consideration. As an additional detail to the loop, every loop iteration begins by removing sets that no longer contain any vertices—that is, due to vertices being accounted-for in previous loop iterations, sets may become empty, for which they no longer need to be considered for inclusion.

GA In `Phase II`, we then further process the solution output of the MIS - Phase I implementation as a Genetic Algorithm. In this algorithm, we have a population of organisms, where every organism is a "potential solution"—that is, every organism is a bitstring of length n (n being the number of sets that there are in the problem input) consisting of 0s and 1s for sets that are included or excluded for that organism (that solution), respectively. Every organism can be evaluated for a fitness that denotes how valuable that organism is with respect to the rest of the population, and that fitness value is then used to decide how likely that organism is to be chosen as a parent that will generate offspring for the next generation of organisms (the new population), and this drives the logic behind the genetic algorithm.

The GA takes the input from the MIS - Phase I implementation as duplicates that solution for every organism in the population. Every organism is then mutated based on a mutation rate that can be specified as a parameter for that problem instance, and this generates the biodiversity of the population. Then, generations are simulated until a termination criteria is met. This can be user-specified by sending a `SIGINT` signal to the process, or can be specified as a timeout parameter, or can be specified as a value that is the number of generations to simulate in the GA. In every generation of organisms, we first compute each organism's fitness. This determines how likely an organism is to be picked as a parent during the offspring generation part of the loop. A pair of parents are picked based on fitness, and a new organism is created from those parents, where the new organism takes the bit from either parent 1 or parent 2 based on a 50% chance. This new offspring can then (1) be mutated as a mutation rate specified by a configuration parameter, (2) "minimized" at a specified rate in that sets in the organism that are included by means of other sets that contain the original set's vertices *and* cost more are removed, and (3) "ensured" in that the organism becomes feasible if currently-unfeasible based on its selected sets (its "genes", and also specified by a configuration parameter).

These factors ultimately influence the direction of the population under the simulation, which may lead to organisms that become more fit over time. Indeed, our preliminary results (see below) show that even though

there is a large factor of indeterminism in every run of the program, there is a tendency for the population to become more fit as mutations, minimizations, and ensurances all take place.

Algorithm 1 Phase I - Maximal Independent Set

```
1: S_out ← S
2: S_in ← ∅
3: V_out ← V
4: V_in ← ∅
5: Include in S_in all sets from S_out that contain a unique vertex
6: Remove from S_out all sets that were included in S_in
7: repeat
8:     for all sets s in S_out do
9:         if s = ∅ then
10:            Remove s from S_out
11:        end if
12:    end for
13:
14:    Initialize array v_array[1 ... |V|] of 64 bits
15:    // MS 32 bits: ∞, LS 32 bits: 2^32 − 1
16:
17:    for all sets s in S_out do
18:        Initialize val_s: 64 bits
19:        // MS 32 bits: cost(s) / num unaccounted vertices
20:        // LS 32 bits: id(s)
21:    end for
22:
23:    for all vertices v ∈ V_out do
24:        Run atomicMin over sets of v,
25:        Assign cheapest set's value to v_array[v]
26:    end for
27:
28:    for all sets s ∈ S_out do
29:        if all vertices of s chose s as cheapest then
30:            Include s in S_in
31:        end if
32:    end for
33:
34:    for all sets s ∈ S_in do
35:        for all vertices v in s do
36:            Add v to V_in
37:            Remove v from V_out
38:            Remove v from all sets that contain v
39:        end for
40:    end for
41: until V_in contains all vertices of V
```

5 Experimental Methodology

Ithaca Our test system "Ithaca" consisted of an AMD Ryzen Threadripper 2950X with 16 cores operating at 3.5 GHz, using `gcc` 13.3 and running the Fedora Linux 41 operating system. The node was also equipped with an NVIDIA RTX 4090 with 24 GB memory, which has 16,384 processing elements split amongst 128 streaming multiprocessors, and used `nvcc` 12.6. Other GPUs also used in this study are listed in Table 3.

Algorithm 2 Phase II - Genetic Algorithm

Require: Result from Phase I

// Generate initial population based on MIS result from Phase I

1: **for all** organisms **do**
2: Copy input organism
3: Mutate organism at initial mutation rate
4: **end for**

// Simulate population generations

5: **repeat**
6: **for all** organisms **do**
7: Compute fitness: $fitness_{org} \leftarrow (feasible_{org}?1:0) + \left(\frac{1}{\text{total set cost}}\right)$
8: **end for**
9: Add most-fit organism to the new population

// Generate remainder of new population from old population

10: **while** not enough new organisms **do**
11: Select parent pair by fitness
12: Generate offspring from parents

// Offspring selects set alleles from parent 1 or 2 (50% chance)

13: Mutate offspring at mutation rate
14: Minimize offspring at minimization rate
15: Ensure offspring at ensurance rate
16: Add offspring to new population
17: **end while**
18: **until** exit condition is met

Table 3: GPU Specifications

	PE	SM	PE/SM	Mem (GB)	NVCC
RTX 4090	16384	128	128	24	12.6
RTX 3090	10496	82	128	24	12.0
A100	6912	108	64	40	12.0
GTX 1650	1024	16	64	4	12.6

6 Results

Shown in Figure 3 is the Brute Force comparison of the Serial CPU implementation against the Parallel GPU implementation for randomly-generated SCP instances ranging from 10 bits up to 32 bits. Granted, we would normally convey this as an unfair hardware comparison, but here, we show simply how quickly the Brute Force approach becomes intractable, even for inputs as small as 32 vertices by 32 sets. We can see that at 27 bits, the serial CPU implementation crosses the boundary into over 100 seconds. By comparison, the parallel GPU implementation remains steady under 1 second for 32 bits. Unfortunately, as shown in Figure 4, the GPU Brute Force implementation can only be taken as high as roughly 10 more bits, where we cross the boundary of runtime into more than 100 seconds at 39 bits. Interestingly, our results show that the 40-bit SCP instance (40 vertices by 40 sets) ran faster than the 39-bit instance. We can mostly attribute this to the differences in set composition between our the randomly generated instances. To show this, we also randomly generated 10 SCP instances of 35 bits each—that is, we randomly generated SCP instances that contained 35 vertices and 35 sets. Shown in Figure 6, we can see that even though all our SCP instances are of the same size, their set compositions influence the runtimes. Some instances perhaps have sets that are more saturated with vertices than others, and this directly translates to more branching during runtimes due to the checks required to ensure all vertices have been accounted for.

In Figure 5, we show that the GPUs become saturated with set-processing at 30 bits, where 10- and 20-bit problem inputs don't yet reach saturation point. With 30 bits and above, throughput remains steady for all the instances leading up to 30 bits. Naturally, the RTX 4090 obtains the highest throughput, reaching more than 10 billion 35-bit combinations per second for the 37x37 SCP instance. Throughputs are shown for the 35x35 SCP instances in Figure 7, where set composition varies from SCP instance to SCP instance. With SCP instance 0fb9d12c, we reached a throughput of up to 25 billion 35-bit combinations per second.

Figure 3: Serial CPU vs. Parallel GPU

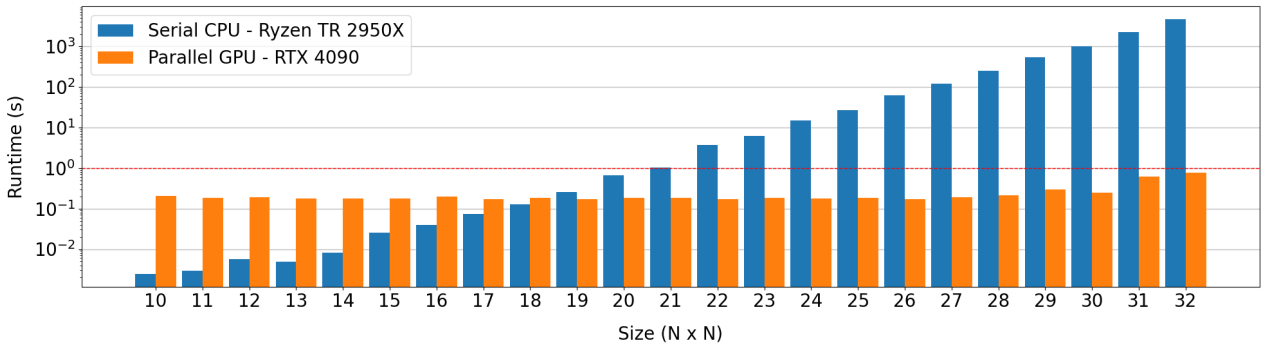


Figure 4: Runtimes (s) vs. SCP Size - Device Comparison

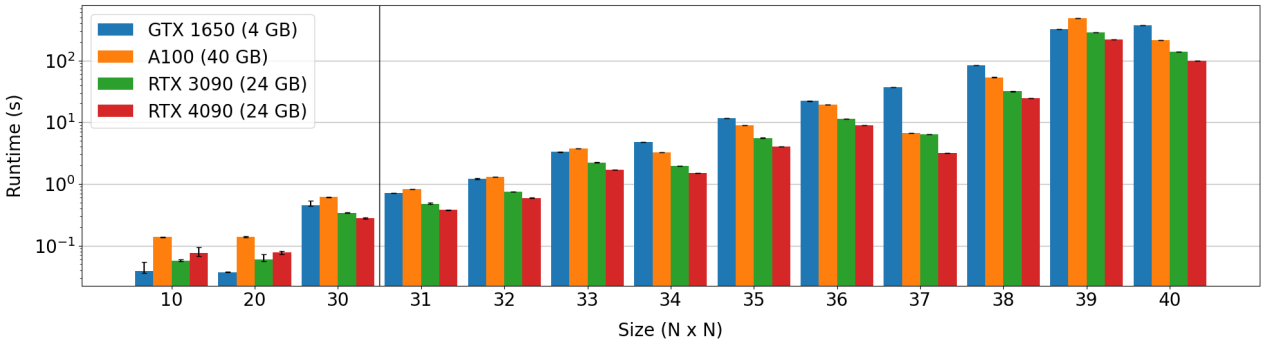


Figure 5: Throughput (combinations / sec) in billions - Device Comparison

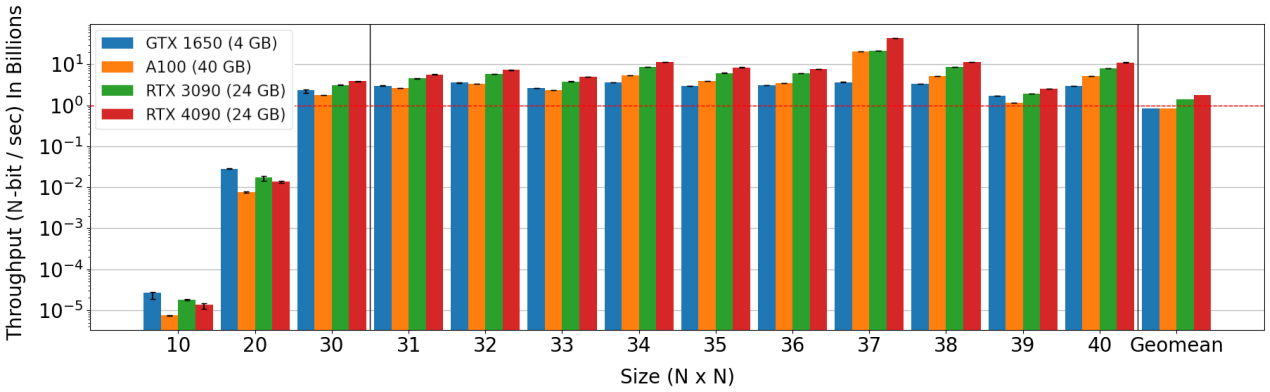


Figure 6: Runtimes (s) vs. SCP (35 x 35) - Device Comparison

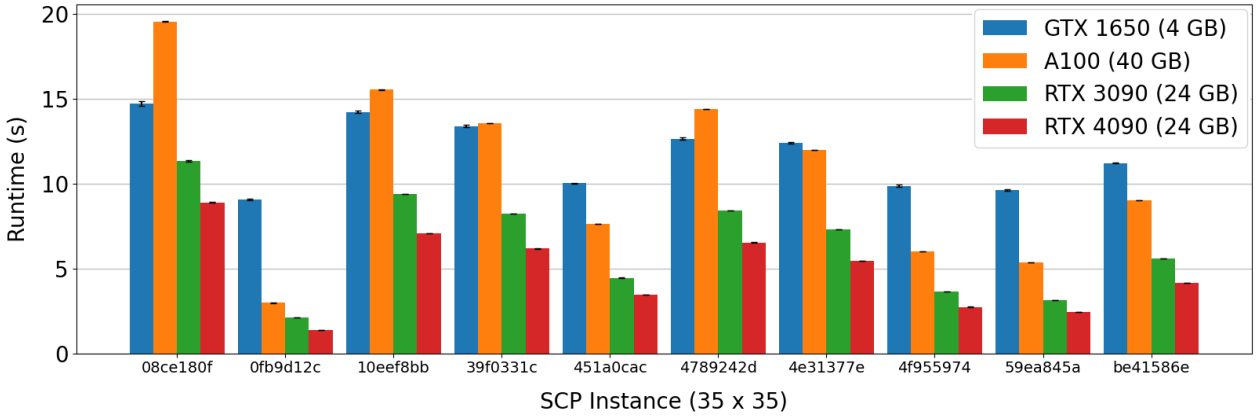
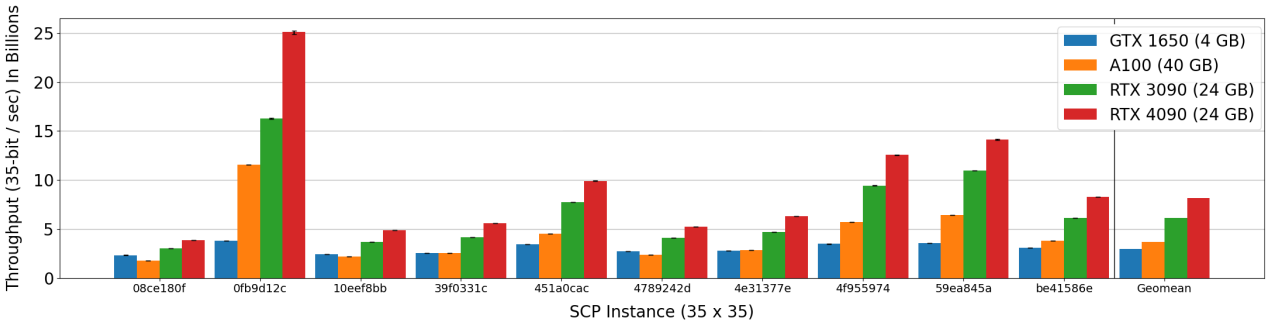


Figure 7: Throughput (combinations / sec) in billions - Device Comparison



When compared against 3rd party implementations CBC, SCIP, CYLP, and HiGHS, it is immediately evident that Brute Force approaches are not the most efficient. Shown in Figure 8, we have taken the same 10, 20, and 30-40 bit SCP instances and compared our fastest GPU implementation to the 3rd party open-source codes. It is interesting to see that cuOpt, which happens to use the Simplex Method in its implementation, is also an order of magnitude slower than the other open-source codes. The 40-bit SCP instances remain at a runtime of under 10 milliseconds for the non-GPU codes, whereas our Brute Force GPU code is beyond 100 seconds.

The same can be said with the 35x35 SCP instances in Figure 9—our Brute Force GPU implementation is more than a couple of orders of magnitude slower than even the slowest of the 3rd party codes. What's interesting to also see is that, even though the HiGHS implementation is what MATLAB chooses to use, a few of these 35x35 instances perform slightly worse for HiGHS than CyLP. Perhaps these SCP instances are too small in order to convey any meaningful comparison between the 3rd party codes, but they effectively highlight that our Brute Force implementation has room for algorithmic improvements.

Figure 8: Runtimes (s) vs. SCP Size - 3rd Party Comparison

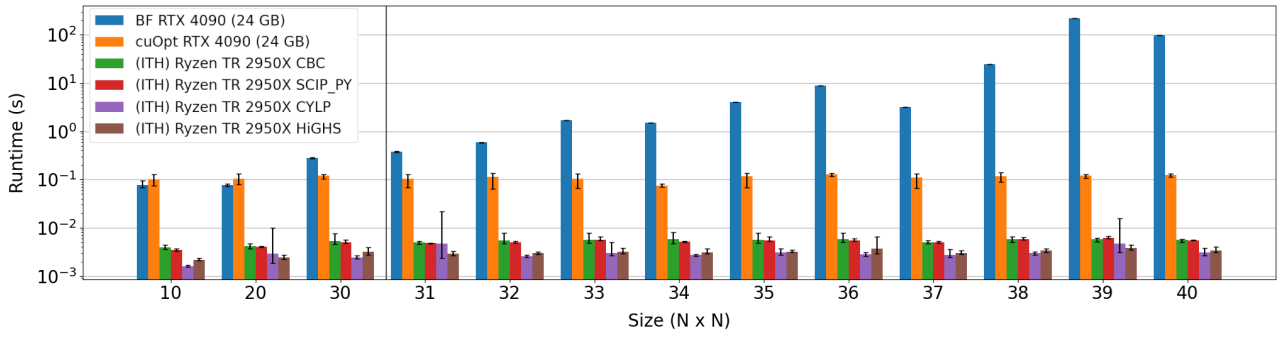


Figure 9: Runtimes (s) vs. SCP Size (35 x 35) - 3rd Party Comparison

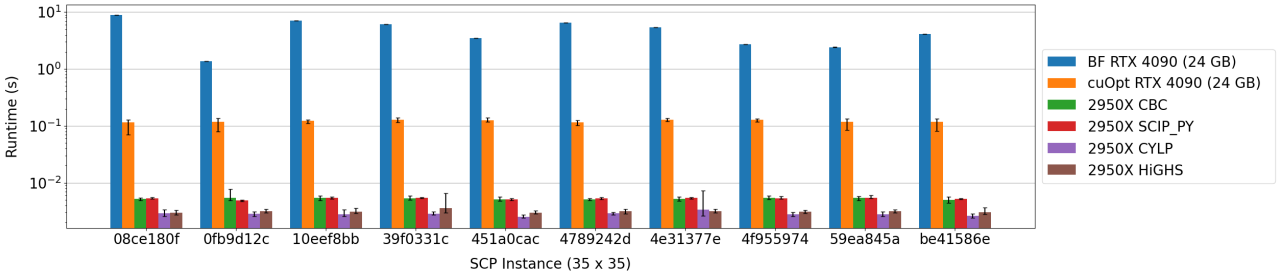
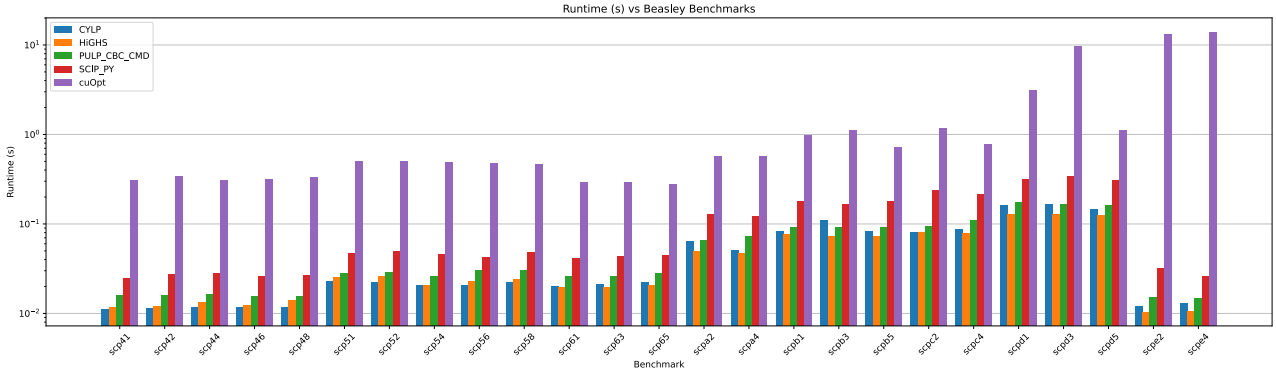


Figure 10: Runtimes (s) vs. Beasley SCPs - OR-Small - 3rd Party Comparison



Shown in Figures 10, 11, and 12 are the runtime comparisons between the 3rd-party codes on several of the Beasley benchmark instances, including the rail benchmarks. Given the mathematical approach that the 3rd-party codes take, the SCPs from the "OR-Small" and "OR-Large" categories run to optimal solution in a relatively-short amount of time. It is notable, however, that the "Rail" benchmarks have runtimes that are orders of magnitude slower. It is in these benchmarks (and that of the MIPLIB 2017-and-onward benchmarks) where implementation differences can be effectively assessed, mostly due to inputs of the smaller sizes reaching optimal solutions in a shorter amount of time.

Shown in Figure 13 is the result of running our MIS and GA implementation on the SCP benchmark SCP41. The x-axis denotes the current runtime of the algorithm and the y-axis shows the distance to the optimal solution. For SCP41, the MIS approach quickly arrives at a distance of 34 from the known optimal solution, and from that point onward, the rest of the improvements in solution quality have to do with the GA implementation carrying out and finding better solutions. This naturally paints a downward curve in the chart, where the more time progresses, the more the GA has the opportunity to find improvements in solutions. We depict five runs for the benchmark, which shows that even though the GA is nondeterministic in the manner in which it finds better solutions, indeed, the general trend is capable of finding better solution. The same is shown for Figure 14 for SCP42. In Figure 15, we show that across some more of our benchmarks, the trends remain.

An interesting point is that cuOpt is noticeably slower than the other implementations, and upon further inspection, the "optimal" solutions obtained from the CPU open-source implementations further show that the "optimal" solutions are not integral (see 4, where the optimal solution obtained by the 3rd-party solvers is shown

Table 4: Rail Benchmark Optimal Solutions and MIS Results

Problem Instance	Vertices	Sets	Optimal	Factor
Rail507	507	63009	172.14	1.24x
Rail516	516	47311	182.00	1.10x
Rail582	582	55515	209.71	1.19x
Rail2536	2536	1081841	688.39	1.28x
Rail2586	2586	92683	935.92	1.24x
Rail4284	4284	1092610	1054.05	1.31x
Rail4872	4872	968672	1509.63	1.26x

Figure 11: Runtimes (s) vs. Beasley SCPs - OR-Large - 3rd Party Comparison

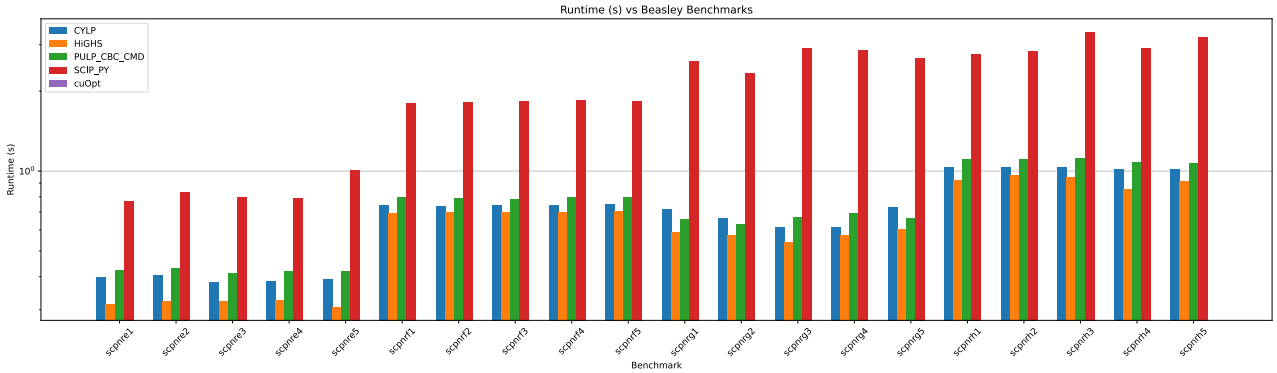


Figure 12: Runtimes (s) vs. Beasley Rail Benchmarks - 3rd Party Comparison

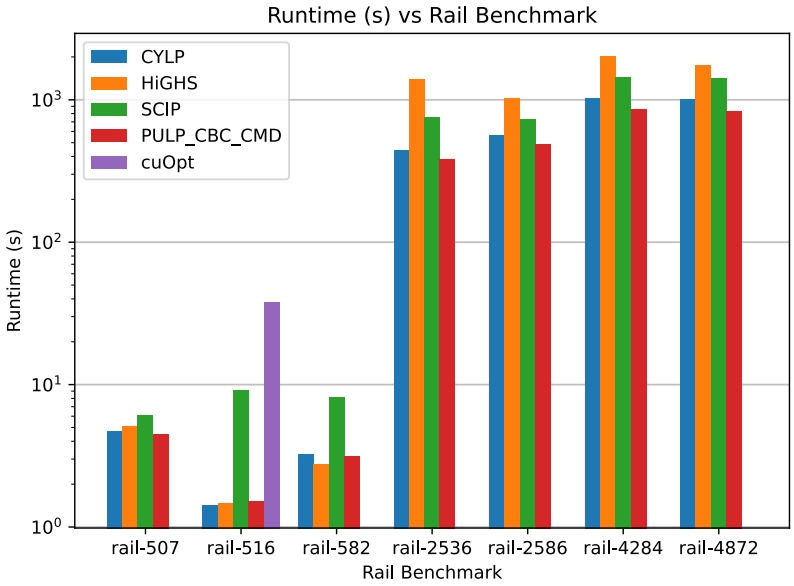
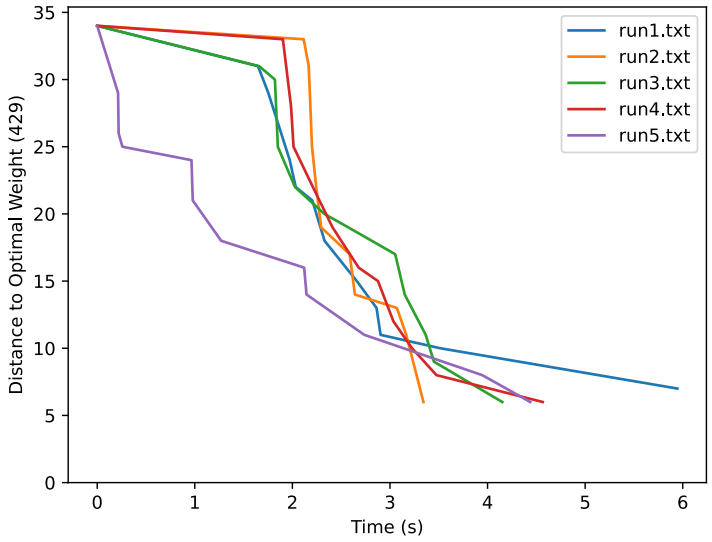


Figure 13: Distance to Optimal vs. Time - SCP41



along the factor of the solution obtained from the MIS phase of our implementation). One might wonder why this is, but the explanation lies in the fact that the open-source solvers are taking an approach to solving the SCP by means of relaxing the binary-integer constraint in the LP formulation. This LP relaxation allows for the solution search-space to become a continuum, wherein values for the variables denoting set-inclusion can take values between 0 and 1 inclusive. This approach to relaxing the constraint is one that is often taken advantage of in implementations due to the comparable solution quality given the faster computations. Rightly-so, keeping the LP formulation as that of binary integer 0-1 is NP-Hard, which raises the complexity of the problem and computations required for finding *the* (as opposed to *a*) optimal solution. In our preliminary results, the GA was unable to improve upon the solution obtained by the MIS—we list the factor of the MIS results because of this.

7 Discussion and Future Work

Our MIS and GA approach has shown that the implementation takes us a considerable distance towards finding a solution that is of good quality, but unfortunately, it does not yet provide for matching against optimal solutions. Several strategies remain to be implemented that could ultimately contribute towards that goal.

For example, there is the idea that much in the same way that organisms can have a calculated fitness, the alleles themselves can also have a fitness value [18]. Stated another way, perhaps there is a tendency for a particular set to make an appearance in organisms that are consistently more fit, meaning that perhaps that set is quantifiably-valuable. Is there a chance that the optimal solution contains that set due to the fact that the more-fit organisms have that set?

We have also yet to take into account the potential for established lower and upper bounds on the solutions. For example, we can establish a lower by sorting the sets based on size, add incrementally-adding sets until the sets' number of vertices have summed to a number that is greater than the number of vertices there actually are in the input. This means that we would need *at least* that many sets in our solution, giving us the possibility of discarding solutions that contain less than that threshold. We can also establish an upper bound on the implementation by taking any randomly-generated feasible solution—the thinking being that any randomly-generated solution is highly-likely neither the optimal solution nor a solution of good quality. This means that we can then only take into considerations that perhaps cost less or contain a smaller number of sets.

Another noticeable detail is that in the results seen so far, most of the end-solutions, when compared to the solution generated by the MIS phase, tend to be solutions that have simply been removal of sets that are redundant. Perhaps these redundant sets can be removed as a GA-preprocessing step instead of having to wait for them to be removed as part of the evolutionary process.

Furthermore, the implementation does not yet take into account the idea that we can process each input on a per-connected-component basis. Although the SCP input can (and is currently) processed as one component,

Figure 14: Distance to Optimal vs. Time - SCP42

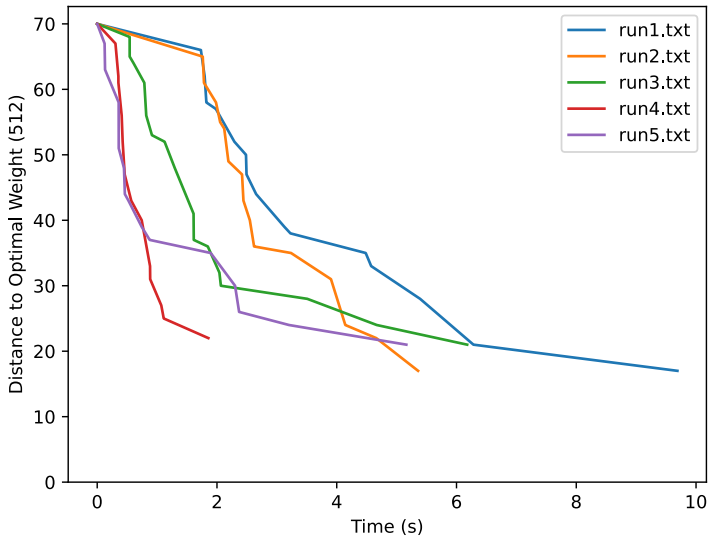
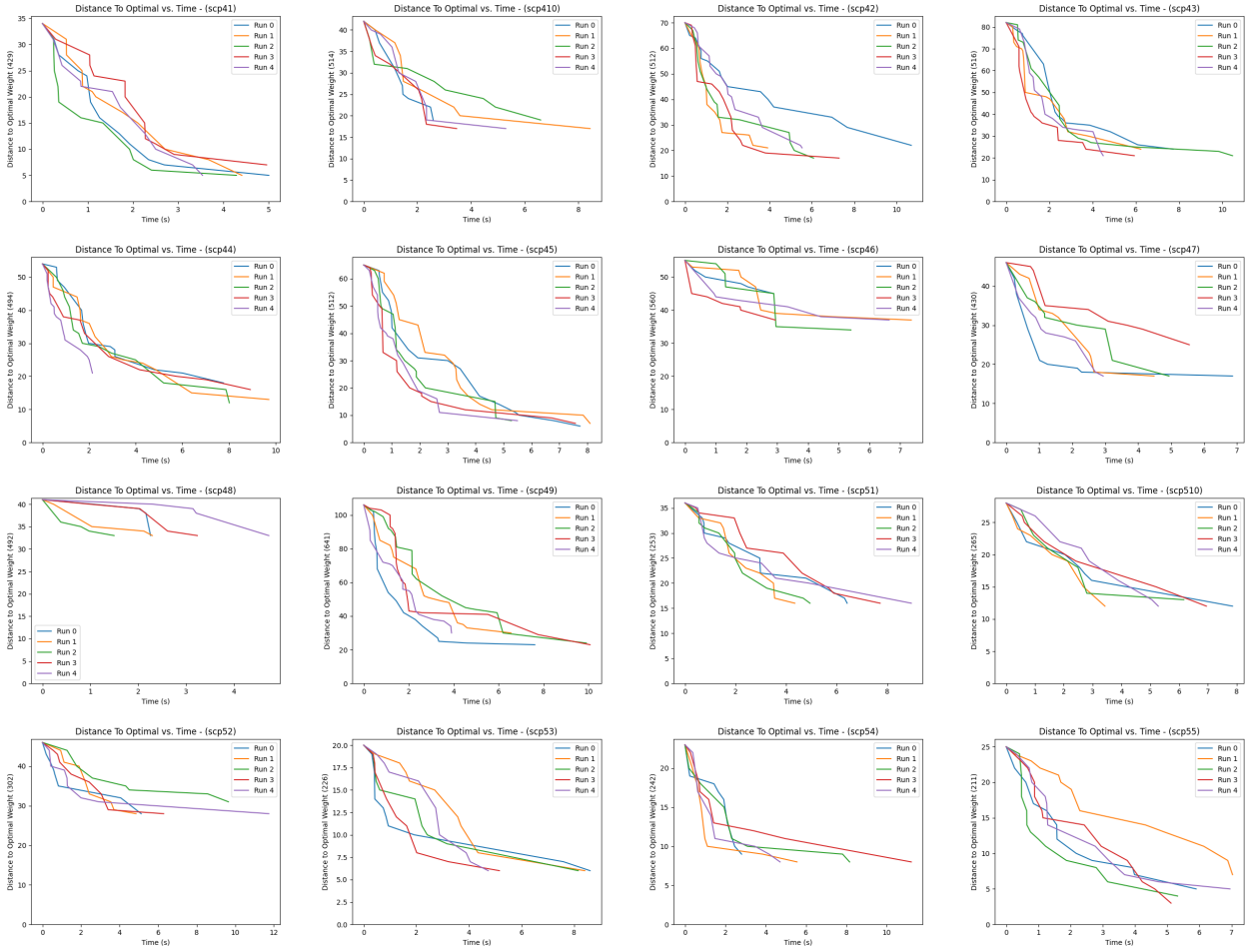


Figure 15: Distance to Optimal vs. Time



problem sizes may be reduced to the point that perhaps an exhaustive brute force is applicable to that smaller connected-component. The solution to that smaller component can then be concatenated with the solutions from the other components.

The work in this study focused on open-source codes, but the companies that have the closed-source codes do offer academic licenses (such as Mosek). It would be worth looking into the performance of these closed-source implementations so as to compare their runtimes and solution qualities to the open-source implementations.

Lastly, we are looking to apply this implementation to problem instances of more than 5,000 vertices and 1,000,000 sets. These are the input sizes that tend to timeout after 2 hours (HiGHS) on the open-source implementations.

8 Summary and Conclusions

In this study, we presented `ECL-SCP`, a parallel GPU Maximal Independent Set and Genetic Algorithm implementation, capable of getting within 1 unit of cost from the established-optimal within seconds. Additionally, obtained solutions can match 97.7% of the optimal solutions found by solvers. These preliminary results have shown promise in that we can obtain quality solutions through our heuristic, though the question of optimality remains. Brute-force approaches quickly become untenable even with the smallest of inputs (e.g., SCP inputs consisting of only 40 vertices and 40 sets) as exhaustively-searching runtimes exceed 100 seconds. We aim to apply our fast MIS and GA implementation to larger inputs consisting of more than 5,000 vertices and 1,000,000 sets, and not only does our implementation obtain good quality results, it can get them fast with a GPU.

References

[1] Zahi Ajami and Sara Cohen. *Enumerating Minimal Weight Set Covers*. Generic. 2019. DOI: 10.1109/ICDE.2019.00053.

[2] Eduardo Álvarez-Miranda et al. "The Generalized Reserve Set Covering Problem with Connectivity and Buffer Requirements". In: *European Journal of Operational Research* 289.3 (2021), pp. 1013–1029. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2019.07.017.

[3] Egon Balas, Andrew Ho, and Rainer Burkard. "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study". In: *Mathematical Programming Study* No. 12 (1980), pp. 37–60. ISSN: 0303-3929 (print).

[4] Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. *A constant factor approximation algorithm for generalized min-sum set cover*. Generic. 2010. DOI: 10.5555/1873601.1873726.

[5] Nikhil Bansal et al. "ON MIN SUM VERTEX COVER AND GENERALIZED MIN SUM SET COVER". In: *SIAM JOURNAL ON COMPUTING* 52.2 (2023), pp. 327–357. ISSN: 00975397 10957111. DOI: 10.1137/21M1434052.

[6] F. Barriga-Gallegos, A. Luer-Villagra, and G. Gutierrez-Jarpa. *The Angular Set Covering Problem*. Generic. 2024. DOI: 10.1109/ACCESS.2024.3416871.

[7] Joaquín Bautista and Jordi Pereira. "A GRASP algorithm to solve the unicost set covering problem". In: *Computers & Operations Research* 34.10 (2007), pp. 3162–3173. ISSN: 0305-0548.

[8] J. E. Beasley. "An algorithm for set covering problem". In: *European Journal of Operational Research* 31.1 (1987), pp. 85–93. ISSN: 03772217. DOI: 10.1016/0377-2217(87)90141-X.

[9] Aleksander Belykh et al. "Efficient heuristics for a partial set covering problem with mutually exclusive pairs of facilities". In: *Optimization, simulation and control*.

[10] Nehme Bilal, Philippe Galinier, and Francois Guibault. "An iterated-tabu-search heuristic for a variant of the partial set covering problem". In: *Journal of Heuristics* 20.2 (2014), pp. 143–164. ISSN: 13811231 15729397. DOI: 10.1007/s10732-013-9235-9.

[11] Suresh Bolusani et al. *The SCIP Optimization Suite 9.0*. Technical Report. Optimization Online, Feb. 2024. URL: https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

[12] Suresh Bolusani et al. *The SCIP Optimization Suite 9.0*. ZIB-Report 24-02-29. Zuse Institute Berlin, Feb. 2024. URL: https://nbn-resolving.org/urn:nbn:de:0297-zib-95528.

[13] Valentin E. Brimkov et al. "Approximation algorithms for a geometric set cover problem". In: *Discrete Applied Mathematics* 160.7-8 (2012), pp. 1039–1052. ISSN: 0166-218X. DOI: 10.1016/j.dam.2011.11.023.

[14] Brian D. Bunday. *Basic linear programming / Brian D. Bunday*. Edward Arnold, 1984. ISBN: 0713135093.

[15] Francesco Carrabs et al. "Solving the Set Covering Problem with Conflicts on Sets: A new parallel GRASP". In: *Computers and Operations Research* 166 (2024). ISSN: 0305-0548. DOI: 10.1016/j.cor.2024.106620.

[16] Kenneth L. Clarkson and Kasturi Varadarajan. *Improved approximation algorithms for geometric set cover*. Generic. 2005. DOI: 10.1145/1064092.1064115.

[17] Broderick Crawford et al. "Constructive metaheuristics for the set covering problem". In: *International Conference on Bioinspired Methods and Their Applications*. Springer, pp. 88–99.

[18] Yuval Davidor. "Epistasis variance: Suitability of a representation to genetic algorithms". In: *Complex Systems* 4.4 (1990), pp. 369–383.

[19] Jorge Delgado, Héctor Ferrada, and Cristóbal A. Navarro. "A succinct and approximate greedy algorithm for the Minimum Set Cover Problem". In: *Journal of Computational Science* 81 (2024). ISSN: 1877-7503. DOI: 10.1016/j.jocs.2024.102378.

[20] Jens Dorpinghaus, Carsten Duing, and Vera Weil. *A Minimum Set-Cover Problem with several constraints*. Generic. 2019. DOI: 10.15439/2019F2.

[21] Dimitris Fotakis et al. "On the Approximability of Multistage Min-Sum Set Cover". In: *arXiv preprint arXiv:2107.13344* (2021).

[22] Ambros Gleixner et al. "MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library". In: *Mathematical Programming Computation: A Publication of the Mathematical Optimization Society* 13.3 (2021), pp. 443–490. ISSN: 18672949 18672957. DOI: 10.1007/s12532-020-00194-3.

[23] Kathleen E. Hamilton and Travis S. Humble. "Identifying the minor set cover of dense connected bipartite graphs via random matching edge sets". In: *Quantum Information Processing* 16.4 (2017), pp. 1–17. ISSN: 15700755 15731332. DOI: 10.1007/s11128-016-1513-7.

[24] Q. Huangfu and J. A. J. Hall. "Parallelizing the dual revised simplex method". In: *Mathematical Programming Computation: A Publication of the Mathematical Optimization Society* 10.1 (2018), pp. 119–142. ISSN: 18672949 18672957. DOI: 10.1007/s12532-017-0130-5.

[25] Yabuta Kenichi and Kitazawa Hitoshi. *Optimum camera placement considering camera specification for security monitoring*. Generic. 2008. DOI: 10.1109/ISCAS.2008.4541867.

[26] Jochen Könemann, Ojas Parekh, and Danny Segev. "A Unified Approach to Approximating Partial Covering Problems". In: *Algorithmica* 59.4 (2011), pp. 489–509. ISSN: 01784617 14320541. DOI: 10.1007/s00453-009-9317-0.

[27] C. Luo et al. *NuSC: An Effective Local Search Algorithm for Solving the Set Covering Problem*. Generic. 2024. DOI: 10.1109/TCYB.2022.3199147.

[28] Zahra Naji-Azimi, Paolo Toth, and Laura Galli. "An electromagnetism metaheuristic for the unicost set covering problem". In: *European Journal of Operational Research* 205.2 (2010), pp. 290–300. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2010.01.035.

[29] Nicholas R. Paul, Brian J. Lunday, and Sarah G. Nurre. "A multiobjective, maximal conditional covering location problem applied to the relocation of hierarchical emergency response facilities". In: *Omega* 66 (2017), pp. 147–158. ISSN: 0305-0483. DOI: 10.1016/j.omega.2016.02.006.

[30] Luciana S. Pessoa, Mauricio G. C. Resende, and Celso C. Ribeiro. "Experiments with LAGRASP heuristic for set ¡i¿k¡/i¿-covering". In: *Optimization Letters* 5.3 (2011), pp. 407–419. ISSN: 18624472 18624480. DOI: 10.1007/s11590-011-0312-4.

[31] Yingli Ran, Yaoyao Zhang, and Zhao Zhang. "A parallel algorithm for minimum weight set cover with small neighborhood property". In: *Journal of Parallel and Distributed Computing* 198 (2025). ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2025.105034.

[32] Saeed Saffari and Yahya Fathi. "Set covering problem with conflict constraints". In: *COMPUTERS & OPERATIONS RESEARCH* 143 (2022), p. 105763. ISSN: 03050548 1873765X. DOI: 10.1016/j.cor.2022.105763.

[33] Amir Salehipour. "A heuristic algorithm for the set k-cover problem". In: *International Conference on Optimization and Learning*. Springer, pp. 98–112.

[34] Shay Solomon, Amitai Uzrad, and Tianyi Zhang. *A Lossless Deamortization for Dynamic Greedy Set Cover*. Generic. 2024. DOI: 10.1109/FOCS61266.2024.00025.

[35] Claudio Valenzuela et al. “A 2-level Metaheuristic for the Set Covering Problem”. In: *International Journal of Computers, Communications & Control* 7.2 (2012), pp. 377–387. ISSN: 18419836.

[36] Yiyuan Wang et al. “A restart local search algorithm for solving maximum set ¡i¿k¡/i¿-covering problem”. In: *Neural Computing and Applications* 29.10 (2018), pp. 755–765. ISSN: 09410643 14333058. DOI: 10.1007/s00521-016-2599-7.