State of the art - Hitting Set

Ana-Delia Manoliu, Alexandru-Cristian Grăjdeanu

May 9, 2025

1 Introduction

The **Hitting Set Problem** is a well-known problem in computer science and combinatorics that plays a significant role in the study of computational complexity. It is classified as an NP-complete problem, which means that it is in both the NP class (nondeterministic polynomial time) and as "hard" as any other problem in NP.

A problem is NP-complete if:

1. It is in the NP class: A proposed solution to the problem can be verified quickly (in polynomial time) by a deterministic algorithm;
2. It is NP-hard: Every problem in NP can be reduced to it in polynomial time. This means that if you can solve the NP-complete problem efficiently, you can solve all problems in NP efficiently.

1.1 W-hierarchy

The W hierarchy is a concept in computational complexity theory, which is a branch of computer science that studies how difficult problems are to solve.
The W hierarchy is a way to categorize problems based on their fixed-parameter tractability. It consists of levels like W[1], W[2], W[3], and so on, with each level representing a higher degree of complexity. Here's a simple breakdown:

1. W[0]: Problems that are fixed-parameter tractable (FPT). These are problems that can be solved efficiently if the parameter is fixed.
2. W[1]: Problems that are believed to be harder than those in W[0]. A classic example is the k-Clique problem (finding a clique of size k in a graph).
3. W[2]: Problems that are even harder, such as the Dominating Set problem (finding a set of k vertices that "dominate" all other vertices in a graph) or Hitting Set problem.

2 Problem formulation

Algorithm 1 Hitting Set Problem

Input: A set system / hypergraph $\mathcal{S}$
Output: A hitting set for $\mathcal{S}$. A hitting set is a set $H \subseteq V(G)$ such that every set $S \in E(\mathcal{S})$ contains at least one vertex from H, that is, for all $S \in E(\mathcal{S})$ we have $S \cap H = \emptyset$

3 Algorithms

3.1 Deterministic algorithm

Theorem 1 *Let $\Sigma = (S_1, ..., S_n)$ where S_i is a subset of $V = \{1, ..., n\}$ of size $|S_i| \geq R$. There is a deterministic algorithm that runs in O(nR) time and finds a subset $S \subseteq V$ with $|S| \leq (n/R) \ln n$ and $S \cup S_i \neq \emptyset$ for all i.*

The algorithm uses a greedy approach, starting with an empty set S and iteratively adding elements to S based on their frequency of appearance in the subset of Σ, and achieves the desired runtime by using a data structure that supports insertions, maximum value queries, and decrements in $O(\log n)$ time.

3.2 Randomized algorithm

Theorem 2 *Let $\Sigma = (S_1, ..., S_n)$ with each S_i a subset of $V = \{1, ..., n\}$ of size $|S_i| \geq R$. For any constant $C > 0$, there is a randomized algorithm that runs in time $O(n)$ and finds a subset $S \subseteq V$ with $|S| \leq (n(1+c)/R) \log n$, such that $S \cup S_i \neq \emptyset$ for all i holds with probability $\geq 1 - n^{-c}$. The algorithm does not need to know Σ.*

Both algorithms provide a solution to the hitting set problem, but the randomized algorithm has a faster runtime and a higher probability of success, while the deterministic algorithm provides a guaranteed bound on the size of the hitting set.

The probability of a set S missing a particular subset S_i is calculated using formula

$$(1 - s/n)^{|S_i|} \leq (1 - (C/R) \ln n)^R \leq n^{-C} = n^{-1-c} \tag{1}$$

and taking the union bound over Σ proves that S fails to be a hitting set with probability at most n^{-c}.

To return a hitting set of exactly the correct size, a random subset of V of size s is chosen and the probability for S to miss a particular subset S_i is calculated using the formula

$$\prod_{j=1}^{s} \frac{n - R - (j-1)}{n - (j-1)} \leq (1 - R/n)^s \leq n^{-C} \leq n^{-1-c} \tag{2}$$

The use of a randomized algorithm to obtain a hitting set can lead to a faster time algorithm of $O(m\sqrt{n})$ compared to the deterministic approach, which has a time complexity of $O(m\sqrt{n} + n^2)$, although the randomized algorithm may fail to obtain a good estimate with a small probability.

3.3 Mapping onto Boolean Satisfiability Problem

In the article *New Approaches for Efficient Solution of the Hitting Set Problem*, a mapping onto the Boolean Satisfiability Problem is presented. The problem is modeled using a hypergraph matrix representation, where $\mathcal{S} = S_1, ...S_m\}$ is a collection of nonempty subsets, and $M = \{m_1, ..., m_n\}$ is the set of elements. Each entry in matrix a_{ij} indicates whether element m_j is present in subset S_i.

	m_1	m_2	$\cdots$	m_n
S_1	1	0	$\cdots$	0
S_2	0	1	$\cdots$	1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
S_m	1	1	$\cdots$	0

The authors introduce Boolean variables $x_1, x_2..., x_n$ where each variable x_j represents the element m_j. For each subset $S_i = \{m_{j1}, m_{j2}, ...m_{jn}\}$, they construct a disjunction.

$$F_i = x_{j_1} \vee x_{j2} \vee ... \vee x_{jn_i} \tag{3}$$

Consequently, the problem is mapped to a CNF (Conjunctive Normal Form) formula, where each hitting set of the system $\mathcal{S}$ corresponds with a satisfying truthassignment for the CNF F_s, and vice versa.

$$F_{\mathcal{S}} = F_1 \wedge F_2 \wedge ... \wedge F_m \tag{4}$$

4 Benchmark instances

To test potential implementations of the Hitting Set problem, a dataset from the *PACE Challenges* will be used. The data consists of DIMACS-format files, specifically **.hgr** files for representing hypergraphs and **.sol** files for verifying whether the solution produced by the implementation matches the official solution.

Below is an example of a .hgr file and a .sol file.

`c I am a comment`
`p hs 6 5`
`1 2`
`2 3 4`
`2 4 5`
`1 3 6`
`5 6`
	`c The first non-comment line`
	`represents the solution size`
	`2`
	`2`
	`6`

.hgr file	**.sol file**

Table 1: Example of a .hgr file and a .sol file

5 Implementation and Performance Analysis

5.1 Implementation Details

The Hitting Set problem was tackled using two distinct methods: a greedy heuristic and an exact approach leveraging SAT solving.
A Hypergraph class encapsulates the structure and behavior of the input hypergraph. Input files follow the .hgr format, with vertices and edges defined as per standard conventions.

- Greedy Hitting Set Solver

  This method iteratively selects the vertex that covers the largest number of uncovered edges. After choosing a vertex, all edges it hits are removed. This process continues until all edges are hit.

- SAT Hitting Set Solver

  This method encodes the Hitting Set problem into CNF (conjunctive normal form) and solves it using the MiniSAT solver from the PySAT library. A binary search strategy is used to find the minimum size of the hitting set. For each potential size bound, it checks for satisfiability under the constraint of selecting at most k vertices. Cardinality constraints are encoded using a sequential counter

5.2 Evaluation Metrics

Table 2: Comparison of Greedy and SAT Hitting Set Algorithms on Bremen Subgraphs

Graph	Greedy Size	SAT Size	Greedy Time	SAT Time	Best Method
subgraph_5.hgr	2	2	0.00	0.03	Greedy
subgraph_20.hgr	10	9	0.00	0.07	SAT
subgraph_50.hgr	19	17	0.00	1.22	SAT
subgraph_100.hgr	34	29	0.00	5.92	SAT
subgraph_150.hgr	49	43	0.00	646.44	SAT
subgraph_200.hgr	65	58	0.00	803.88	SAT
subgraph_250.hgr	81	78	0.01	1004.94	SAT
subgraph_300.hgr	98	90	0.01	1281.45	SAT

Each method was evaluated based on the following metrics:

- Hitting Set Validity: Whether the solution is a valid hitting set.
- Hitting Set Size: The number of elements in the hitting set (smaller is better)
- Execution Time: Time taken to compute the hitting set.
- Best Method: The method yielding the smaller valid hitting set.

Greedy Approach

- Advantages
  - Extremely fast execution time
  - Produces valid hitting sets.

- Disadvantages
  - Produces larger hitting sets, especially on larger or denser graphs.
  - Never found the optimal or best set size except trivial cases.

SAT Approach

- Advantages
  - Consistently found smaller (and likely optimal) hitting sets in all non-trivial instances.
  - More effective in minimizing solution size.
- Disadvantages
  - Significantly slower, especially on larger graphs
  - May not scale well for very large instances without optimization.

References

[1] PACE Challenge. Hitting Set. Available at: https://pacechallenge.org/2025/hs/.

[2] CS 267 Lecture 5. Retrieved from: https://theory.stanford.edu/~virgi/cs267/lecture5.pdf.

[3] New approaches for efficient solution of hitting set problem. Retrieved from: https://www.researchgate.net/publication/242106428_New_approaches_for_efficient_solution_of_hitting_set_problem.