Project presentation

Formal Methods for System Verification

Indian Institute of Technology, Guwahati

Authors: Bhogi Sai Sathwik, Bussa Sai Santhosh, Mukka Koushik, Usha Sree Kella, Pratyush R

July-November 2024

Problem 1 - Graph Coloring

Problem: Given a undirected graph $G = (V, E)$ and the number of colors d, find a coloring of the vertices, such that any two vertices connected by an edge do not get the same color using atmost d colors.
More formally,
Given graph $G = (V, E)$, and $C = \{c_1, c_2, \ldots, c_d\}$ be the set of d colors. Find a function $\phi : V \rightarrow C$, such that $\forall (v_i, v_j) \in E, \phi(v_i) \neq \phi(v_j)$.

Formulation of SAT Encoding

We let $|V| = n$ and so, vertices are numbered as $v_1, \ldots, v_n$
We introduce the variables:

$$v_{i,j} = \begin{cases} \top, & \text{if vertex } v_i \text{ is colored with color } j \\ \bot, & \text{otherwise} \end{cases}$$

$\forall 1 \leq i \leq n, 1 \leq j \leq d$
From the definition itself, we can arrive at the following two constraints:

- Every vertex must have atleast one color.
- Adjacent vertices must have different color.

Formulation of SAT Encoding - Contd.

The above two can be encoded as follows:

- $\bigvee_{j=1}^{d} v_{i,j} \quad \forall 1 \leq i \leq n$
- $\bigwedge_{k=1}^{d} \overline{v_{i,k} \wedge v_{j,k}} \quad \forall (v_i, v_j) \in E \equiv \bigwedge_{k=1}^{d} \overline{v_{i,k}} \vee \overline{v_{j,k}} \quad \forall (v_i, v_j) \in E$

So overall encoding is:

$$\Phi = \left(\bigwedge_{i=1}^{n} \bigvee_{j=1}^{d} v_{i,j} \right) \bigwedge \left(\bigwedge_{(v_i,v_j) \in E} \bigwedge_{k=1}^{d} \overline{v_{i,k}} \vee \overline{v_{j,k}} \right)$$

Formulation of SAT Encoding - Extras

We can even add constraints, on the already obtained Φ to limit the number of vertices colored in one of the colors to be some upper limit r.
Here, we show two encoding, which ensure that the above constraint is met.
In our implementation, we take color 1 to be constrained.
Formally, this is expressed as $v_{1,1} + v_{2,1} + \ldots + v_{n,1} \leq r$

Formulation of SAT Encoding - Extras

The following encoding is by C. Sinz, [LNCS 3709 (2005), 827–831]:
Introduce:

$$s_j^k = \begin{cases} \top, & \sum_{i=1}^{j+k-1} v_{i,1} \geq k \\ \bot, & \text{otherwise} \end{cases} \qquad \forall 1 \leq j \leq n-r, 1 \leq k \leq r$$

Then final encoding will be $\Phi' = \Phi \wedge \Phi_1$, where Φ_1 is:

$$\Phi_1 = \left(\bigwedge_{k=1}^{r} \bigwedge_{j=1}^{n-r-1} \overline{s_j^k} \vee s_{j+1}^k \right) \bigwedge \Phi_2$$

$$\Phi_2 = \left(\Phi_3 \bigwedge \Phi_4 \bigwedge \Phi_5 \right)$$

Formulation of SAT Encoding - Extras

$$\Phi_3 = \left(\bigwedge_{j=1}^{n-r} \overline{v_{j,1}} \vee s_j^1 \right)$$

$$\Phi_4 = \left(\bigwedge_{k=1}^{r-1} \bigwedge_{j=1}^{n-r} \overline{v_{j+k,1}} \vee \overline{s_j^k} \vee s_j^{k+1} \right)$$

$$\Phi_5 = \left(\bigwedge_{j=1}^{n-r} \overline{v_{j+r,1}} \vee \overline{s_j^r} \right)$$

We refer to Φ_1 as *Sinz's constraints*.

Explanation of Sinz's Constraints

We tabulate the expressions and their equivalent forms and a short justification of why the constraint should be imposed. Together, these constraints make the normal inequality algebra enforced for the new variables, with the last one (on next slide) imposing the required constraint.

Expression	Equivalent	Justification
$\overline{s_j^k} \vee s_{j+1}^k$	$s_j^k \rightarrow s_{j+1}^k$	$s_j^k = \top \Rightarrow \sum_{i=1}^{j+k-1} v_{i,1} \geq k \Rightarrow \sum_{i=1}^{j+k} v_{i,1} \geq k$, regardless of $v_{j+k,1} = \top$ or $\bot$
$\overline{v_{j,1}} \vee s_j^1$	$v_{j,1} \rightarrow s_j^1$	$v_{j,1} = \top \Rightarrow \sum_{i=1}^{j} v_{i,1} \geq 1$ as last variable in the sum is $\top$.

Table: Short justification for the expressions used in constraint.

Explanation of Sinz's Constraints, continued.

Expression	Equivalent	Justification
$\overline{v_{j+k,1}} \vee \overline{s_j^k} \vee s_j^{k+1}$	$v_{j+k,1} \wedge s_j^k \rightarrow s_j^{k+1}$	$v_{j+k,1} = \top \wedge \sum_{i=1}^{j+k-1} v_{i,1} \geq k \Rightarrow \sum_{i=1}^{j+k} v_{i,1} \geq k+1$ as last variable added in the sum is $\top$.
$\overline{v_{j+r,1}} \vee \overline{s_j^r}$	$v_{j+r,1} \rightarrow \overline{s_j^r}$	$v_{j+r,1} = \top \Rightarrow \sum_{i=1}^{j+r-1} v_{i,1} \leq r$, which is required, and is not trivially satisfied for $> r$ variables.

Table: Short justification for the expressions used in constraint.

Formulation of SAT Encoding - Extras

Another encoding for the constraint was given by O. Bailleux and Y. Boufkhad [LNCS 2833 (2003), 108−122]:
We have a complete binary tree with internal nodes numbered $1, 2, \ldots, n-1$, and n leaves numbered $n, n+1, \ldots, 2n-1$.
We define:

$$\begin{aligned}
\text{leftChild}(k) &= 2k && \forall 1 \leq k < n \\
\text{rightChild}(k) &= 2k+1 && \forall 1 \leq k < n \\
t_k &= 1 && \forall n \leq k < 2n \\
t_k &= \min(r, t_{\text{leftChild}(k)} + t_{\text{rightChild}(k)}) && \forall 1 \leq k < n
\end{aligned}$$

Formulation of SAT Encoding - Extras

$$b_j^k = \begin{cases} \top, & \sum_{i \in \text{Subtree}(k)} v_{i,1} \leq j \\ \bot, & \text{otherwise} \end{cases} \quad \forall 1 < k < n, 1 \leq j \leq t_k$$

$$b_1^k = v_{k-n+1,1} \quad \forall n \leq k < 2n$$

$$\Phi_6 = \bigwedge_{i+j=r+1, 0 \leq i \leq t_2, 0 \leq j \leq t_3} \overline{b_i^2} \vee \overline{b_j^3}$$

$$\Phi_7 = \bigwedge_{1<k<n} \bigwedge_{\substack{0 \leq i \leq t_{2k} \\ 0 \leq j \leq t_{2k+1} \\ 1 \leq i+j \leq t_k+1}} \overline{b_i^{2k}} \vee \overline{b_j^{2k+1}} \vee b_{i+j}^k$$

With omission: $\overline{b_0^k}, b_{r+1}^k$

Formulation of SAT Encoding - Extras

$$\Phi_8 = \Phi_6 \wedge \Phi_7$$

The final formula is $\Phi'' = \Phi \wedge \Phi_8$
We refer to Φ_8 as *Tree-type constraints*.

Implementation and Testing

- We implemented the clauses for the McGregor's graph of order 10 and tested against the answers as stated in `TAOCP, Volume 4, Fascicle 6, Section 7.2.2.2, Pages 6-8`.
- A recursive procedure was used to generate the graph.
- Variable numbering, for clauses with no extra constraints: The vertex numbered from 1 to n, were, for d colors, changed from $i \in \{1, 2, \ldots, n\}$ to $(i-1)d+1$, thereby allowing color $j \in \{1, 2, \ldots, d\}$ in vertex i to be mapped to $(i-1)d+j$.

Implementation and Testing - Contd.

- Above scheme was also used for *Sinz's constraints*, difference being, new variables were given numbers after the original ones and s_j^k was mapped to $offset + (j-1)r + k$.
- For the *Tree-type constraints*, we first calculated $t_k \ \forall 1 \leq k \leq 2n-1$, with the new variable numbers starting after original ones, we sequentially assigned the numbers starting from vertex 1, numbers $offset + 1, \ldots, offset + t_1$, vertex 2, numbers $offset + t_1 + 1, \ldots, offset + t_1 + t_2$, and so on.
- After the numbering, the clauses, with appropriate constraints and exclusions, were written in `DIMACS` format.

Test case

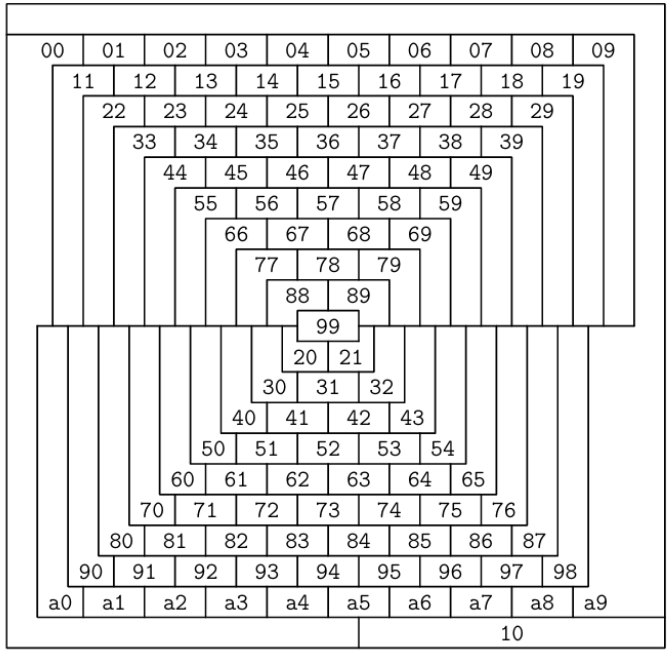


Fig. 33. The McGregor graph of order 10. Each region of this "map" is identified by a two-digit hexadecimal code. Can you color the regions with four colors, never using the same color for two adjacent regions?

Figure: McGregor's graph of order 10, corresponds to a planar graph, where each region is mapped to a vertex, and adjacent regions have an edge. For this graph, $|V| = 110$, $|E| = 324$. By four color theorem, a 4-coloring must exist.

Test case

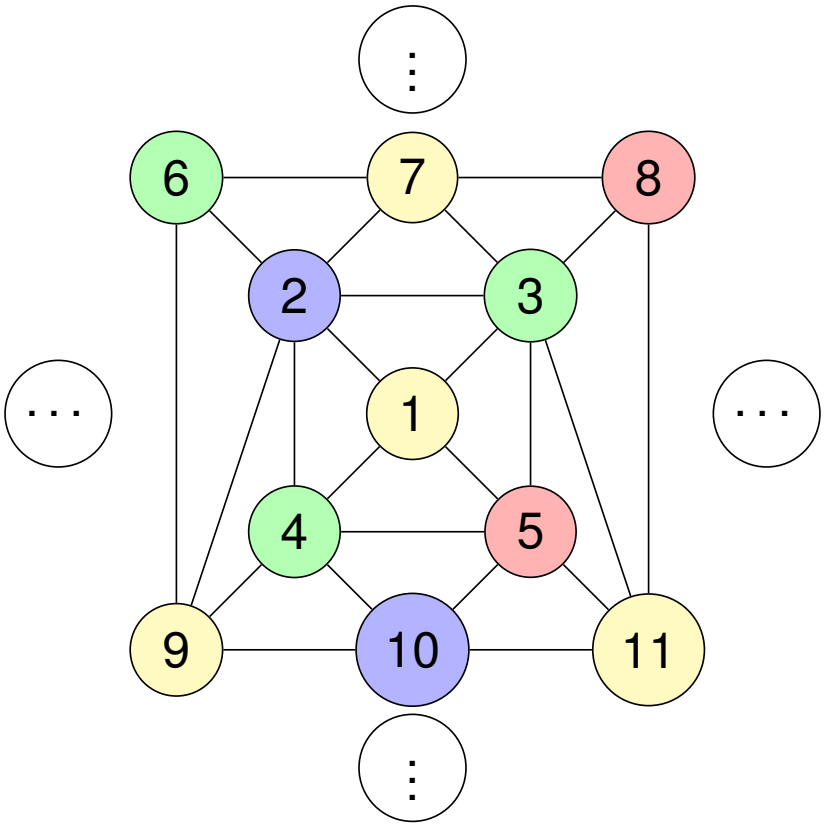


Figure: The variable numbering used in implementation. Only a partial graph is shown, with a possible 4 color solution found by the SAT solver.

Results

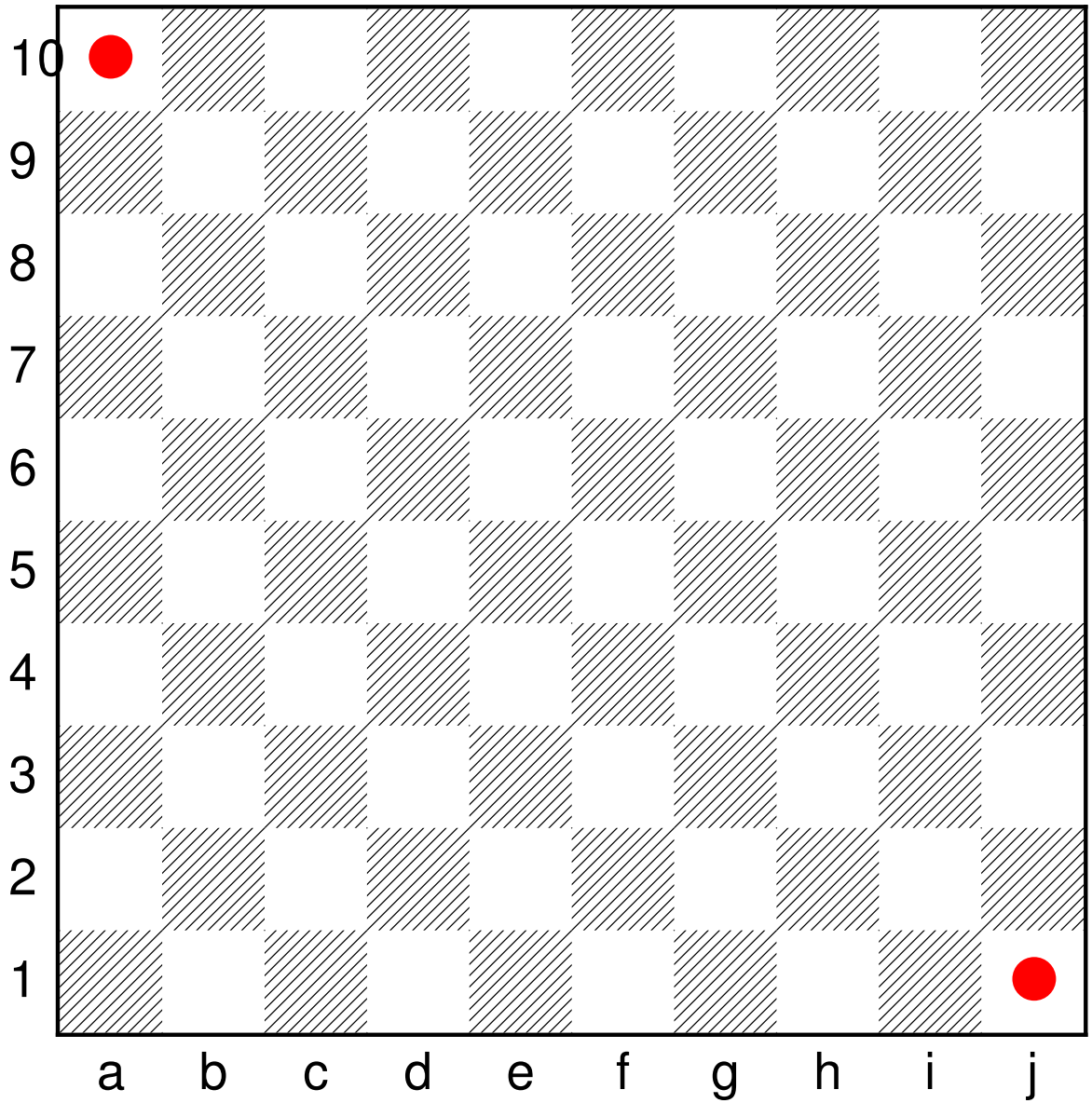
Encoding	Variables	Clauses	Time (s)
No extra constraint	440	1406	0.002
Sinz	1161	2944	0.291
Tree-type	839	2622	0.113

We took $d = 4, r = 7$ as per book, and the results match exactly.

Problem 2 - Domino Tiling Problem

Problem: Given a 10×10 chessboard with two diagonally opposite corners removed, the objective is to determine whether it is possible to cover the remaining 98 squares of the chessboard using exactly 49 2×1 dominoes. No two dominoes may overlap.

The board, with red mark squares removed



Implementation-1

Variables:

$$x_{i,j,d} = \begin{cases} \top, & \text{if a domino is starting at } (i,j) \text{ in direction } d \\ \bot, & \text{otherwise} \end{cases}$$

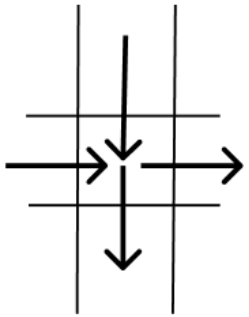
where:

- $0 \leq i \leq 9$, $0 \leq j \leq 9$ are the row and column indices of the top-left square of a domino.
- $d \in \{H, V\}$ is the direction of the domino.
- H means a horizontal domino covering squares (i,j) and $(i, j+1)$,
- V means a vertical domino covering squares (i,j) and $(i+1, j)$.

Total number of variables = 200

Constraints

1. Covering Condition: Each square must be covered.



$$(x_{i,j,H} \vee x_{i,j-1,H} \vee x_{i,j,V} \vee x_{i-1,j,V})$$

2. Mutual Exclusivity: For each square (i,j), ensure that no two dominoes overlap:

$$(\neg x_{i,j,H} \vee \neg x_{i,j-1,H}) \wedge (\neg x_{i,j,H} \vee \neg x_{i,j,V}) \wedge$$
$$(\neg x_{i,j,H} \vee \neg x_{i-1,j,V}) \wedge (\neg x_{i,j-1,H} \vee \neg x_{i,j,V}) \wedge$$
$$(\neg x_{i,j-1,H} \vee \neg x_{i-1,j,V}) \wedge (\neg x_{i,j,V} \vee \neg x_{i-1,j,V})$$

Constraints

3. Boundary Conditions: Dominoes cannot extend beyond the edge of the board, No horizontal domino can start in the last column.

$$\forall i, (\neg x_{i,9,H})$$

No vertical domino can start in the last row.

$$\forall j, (\neg x_{9,j,V})$$

4. Mutilation Condition: Dominoes cannot cover the removed corners.

$$(\neg x_{0,0,H}) \wedge (\neg x_{0,0,V}) \wedge (\neg x_{9,9,H}) \wedge (\neg x_{9,9,V})$$

Total number of clauses = 654

Implementation-2

- Optimizations were made by removing the 4 variables corresponding to the two removed corners, as they are always false by default.

$$x_{0,0,H}, x_{0,0,V}, x_{9,9,H}, x_{9,9,V}$$

- Additionally, variables for domino placements in the last row and last column were eliminated since they cannot start there.

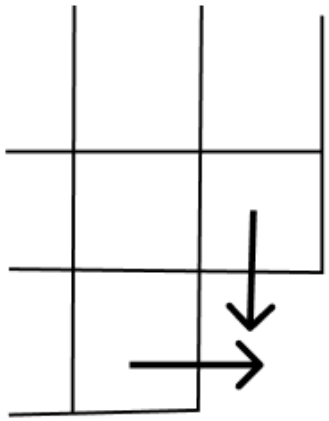
$$\forall i, x_{i,9,H}$$

$$\forall j, x_{9,j,V}$$

- Corresponding clauses were adjusted to reflect these removals, reducing number of variables to 178 and clauses to 572.

Implementation-3

- We can remove the following variables, as dominoes cannot extend to the last cell of the board.



$x_{9,8,H}$, $x_{8,9,V}$

- Corresponding clauses were adjusted similarly. This reduces number of variables to 176 and number of clauses remain same.

Results

Encoding	Variables	Clauses	Time (s)
Impl-1	200	654	0.001
Impl-2	178	572	0.001
Impl-3	176	572	0.001

Optimal one which is Impl-3 matches the values mentioned in the book (TAOCP, Volume 4, Fascicle 6, Section 7.2.2.2 page: 114 Problem B1).

Optimizations in clasp

Optimization Mode (–opt-mode): It Controls how clasp finds and optimizes models based on objective function. It has following modes-

1. **opt**
2. **enum**
3. **optN**
4. **ignore-** Ignores optimize statements, treating the problem as non-optimizing.

Configuration Modes in clasp

Configuration Modes (–configuration): Allows selection of different parameter settings optimized for various problem types. It has following modes-

1. **auto-** Automatically selects configuration based on the problem type.
2. **frumpy-** Stability and reliability are prioritised over speed.
3. **jumpy-** Aggressive settings for faster problem-solving.
4. **tweety**
5. **handy**
6. **crafty**
7. **trendy**
8. **many-** Uses a portfolio of configurations to solve the problem.

Enumeration Modes and threads in clasp

Enumeration Modes (–enum-mode): Configures how clasp enumerates (or lists) multiple solutions. It has following modes-

1. **bt-** Backtracks on decision literals from found solutions.
2. **record-** Adds constraints (nogoods) to prevent re-encountering solutions.
3. **brave**
4. **cautious**
5. **auto**

Threads/Parallel Mode (–parallel-mode): Enables parallel search for faster problem-solving by utilizing multiple threads.

Comparision between Configuration modes and enum modes

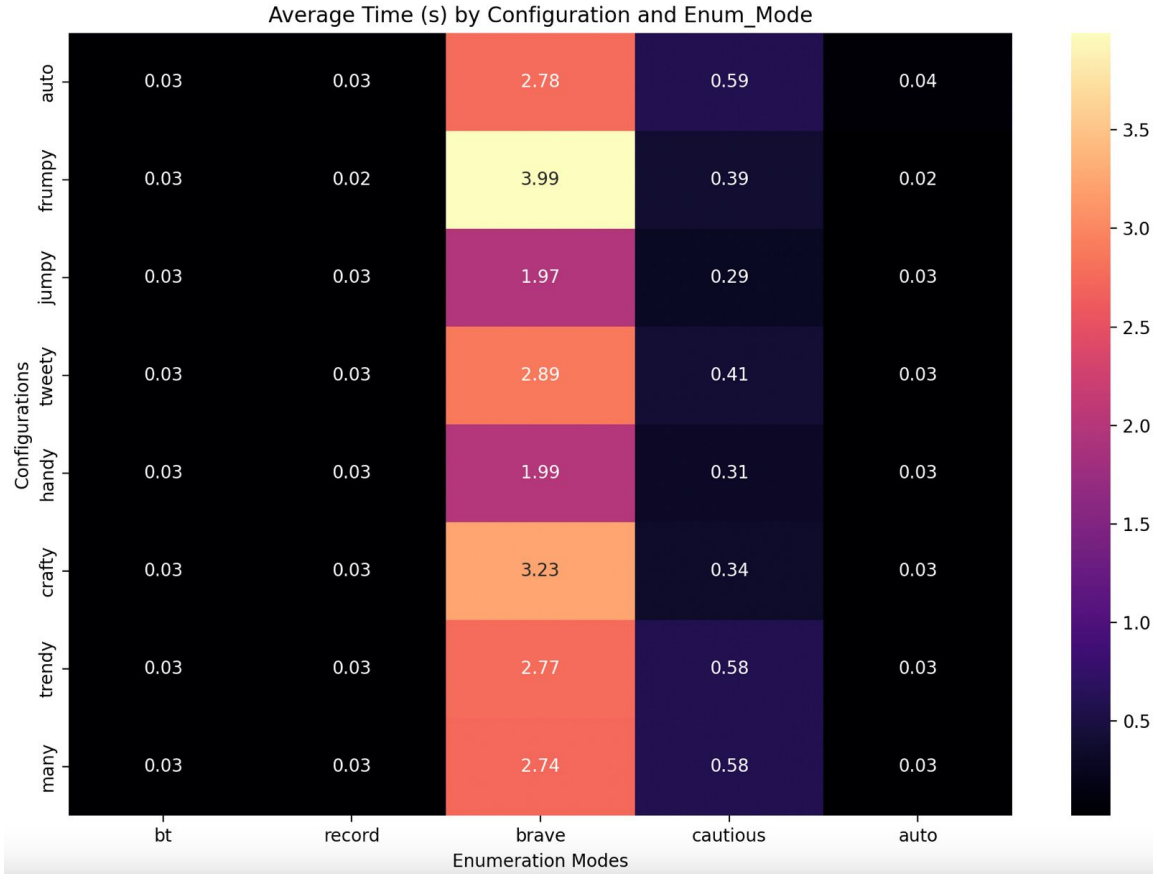


Figure: Brave enumeration modes is not helpful.

Comparision between Configuration modes and Optimization modes

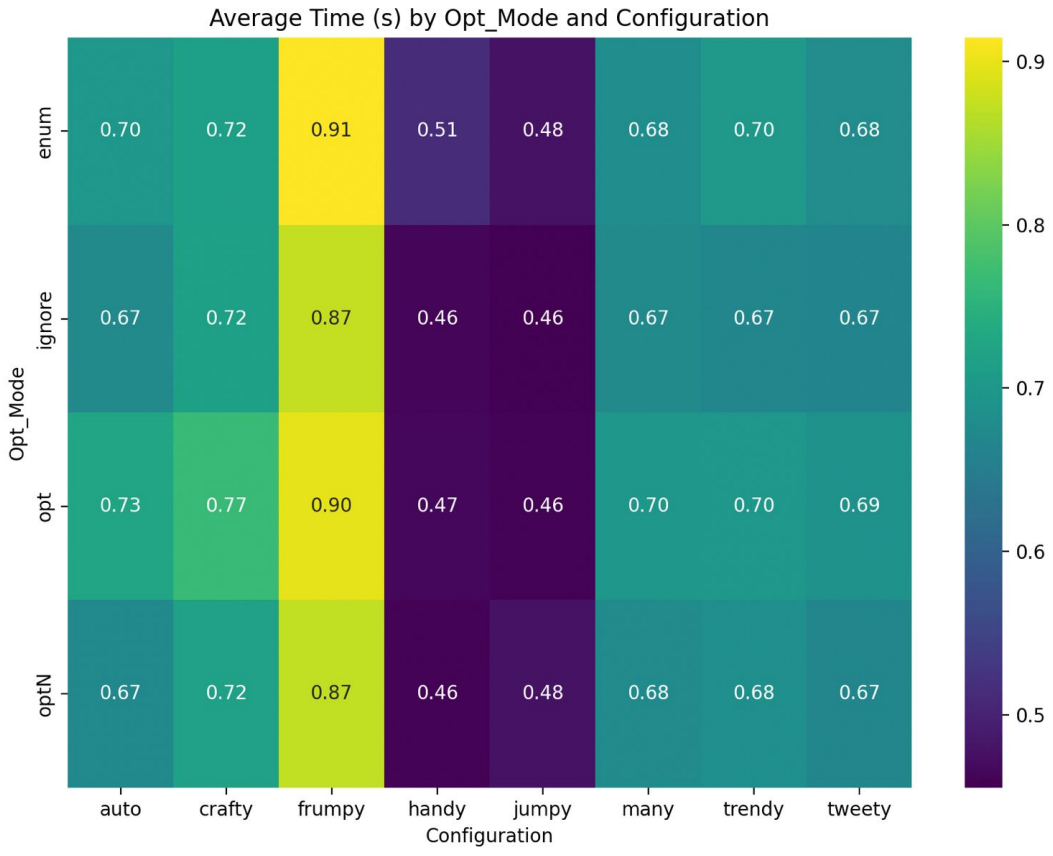


Figure: Frumpy configuration mode is not much helpful with other optimization parameters.

Comparision between Optimization modes and enum modes

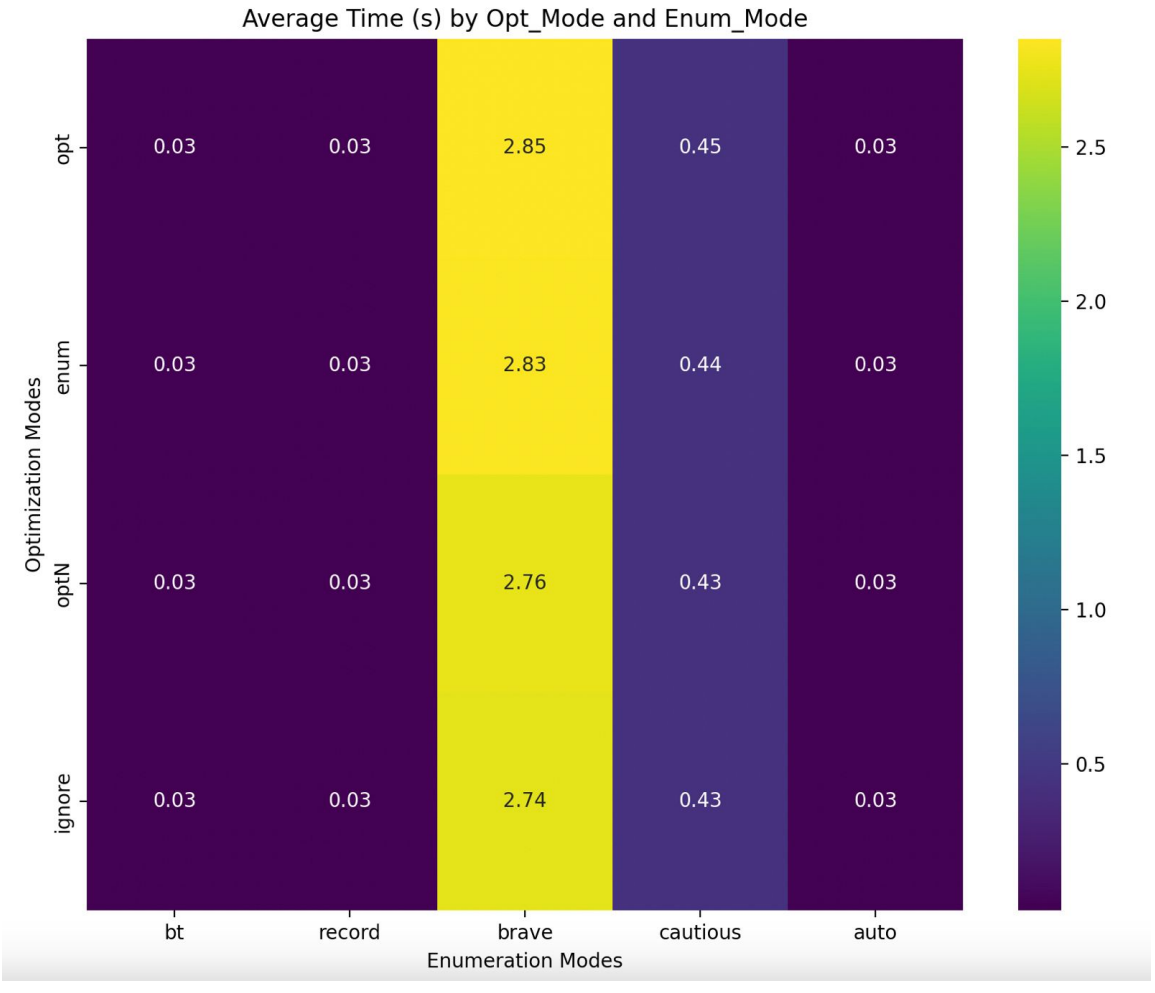


Figure: Brave enumeration modes is not helpful.

With Optimization modes

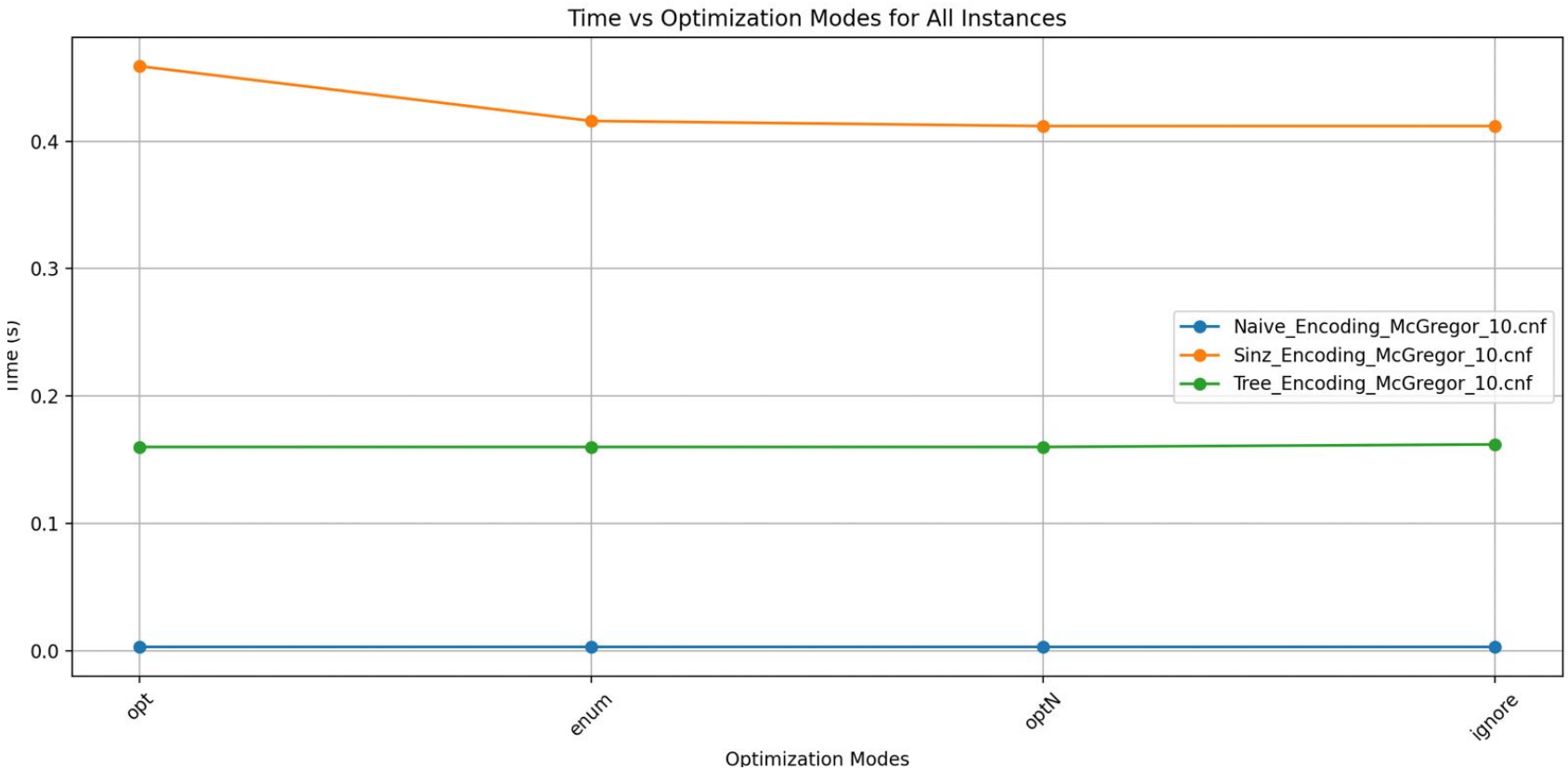


Figure: Execution time vs optimization modes

With Configuration modes

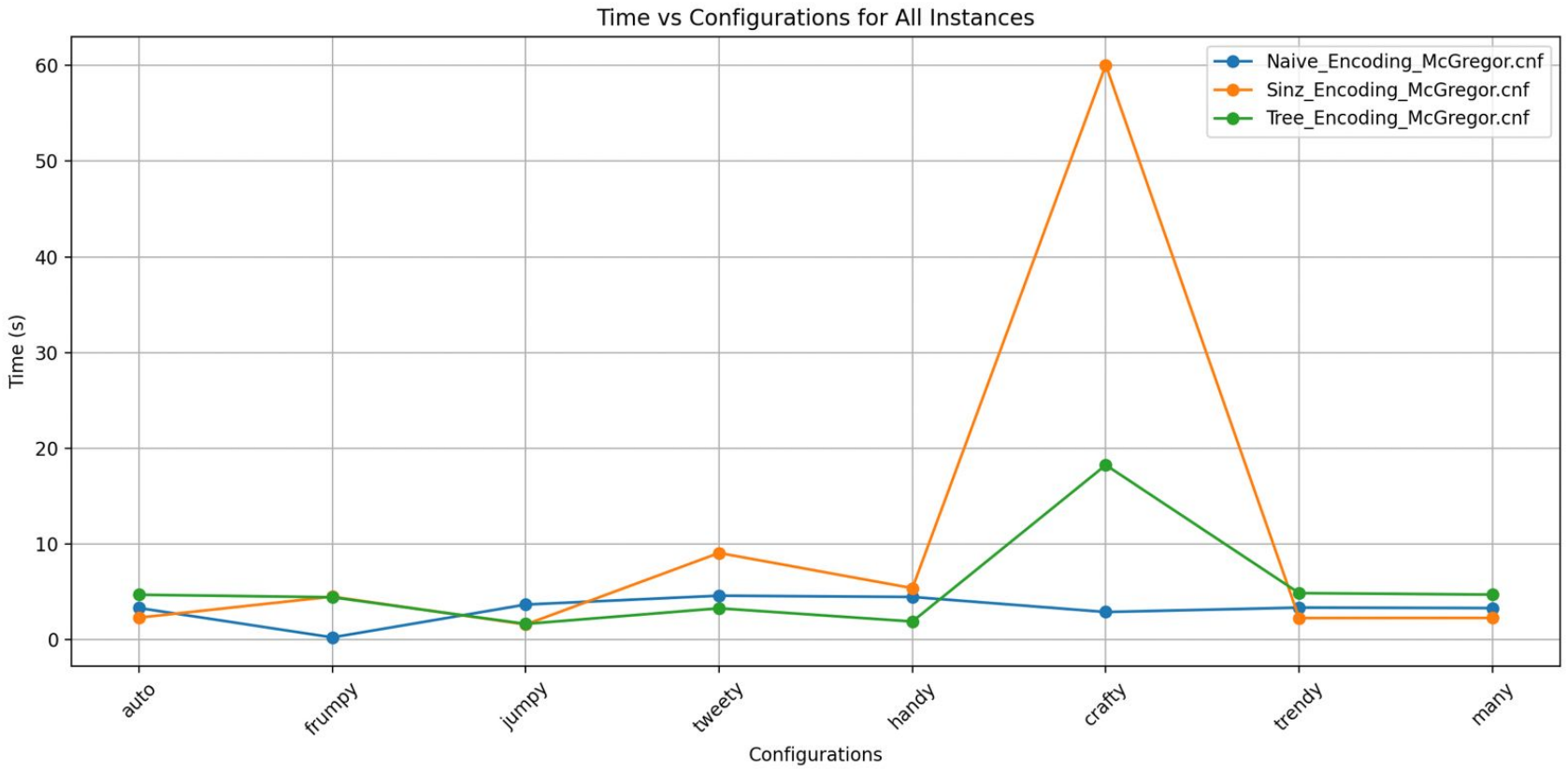


Figure: Execution time vs Configuration modes

With Enumeration modes

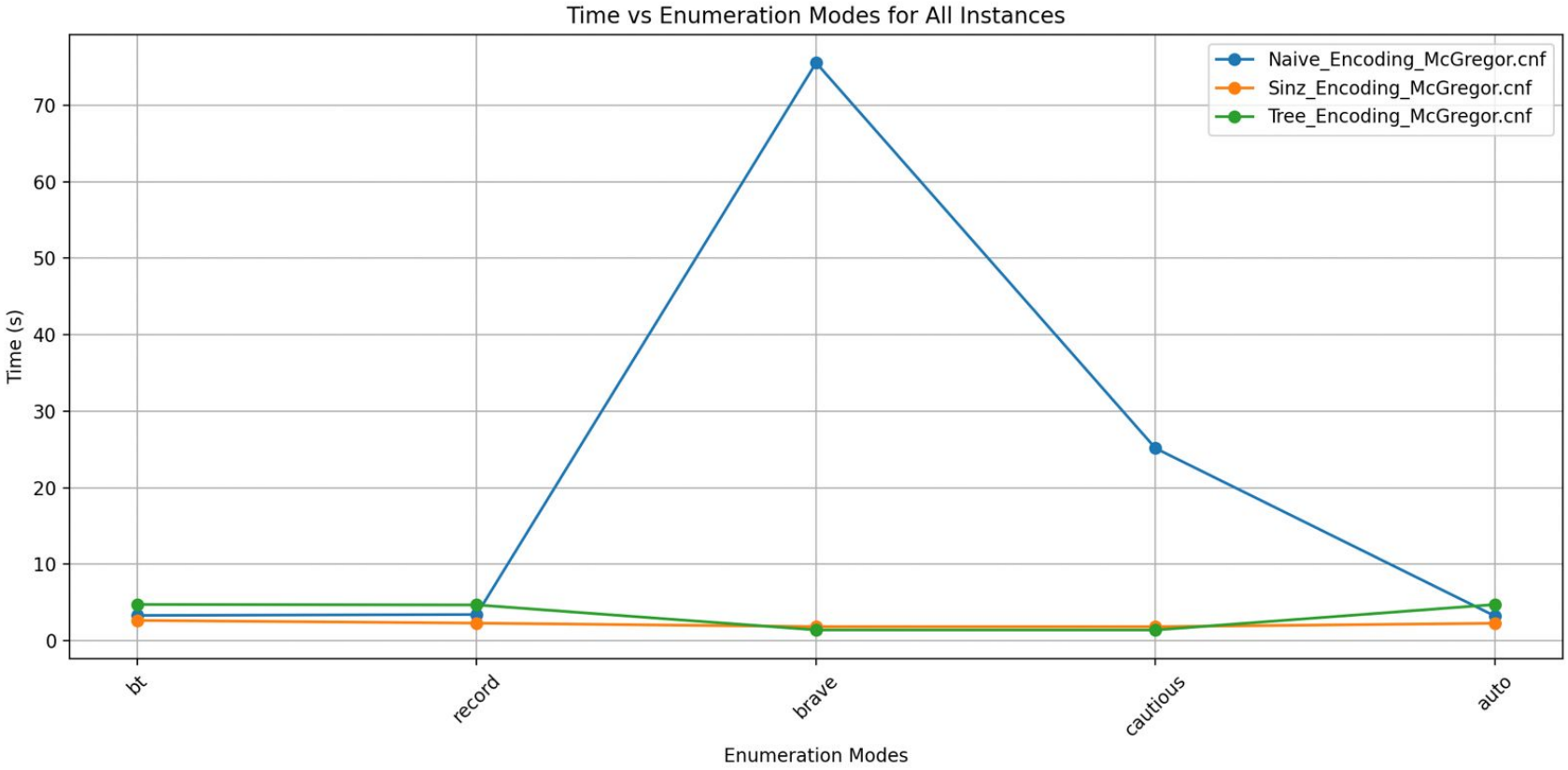


Figure: Execution time vs Enumeration modes

With Threads

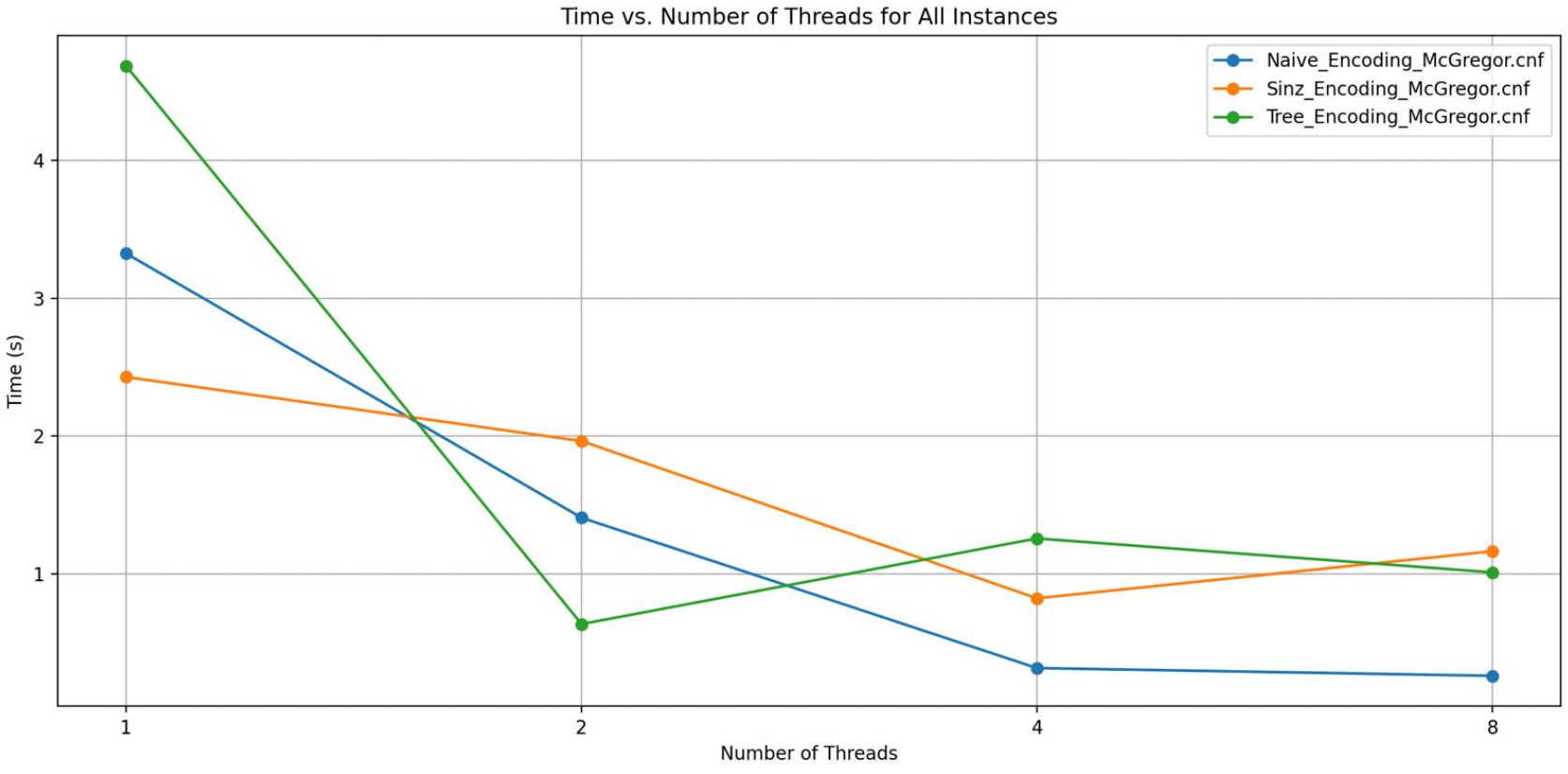


Figure: Execution times vs Number of threads

Optimizations - McGregor graph SAT with different orders

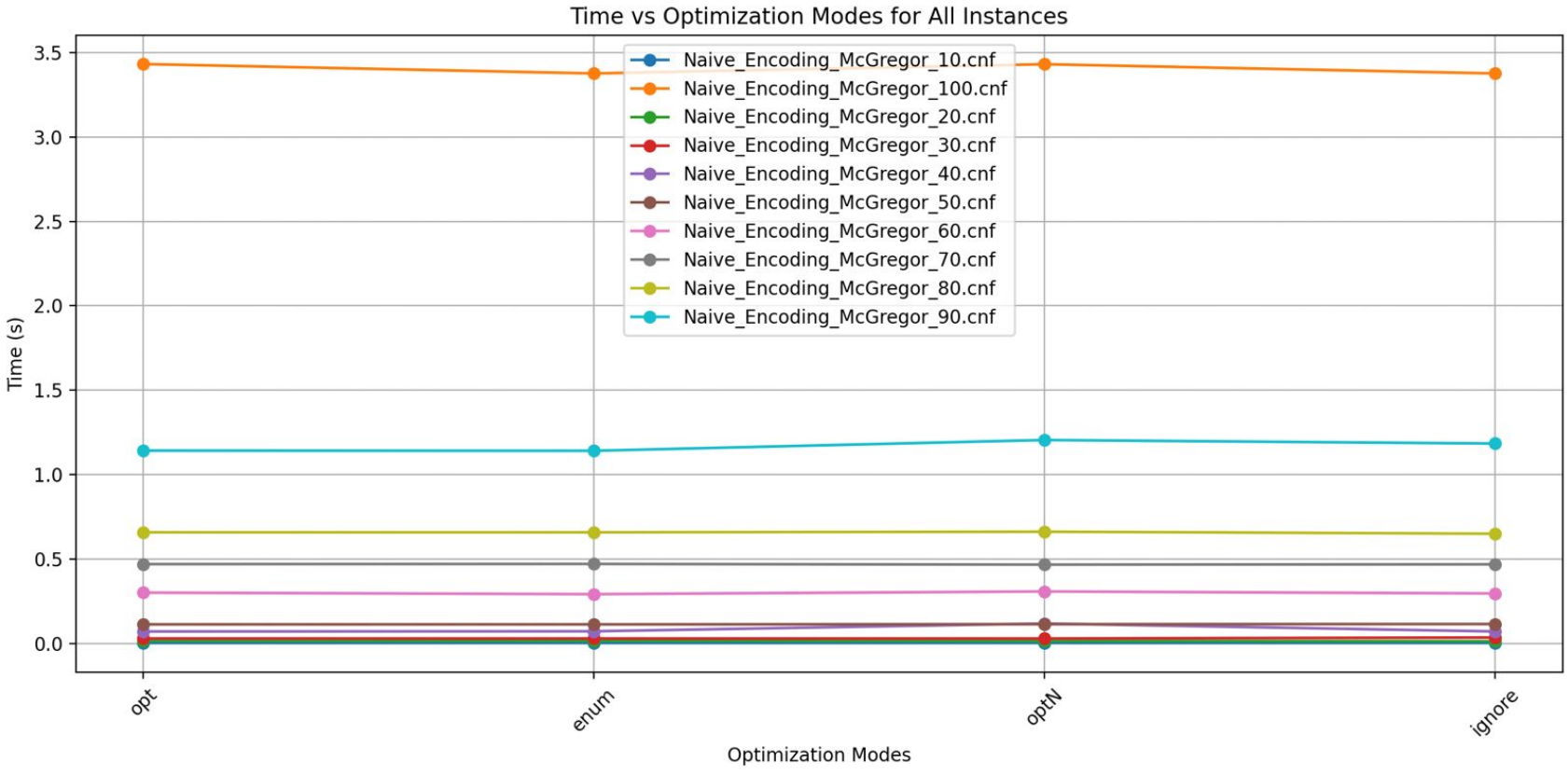


Figure: Here for small orders, optimisation didn't help much, rather it increased the execution time. For large orders enum optimisation mode was giving the optimal value.

Optimizations - McGregor graph SAT with different orders

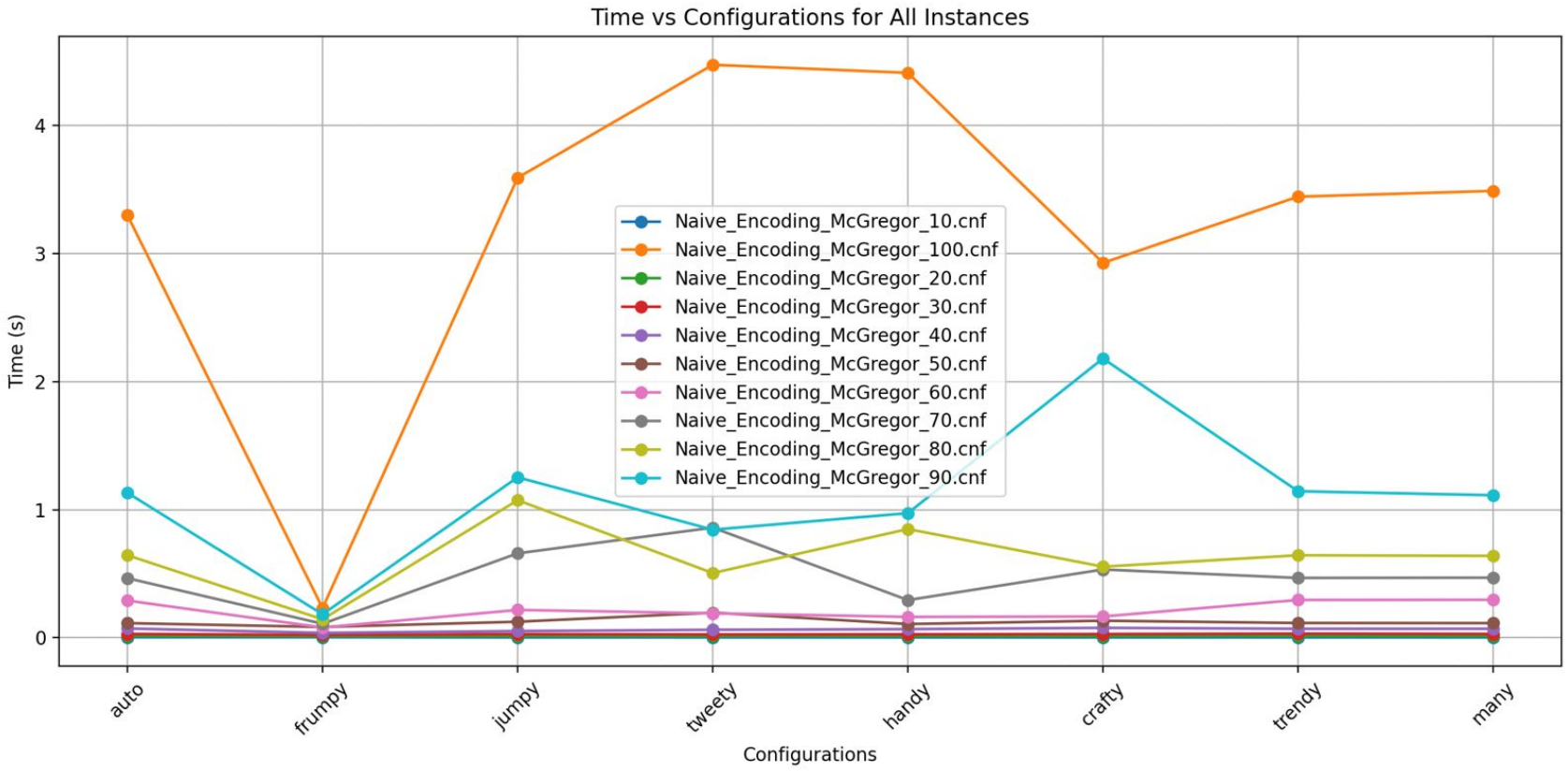


Figure: Here frumpy mode is giving best optimal value irrespective of the order.

Optimizations - McGregor graph SAT with different orders

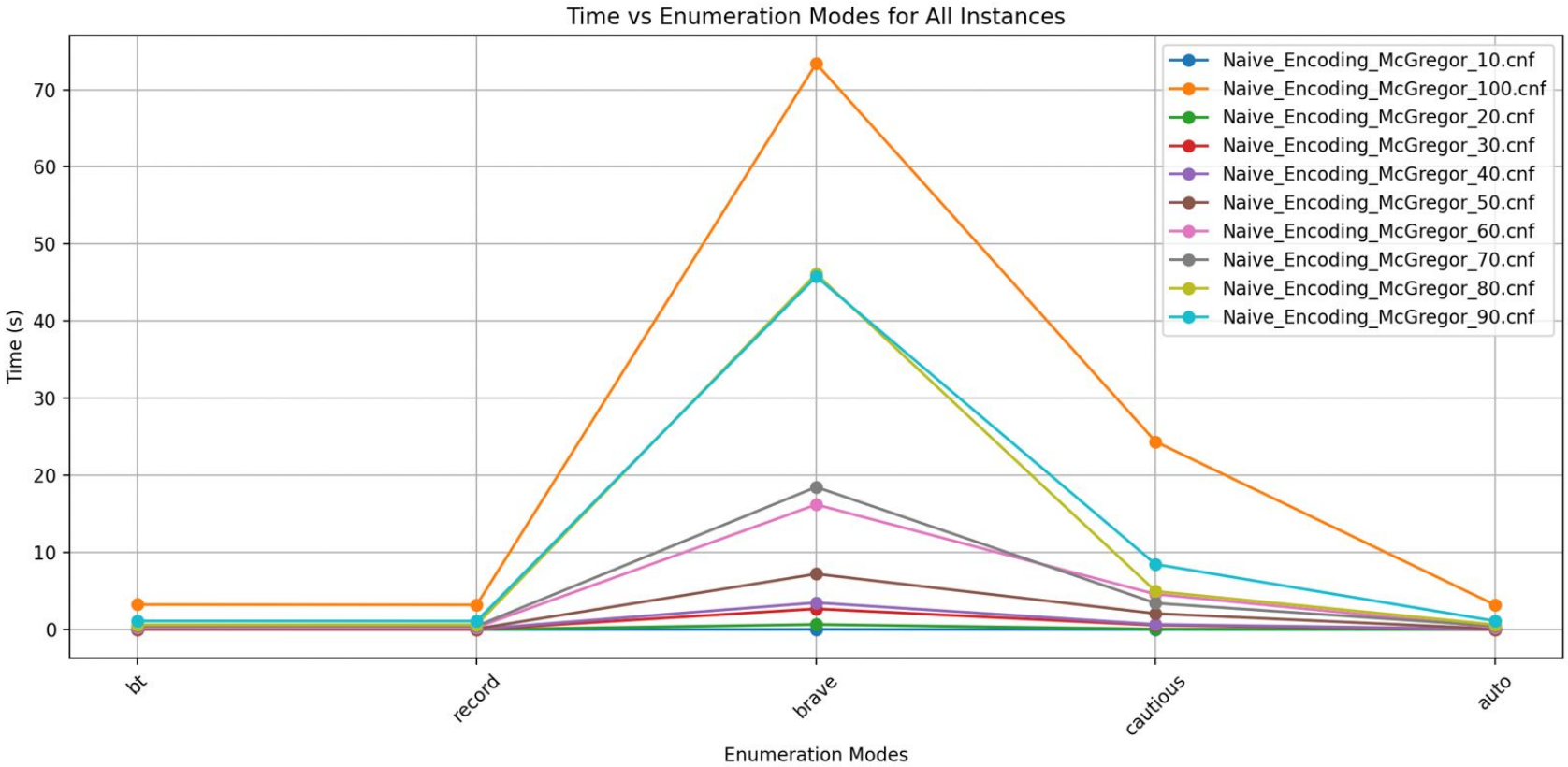


Figure: Brave enumeration modes has more variability with respect to order of the graph

Optimizations - McGregor graph SAT with different orders

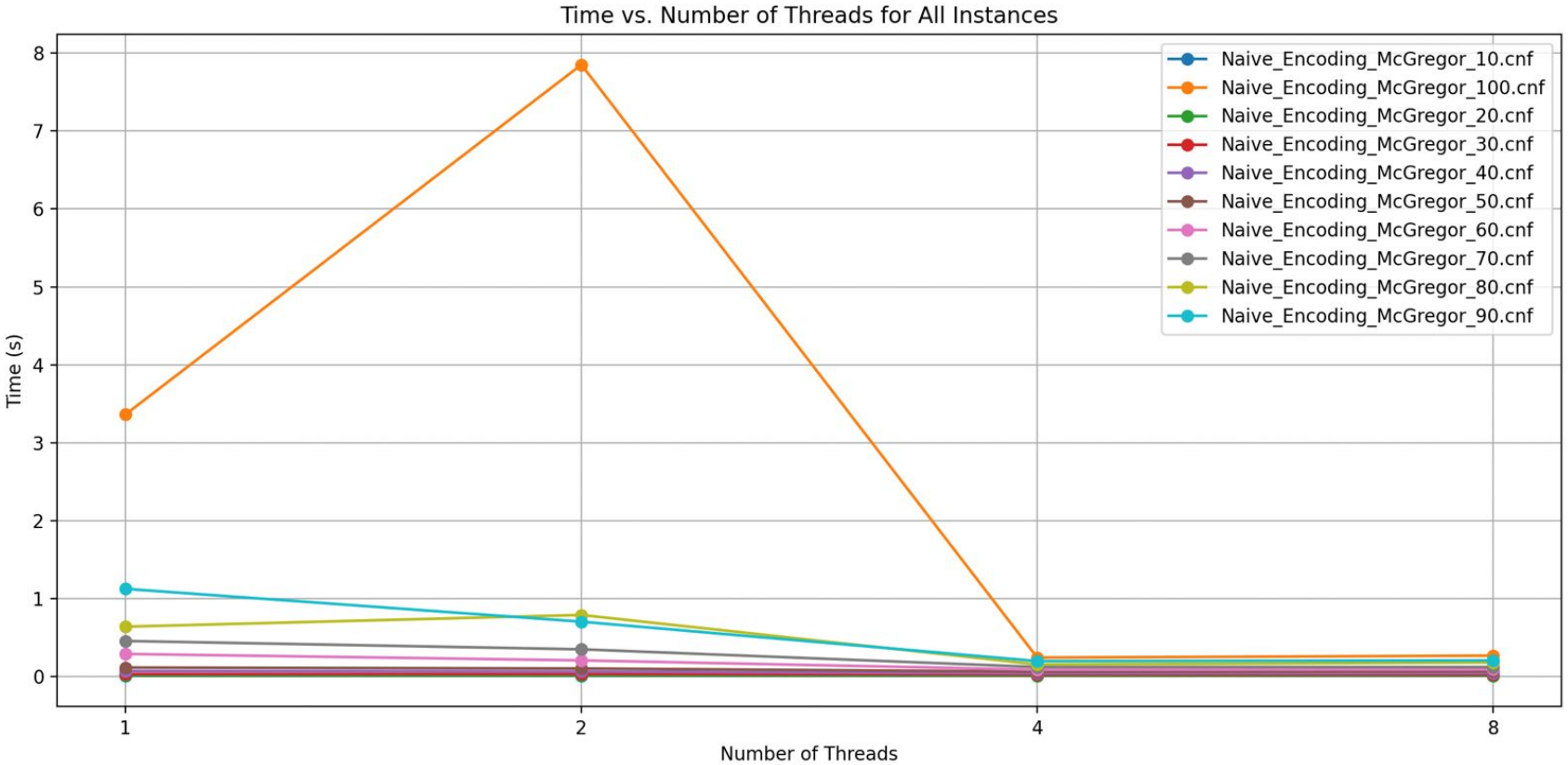
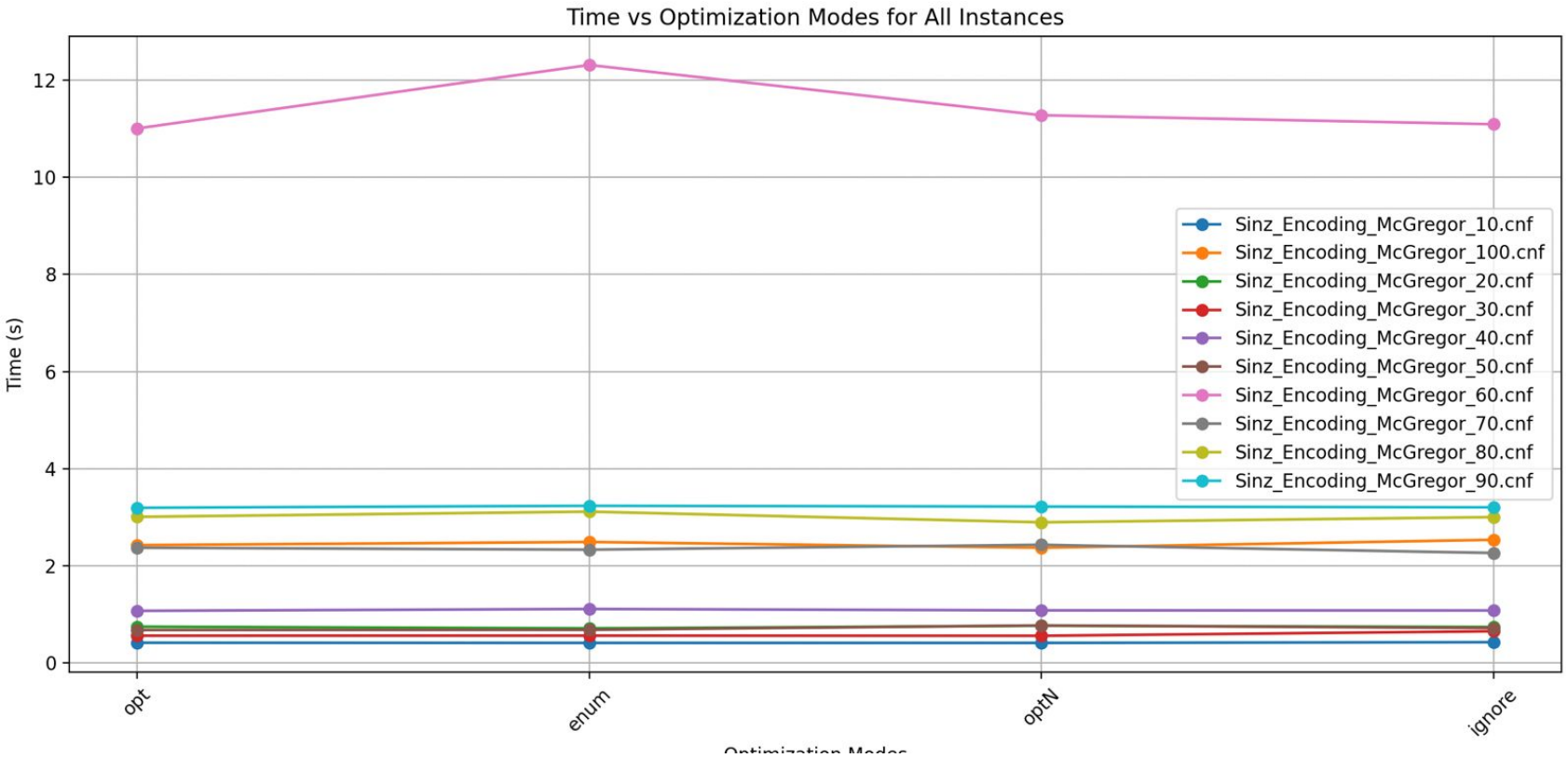
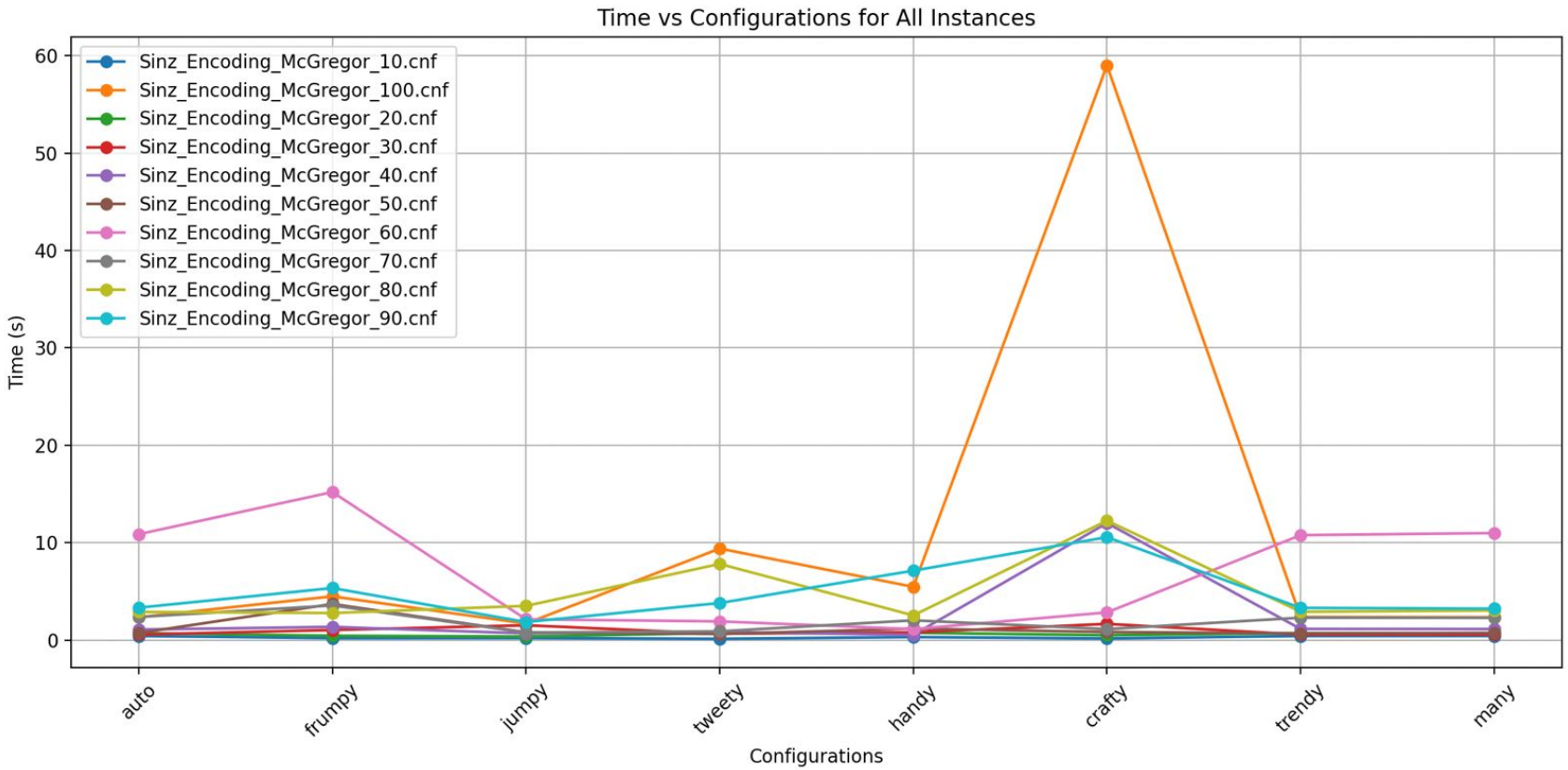


Figure: More number of threads is only giving much variation for large order graphs

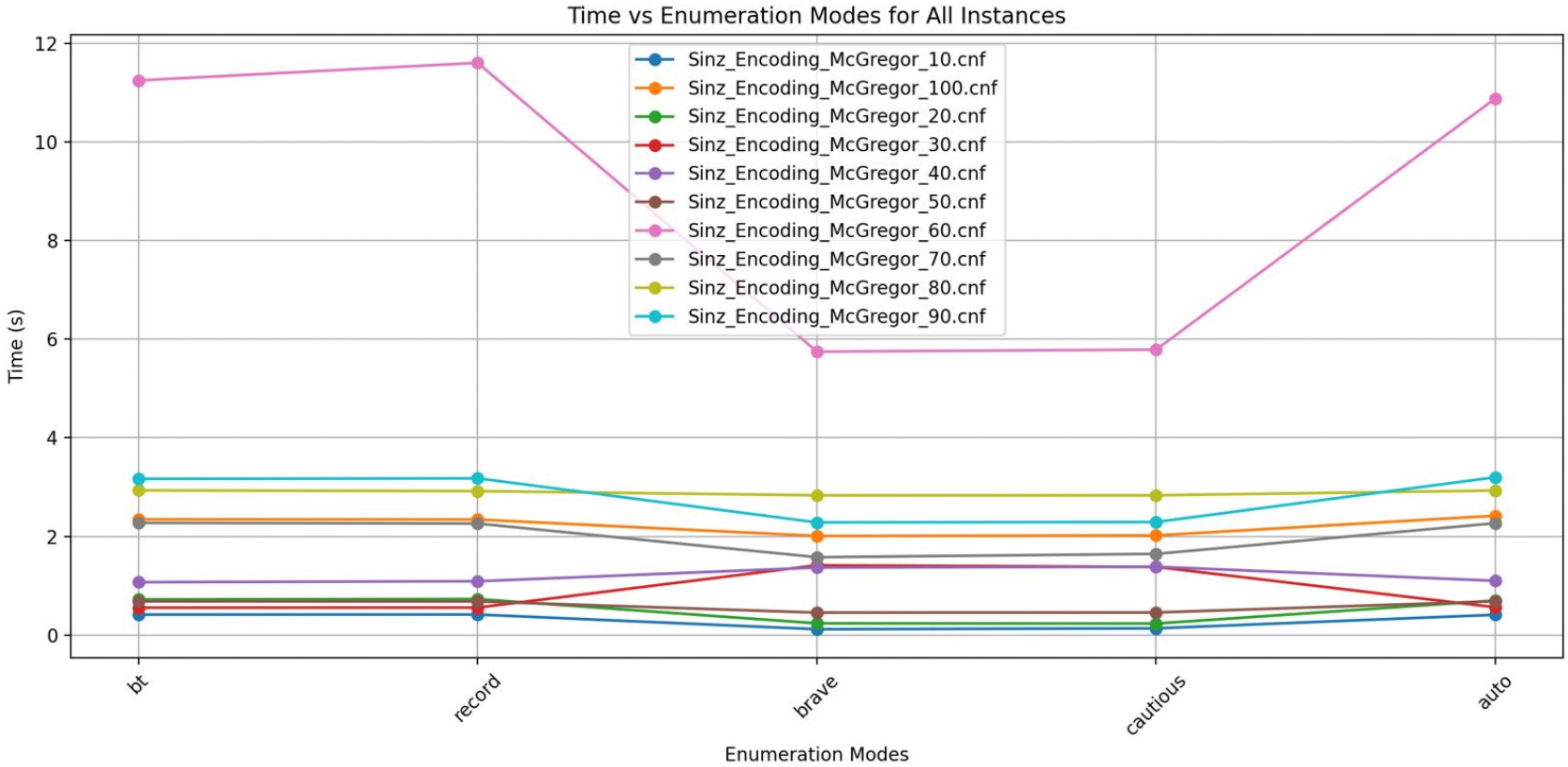
Optimizations - McGregor graph SAT with sinz encoding



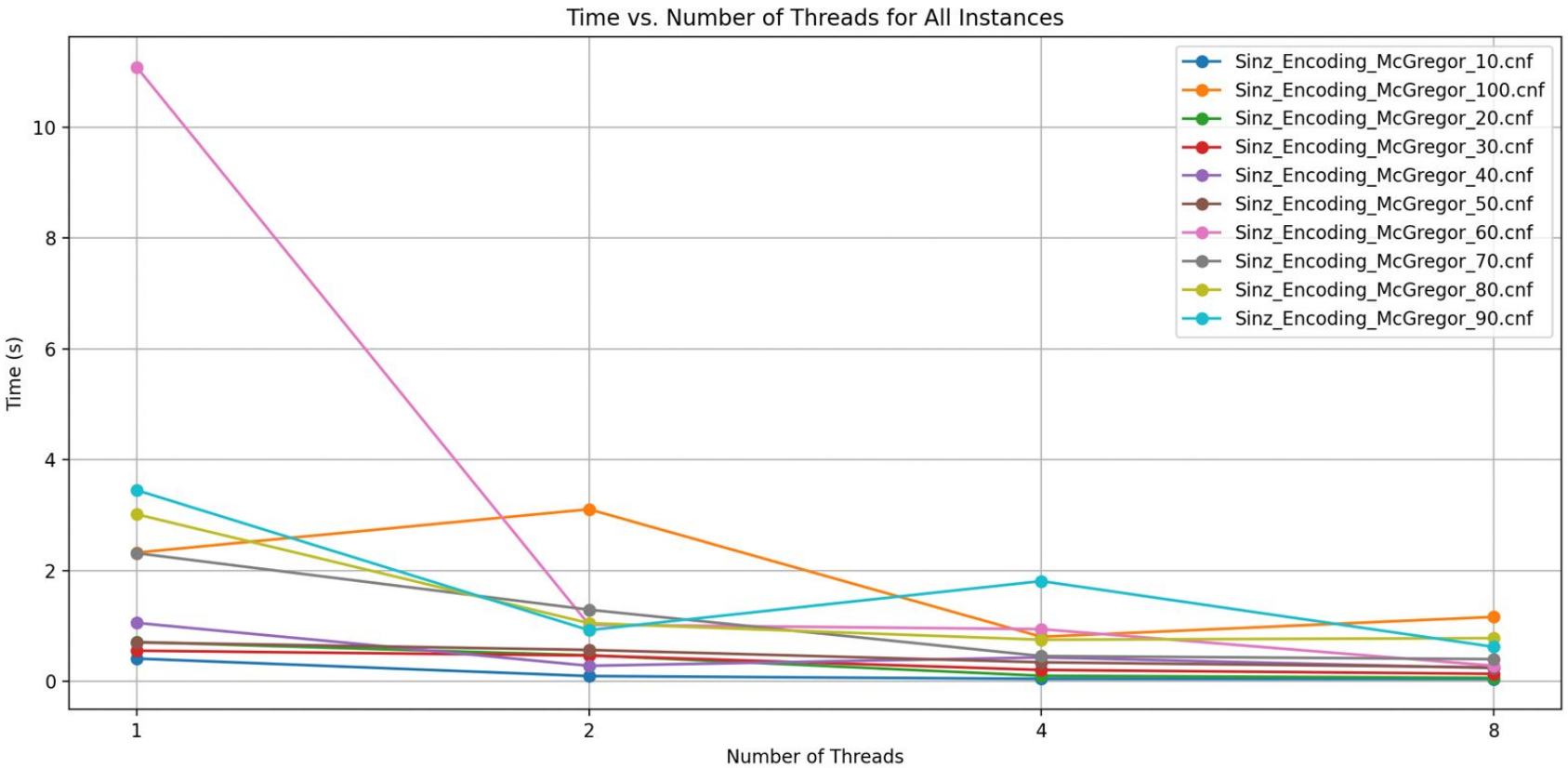
Optimizations - McGregor graph SAT with sinz encoding



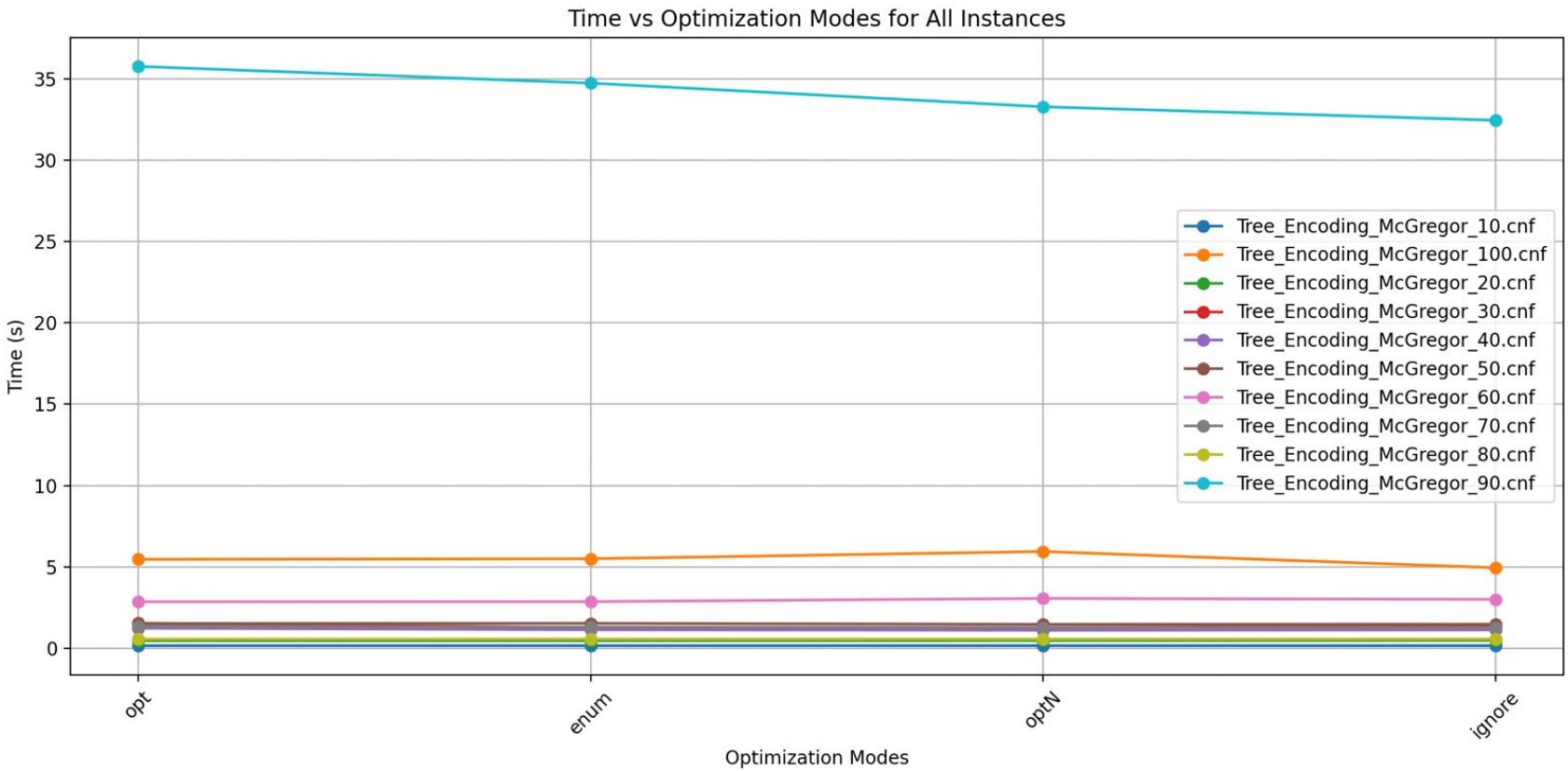
Optimizations - McGregor graph SAT with sinz encoding



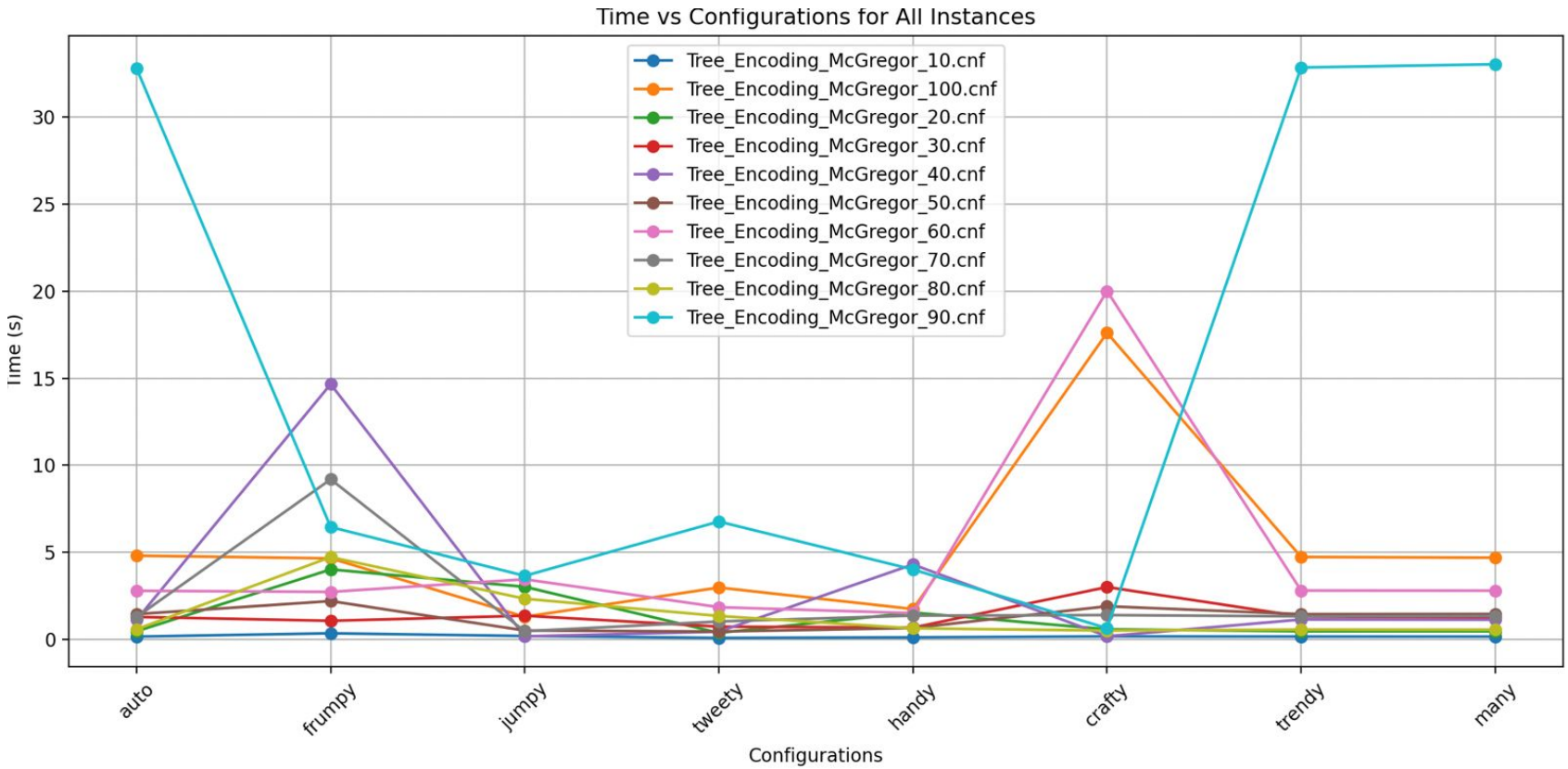
Optimizations - McGregor graph SAT with sinz encoding



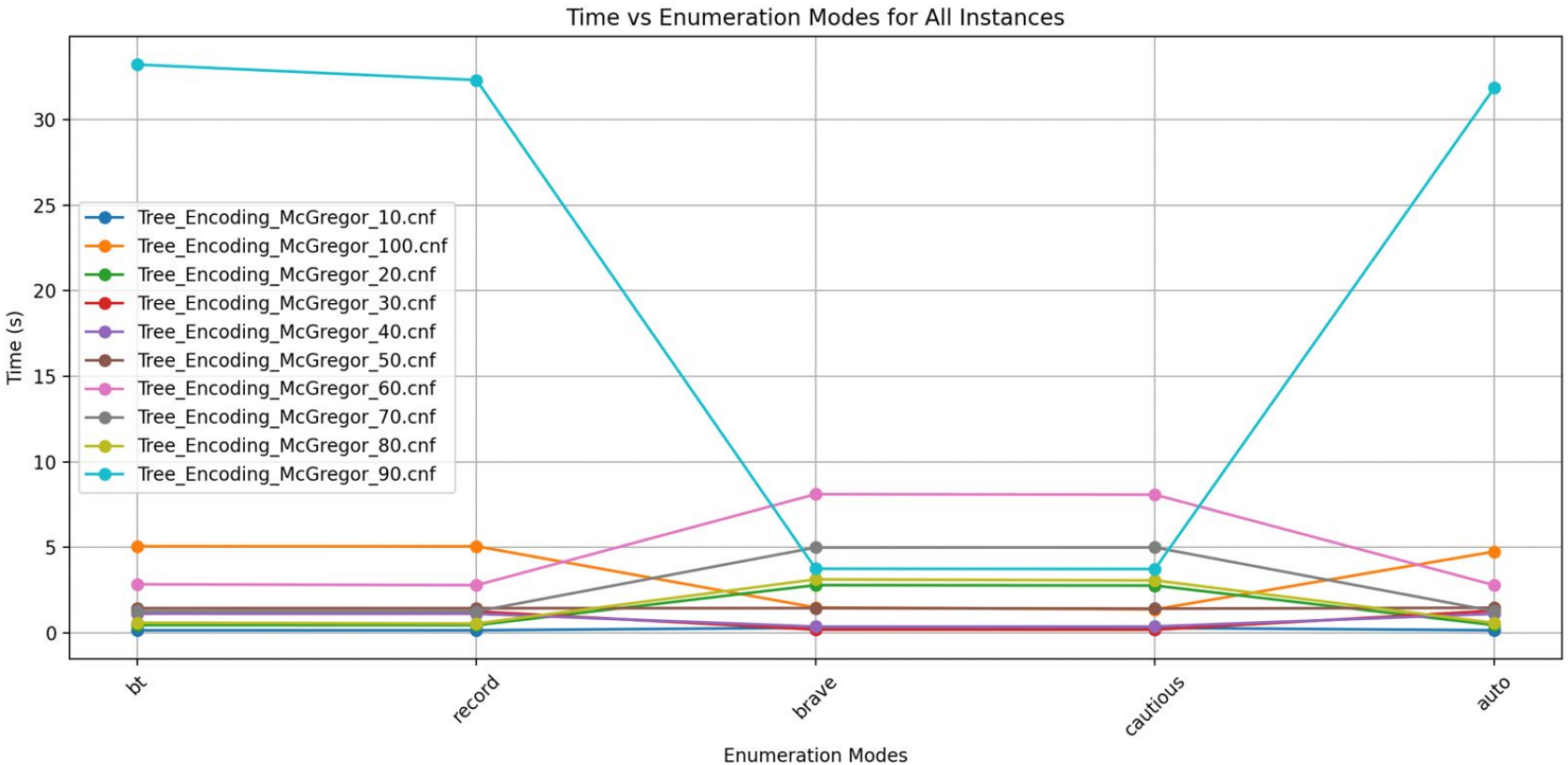
Optimizations - McGregor graph SAT with tree encoding



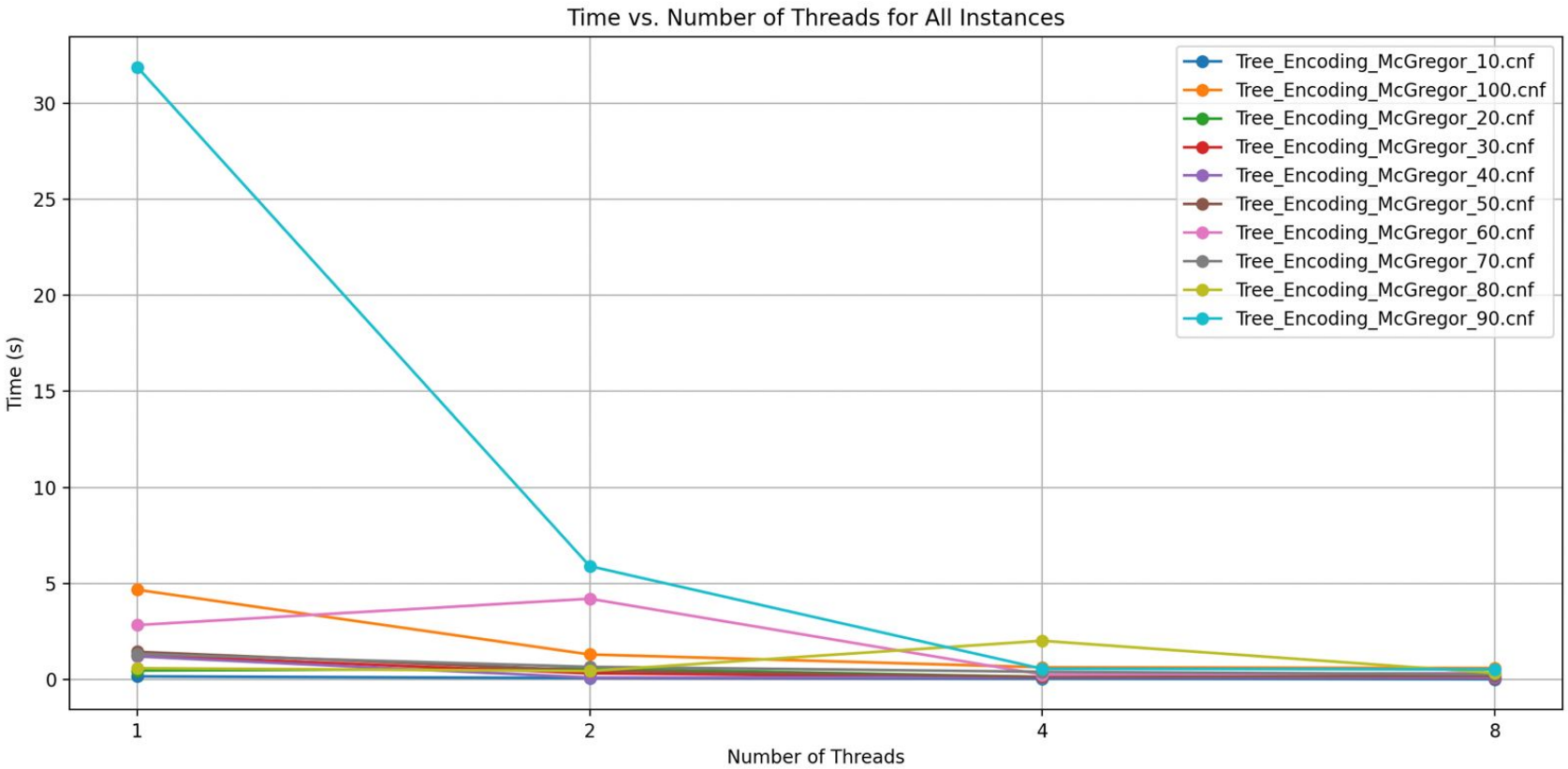
Optimizations - McGregor graph SAT with tree encoding



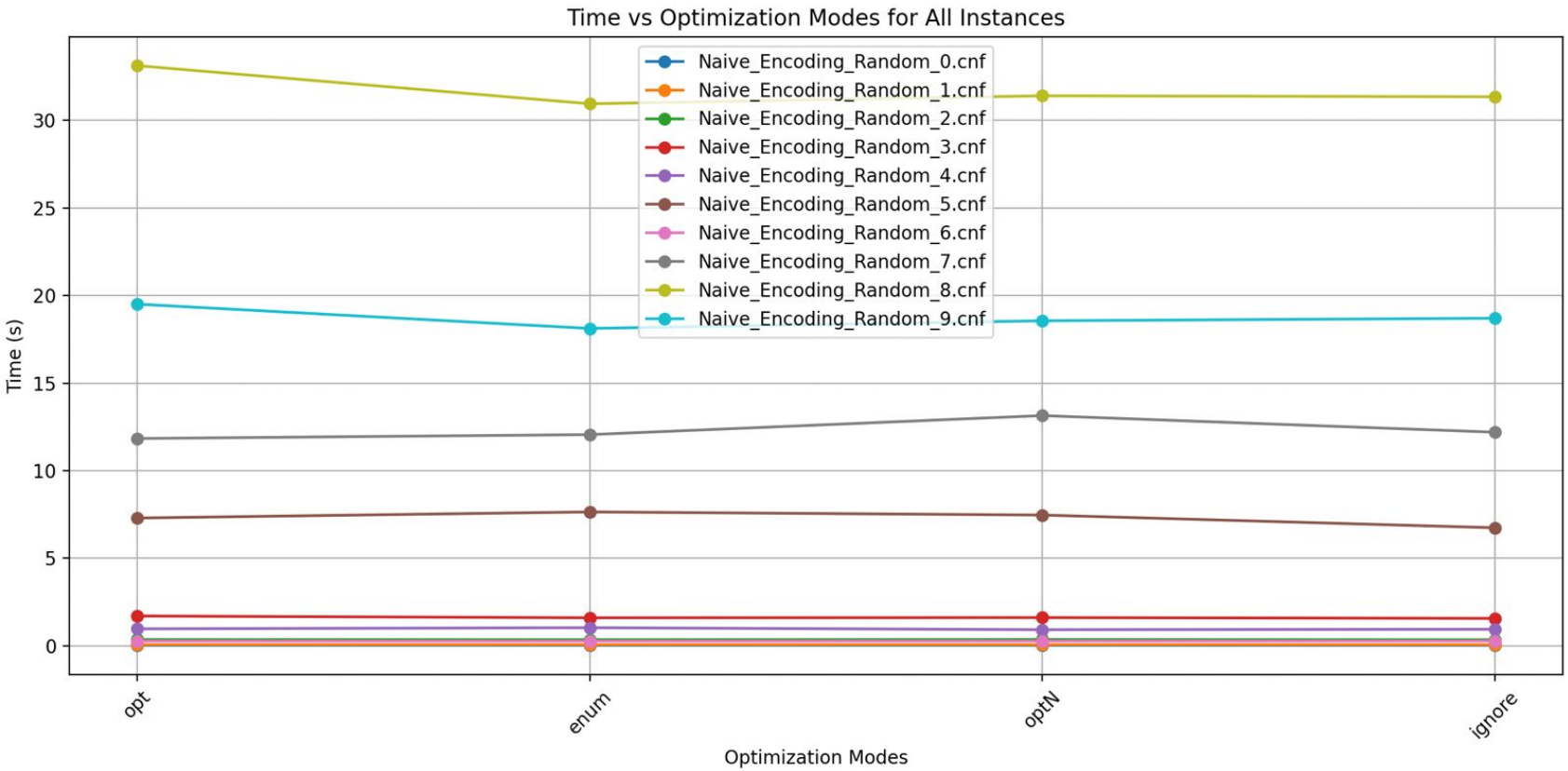
Optimizations - McGregor graph SAT with tree encoding



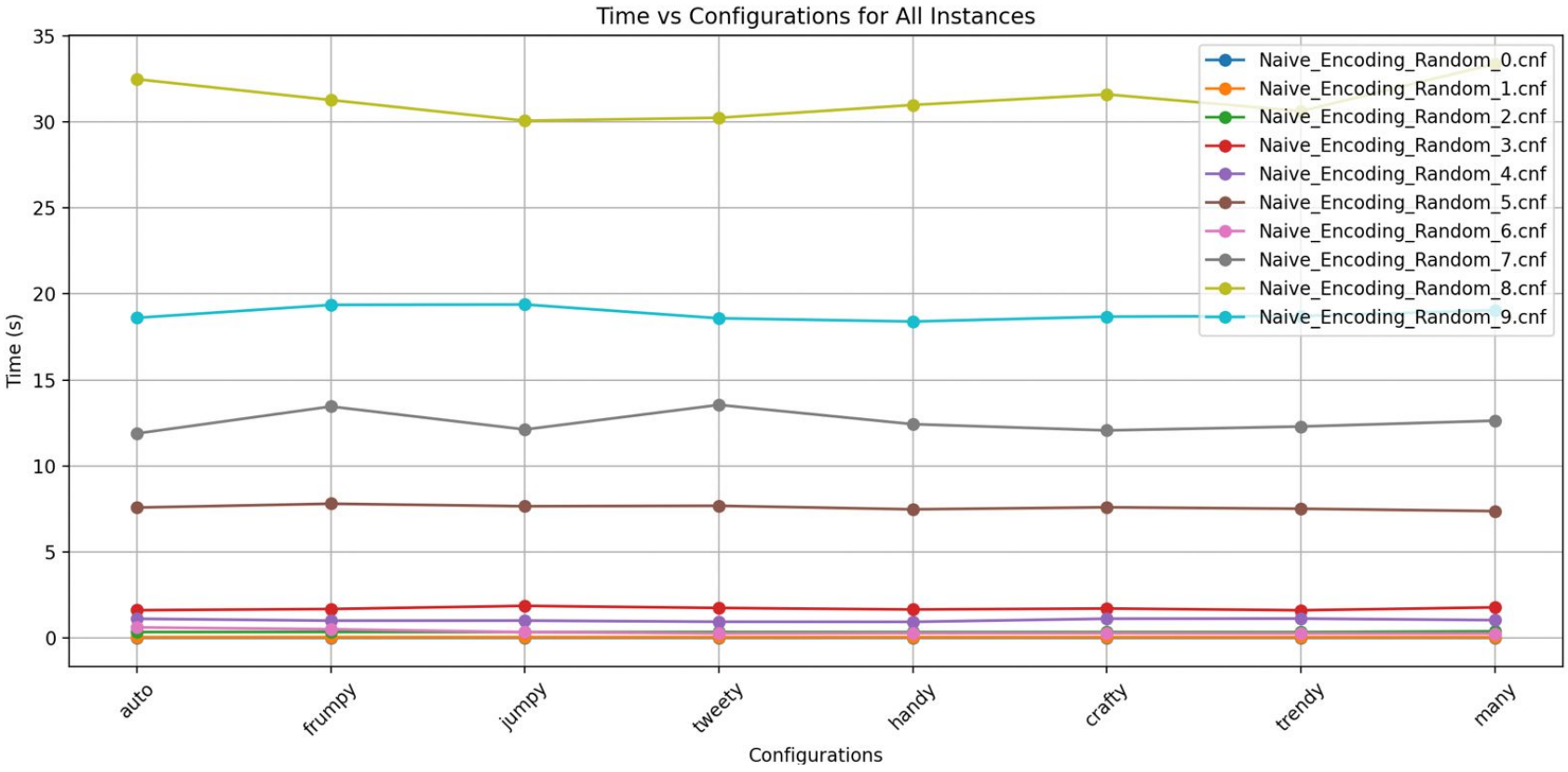
Optimizations - McGregor graph SAT with tree encoding



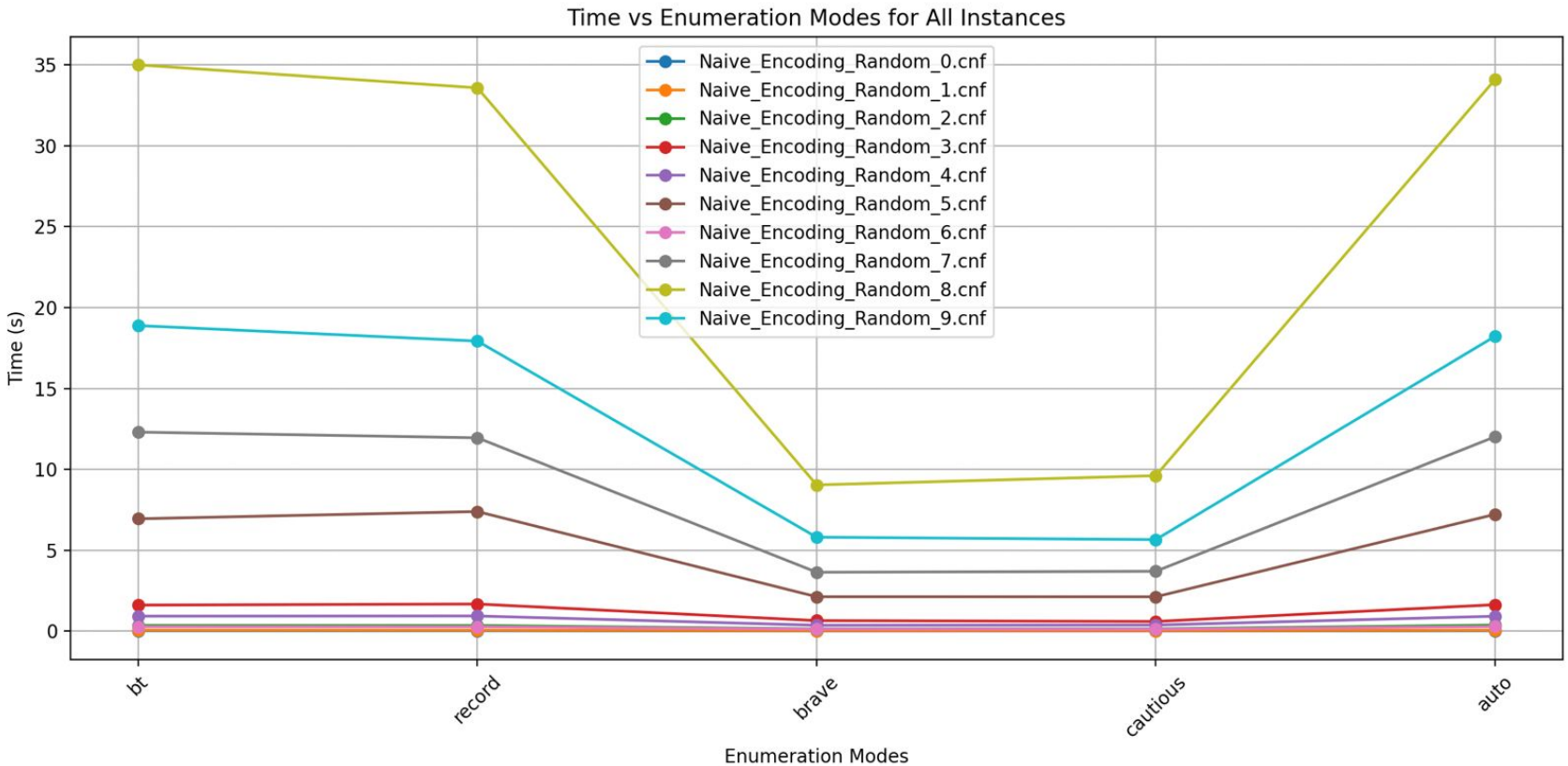
Optimizations - Random graph coloring with vertices upto 1000



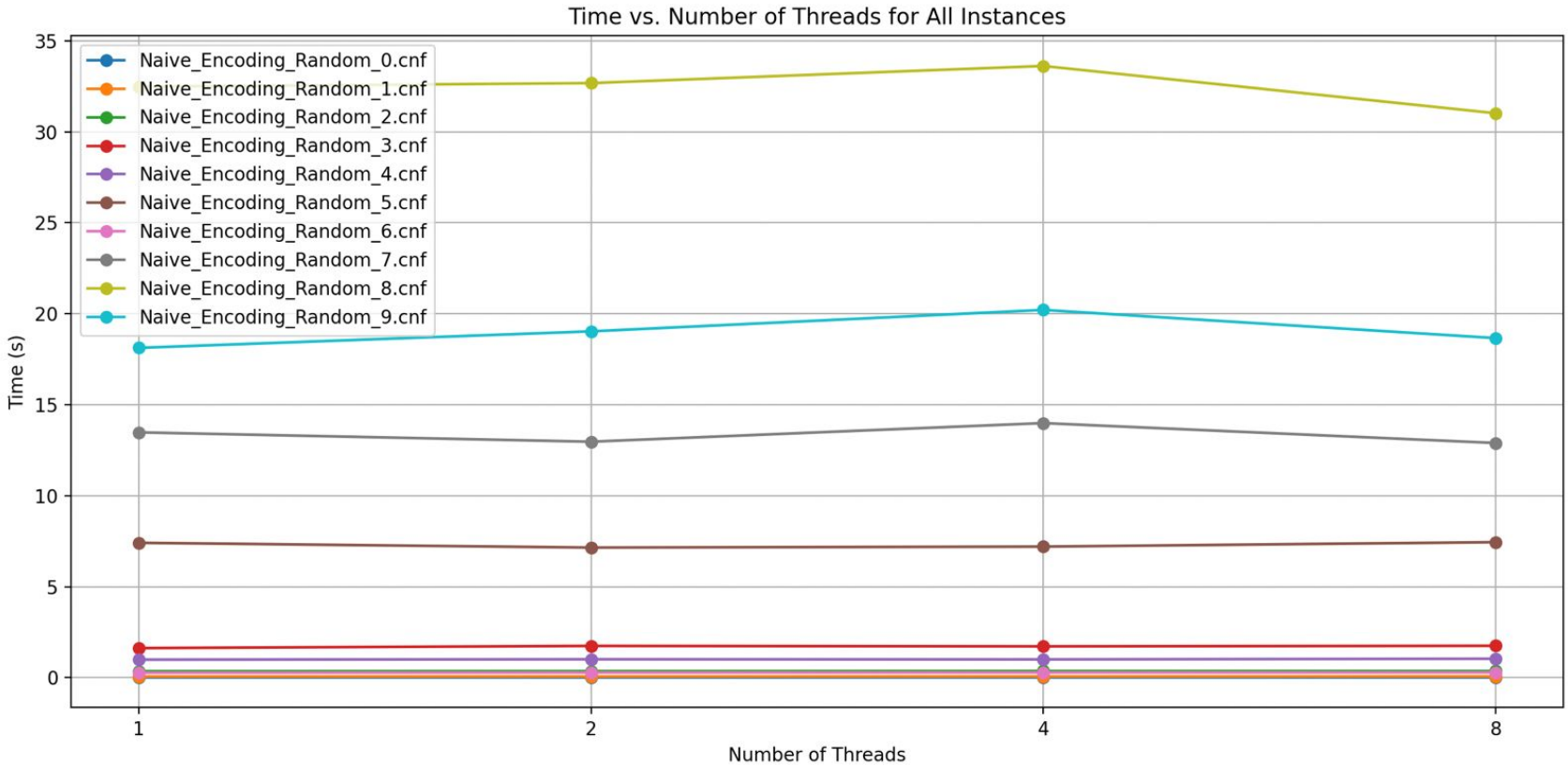
Optimizations - Random graph coloring with vertices upto 1000



Optimizations - Random graph coloring with vertices upto 1000



Optimizations - Random graph coloring with vertices upto 1000



Thank You!