Welcome and Thank You All
HyParSAT: A Hybrid Parallel Complete SAT Solver Using Parallel Java 2

By Jiten Patel
Project Committee

- Chair: Prof. Alan Kaminsky
  http://www.cs.rit.edu/~ark/
- Reader: Prof. Edith Hemaspaandra
  http://www.cs.rit.edu/~eh/
- Observer: Prof. Zack Butler
  http://www.cs.rit.edu/~zjb/
Agenda

1. SAT problem
2. SAT solver
3. Conflict Driven Clause Learning (CDCL) algorithm
4. Parallel complete SAT solvers
5. HyParSAT
6. Experiment results
7. Conclusion
SAT Problem ➔ Preliminaries

- Literal: $x_1, \neg x_2$
- Clause: $(x_1 \lor \neg x_2 \lor \neg x_3)$
- Conjunctive Normal Form (CNF) formula:
  $$E = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4)$$
- Truth assignment
  - Non-satisfying: $(x_1=\text{FALSE}, x_2=\text{TRUE}, x_3=\text{TRUE}, x_4=\text{TRUE})$
  - Satisfying: $(x_1=\text{TRUE}, x_2=\text{TRUE}, x_3=\text{TRUE}, x_4=\text{TRUE})$
- Unit rule
- Conflict rule
SAT Problem Definition

- Boolean or propositional satisfiability, also known as SATISFIABILITY (abbreviated as SAT).

\[ E = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4) \]

- SAT problem: Given a Boolean formula \( E \), decide if \( E \) is satisfiable. If so, find the satisfying truth assignment such as \((x_1=\text{TRUE}, x_2=\text{TRUE}, x_3=\text{TRUE}, x_4=\text{TRUE})\).

- First known example of the NP-complete problems.

- Applications: Circuit and hardware design, automatic theorem proving, AI, electronic design and verification, theoretical computer science, etc.
SAT Solver Definition

- An algorithm to solve the SAT problems.
- Exponential worst case running time.
- Inherently complex nature of SAT.
- Capable of solving SAT instances with a few thousands variables and a few hundred thousands clauses.
- Categorized in mainly two classes.
  - Complete solvers
  - Incomplete solvers
SAT Solver ➔ Preliminaries

- Key terminologies used in CDCL.
  - Branching operation
  - Decision/branching variable
  - Implied variable
  - Antecedent clause
  - Decision/assignment stack
  - Decision level
  - Backtracking
SAT Solver Complete

- Solves SAT problems with 100% certainty.
- Davis-Putnam-Logemann-Loveland (DPLL) algorithm, one of the first complete backtracking-based search algorithm.
- A foundation for almost all the modern complete solvers.
- Available in mainly two flavors:
  - Conflict driven solvers
  - Look-ahead solvers

DPLL Search Tree
Source: http://en.wikipedia.org/wiki/DPLL_algorithm

4/4/2014
SAT Solver ➔ Complete ➔ Conflict Driven Solvers

- Designed based on DPLL.
- Assigns the truth value to a variable $x$ selected based on the statistics derived from the current CNF formula.
- Conflict analysis and conflict driven backtracking.
- Conflict Driven Clause Learning (CDCL) algorithm.
- No effect on the soundness or the completeness of the solver.
- Discussed thoroughly in the later sections.
SAT Solver ➔ Complete ➔ Look Ahead Solvers

- Implement DPLL along with conflict analysis and conflict driven backtracking.

- Look-ahead procedure:
  - Reduces the current CNF considering both values of selected variable $x$.
  - Measures the importance of both values of variable $x$.
  - Backtracks and finishes look-ahead.

- It is hoped that evaluation based on the actual truth assignment is more reliable than just guesses based on the statistics derived from the current CNF state.
SAT Solver → Incomplete

- Solve SAT problems with no guarantee of finding the solution.
- Biased on either satisfiable or unsatisfiable instances.
- Theoretically incomplete with respect to both side.
- Designed based on one of the techniques such as Stochastic Local Search (SLS), Evolutionary Algorithms (EAs), translation to Integer Programming, and Finite learning automata.
- Often outperforms complete solvers on randomized instances.
CDCL algorithm

CDCL(\(\varphi, \nu\))

1. if (\textsc{UnitPropagation}(\(\varphi, \nu\)) == \textsc{Conflict})
2. then return \textsc{UNSAT}
3. \(dl \leftarrow 0\) \quad \triangleright \text{Decision level}
4. while (not \textsc{AllVariablesAssigned}(\(\varphi, \nu\)))
5. \quad do \((x, \nu) = \textsc{PickBranchingVariable}(\varphi, \nu)\) \quad \triangleright \text{Decide stage}
6. \quad \quad \quad dl \leftarrow dl + 1 \quad \triangleright \text{Increment decision level due to new decision}
7. \quad \quad \quad \nu \leftarrow \nu \cup \{(x, \nu)\}
8. \quad if (\textsc{UnitPropagation}(\varphi, \nu) == \textsc{Conflict})
9. \quad \quad then \(\beta = \textsc{ConflictAnalysis}(\varphi, \nu)\) \quad \triangleright \text{Deduce stage}
10. \quad \quad if (\beta < 0)
11. \quad \quad \quad then return \textsc{UNSAT}
12. \quad \quad else \ \textsc{Backtrack}(\varphi, \nu, \beta)
13. \quad \quad \quad \quad \quad \quad dl \leftarrow \beta \quad \triangleright \text{Decrement decision level due to backtracking}
14. return \textsc{SAT}

Just an extension of DPLL but with more sophisticated features:

- Boolean Constraint Propagation (BCP)
- Branching heuristic
- Clause learning
- Random restart
- Restricted clause learning
- Non-chronological backtracking
CDCL Algorithm → BCP

- Applies unit clause rule iteratively.
- Continues until
  - No more literals are implied.
  - The conflict is identified (Conflict rule).
- Reduces the depth of SAT’s binary tree search space.
- Consumes 90% of the overall running time.
- Crucial to have highly optimized BCP engine.
A heuristic used to pick the next branching variable.

Directly affects the BCP operation.

Trade off between the required computation/memory and the ability to improve the efficiency.

- Random selection: RAND
- Maximization function: Böhm, Maximum Occurrences in clauses of Minimum Size (MOMS).
- Largest Frequency: Dynamic Largest Individual Sum (DLIS) and Dynamic Largest Combined Sum (DLCS).
CDCL Algorithm → Clause Learning

- Performed when the BCP detects the conflict.
- Deduces the reason of that conflict.
- The conjunction of the responsible variables’ truth assignment.
- A new learned/conflict clause formed using the compliment of that conjunction.
- To avoid repeating the same mistakes (conflict situations).
- Prunes the binary tree search space.
CDCL Algorithm → Random Restart

- Restarts the whole CDCL search procedure without removing the previously learned clauses.
- Compacts the assignment stack and improves the order of assumptions.
- Typically uses the conflict count to trigger the restart.
- Increases the cutoff value of triggering event after every restart to ensure the completeness of the solver.
At least one learned clause for each conflict.
Possible number of conflicts is exponential.
Average learned clause size increases over the time.
Smaller Learned clauses prune larger part of the tree.
Restricted clause learning avoids memory overflow error.
Size-bound, relevance-based, and heuristics activity-based clause deletion strategies
Most of the modern solvers implement combination of more than one strategies.
• Uses a conflict clause to decide backtracking level.
• Conflict driven backtracking.
• Chronological versus non-chronological backtracking.
• Let’s assume the last learned clause is \((- x_4 \lor x_8 \lor x_9\)).

Parallel Complete SAT Solvers Classification

- Classification based on two main factors.
- The approaches used to design the solver.
  - Divide and conquer
  - Portfolio
- The computing resources used to implement the solver.
  - Network communication based grid (Cluster)
  - Shared memory based multi-processor (SMP)
- Hybrid approaches
Parallel Complete SAT Solvers ➔ Divide and Conquer

- Cooperative parallelism
- Split the problem search space using,
  - Classical heuristic based partitioning
  - Dividing the Boolean formula itself
  - Guiding-paths
- Complicated load-balancing techniques
- Pros: Scalability and true parallelism
- Cons: Lack of diversity

4/4/2014
Parallel Complete SAT Solvers Portfolio

- Competitive parallelism.
- Runs multiple diversified CDCL solvers.
- May or may not share information with each other.
- Diversity in terms of branching heuristics, clause learning schemes, clause sharing heuristics, random restart policies, etc.
- No need for load-balancing.
- Pros: Huge diversity
- Cons: Lack of scalability and true parallelism
Parallel Complete SAT Solvers → Cluster-based

- Designed to run on a cluster of single core processing units.
- Designed using either of the two discussed schemes.
- Slave solvers may or may not share information.
- Trade-off between the shared information versus its effectiveness to improve the overall performance.
- Pros: Scalability and cheap commodity hardware.
- Cons: Difficult load balancing and expensive inter-process communication.
Parallel Complete SAT Solvers → SMP-based

- Designed to run on a single shared memory multi-core unit.
- No inter-process communication but limited shared memory.
- Requires sophisticated clause sharing and deletion.
- Theoretically uniform memory access but need to deal with cache coherence.
- Requires wise selection of data-structures and algorithms.
- Pros: No inter-process communication.
- Cons: Limited scalability.
Parallel Complete SAT Solvers ➔ Hybrid

- Solvers designed to run on the grid of multiple SMP computing units.

- Solvers designed by combining divide & Conquer approach with the portfolio approach.

- Achieved using multi-level architecture.
Title: A Hybrid Parallel Complete SAT Solver Using Parallel Java 2.

- Divide and conquer combined with the portfolio approach.
- Designed to run on SMP computing resources.
- Developed in 100% Java using Parallel Java 2 (PJ2) API.
- Highly configurable and adaptable to the available number of cores.
## HyParSAT - Diversity of Portfolios

<table>
<thead>
<tr>
<th></th>
<th>ManySATPortfolio1</th>
<th>ManySATPortfolio2</th>
<th>zChaffPortfolio1</th>
<th>zChaffPortfolio2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching Heuristic</td>
<td>RAND</td>
<td>RAND</td>
<td>VSIDS</td>
<td>VSIDS</td>
</tr>
<tr>
<td>Restart Policy</td>
<td>Geometric Policy $x_{i+1} = 1.5 \times x_i$, with $x_1 = 100$</td>
<td>Arithmetic Policy $x_{i+1} = x_i + 16000$, with $x_1 = 16000$</td>
<td>Geometric Policy $x_{i+1} = 1.5 \times x_i$, with $x_1 = 100$</td>
<td>Arithmetic Policy $x_{i+1} = x_i + 16000$, with $x_1 = 16000$</td>
</tr>
<tr>
<td>Literal Selection for Preassignment</td>
<td>Random Selection</td>
<td>Random Selection</td>
<td>Literal Frequency based Selection</td>
<td>Literal Frequency based Selection</td>
</tr>
</tbody>
</table>
HyParSAT ➔ Portfolio Creation Using PJ2

```java
solver.parallelDo(final Section {  
    new Section() {
        public void run() throws Exception {
            new ManySATPortfolio1(solver)).startPortfolio();
        }
    },
    new Section() {
        public void run() throws Exception {
            (new ManySATPortfolio2(solver)).startPortfolio();
        }
    },
    new Section() {
        public void run() throws Exception {
            (new zChaffPortfolio1(solver)).startPortfolio();
        }
    },
    new Section() {
        public void run() throws Exception {
            (new zChaffPortfolio2(solver)).startPortfolio();
        }
    }
});
```
public void startPortfolio() throws Exception{
    //preselecting some variables
    selectPreAssignedLits();
    long lb = 0, ub = (1 << preAssignedCount) - 1;
    ManySATSlave1.setMasterAndSolver(this.solver);
    //Creating Chunks, processes and schedule tasks using dynamic scheduling
    solver.parallelFor(lb, ub).threads(slaveCount).schedule(Solver.dynamic).
    exec(new LongLoop(){
        // Thread-local variable declarations
        ManySATSlave1 slave;
        public void start(){
            slave = new ManySATSlave1();
        }
        public void run(long i) throws Exception{
            if(!solver.isSolved()){
                System.out.format("%70s \n","ManySATPortfolio1: Slave"+rank()+":chunk["+i+":] Started");
                String message = slave.startSlave(i,rank());
                if(!solver.isSolved())
                    slave.reset();
                System.out.format("%70s \n","ManySATPortfolio1: Slave"+rank()+":chunk["+i+":] Stopped"
                +message);
            }
        }
    });
HyParSAT ➔ Load Balancing Using PJ2

Work Distribution Schedules

CDCL()
    conflictCount = 0
    restartTrigger = x // depends on restart strategy
    while (unassignedLitCount > 0)
        if (isSolved())
            throw new TerminateException
        depth = depth + 1
        v = getNextBranchingVariable()
        setVar(v)
        while (not booleanConstraintPropagation())
            conflictCount = conflictCount + 1
            if (depth == 0)
                return UNSATISFIABLE
            depth = analyzeConflict()
            if (conflictCount == restartTrigger) {
                reset()
                break;
            }
            if (conflictCount % decayRate == 0)
                decayCounters()
                nonChronologicalBacktracking()
            if (conflictCount % 100 == 0)
                deleteNonAssertingBlackClauses()
    return SATISFIABLE
HyParSAT $\rightarrow$ CDCL $\rightarrow$ Branching Heuristics

- **RAND**:  
  - Random variable selection.  
  - The simplest and the fastest heuristic.  
  - Good for only certain SAT instances such as randomized one.

- **Variable State Independent Decaying Sum (VSIDS)**:  
  - Frequency based selection. Independent of the state  
  - Decays frequency periodically.  
  - Recently learned clauses dominate the search process.
Two watch literals based BCP operation
Each clause watches two of its literals.
Each literal maintains a list of watching clauses.
Clauses are visited only when one of the watch literals is assigned FALSE.
Unassignment of literal can be performed in constant time.
No need to modify the clauses while backtracking.
Reduced number of memory accesses and cache miss rate.
HyParSAT → CDCL → Clause Learning

- Uses Implication graph.
- Unique Implication Point (UIP)
- Cut
- 1-UIP clause learning scheme
- Learns clauses more relevant to the conflict.
- Learns shorter clauses compare to other schemes.
- \((\neg x_4 \lor x_8 \lor x_9)\)
HyParSAT ➔ CDCL ➔ Restricted Clause Learning

- Combination of size-bounded and relevance-based learning strategies.

- Global clause sharing:
  - $k$-ordered learning scheme, where $k = 8$.
  - Shares a clause only if its size $\leq 8$.

- Local clause deletion:
  - $k$-bounded learning scheme, where $k = 8$.
  - Periodically deletes clauses with size $\leq 8$. 
HyParSAT $\Rightarrow$ CDCL $\Rightarrow$ Random Restart

- Fast geometric restart policies:
  - $X_i = 1.5 \times X_{i-1}$ with $X_1 = 100$.
  - Good for well structured industrial SAT instances.

- Slow arithmetic restart policy:
  - $X_i = X_{i-1} + 16000$ with $X_1 = 16000$.
  - Good for hard or randomized instances.
<table>
<thead>
<tr>
<th>Case No.</th>
<th>Instance</th>
<th>Variables</th>
<th>Clauses</th>
<th>Portfolio</th>
<th>Cores per portfolio</th>
<th>DB size</th>
<th>CPU Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>qg4-08</td>
<td>512</td>
<td>9685</td>
<td>M1M2Z1Z2</td>
<td>1</td>
<td>1000</td>
<td>8.196</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M2Z2</td>
<td>1</td>
<td>1000</td>
<td>5.803</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z2</td>
<td>1</td>
<td>500</td>
<td>1.027</td>
</tr>
<tr>
<td>2</td>
<td>qg7-11</td>
<td>1331</td>
<td>49534</td>
<td>Z2</td>
<td>8</td>
<td>500</td>
<td>8.065</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z2</td>
<td>4</td>
<td>500</td>
<td>6.801</td>
</tr>
<tr>
<td>3</td>
<td>qg7-12</td>
<td>1728</td>
<td>70327</td>
<td>Z2</td>
<td>8</td>
<td>500</td>
<td>93.890</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z2</td>
<td>16</td>
<td>500</td>
<td>43.580</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z2</td>
<td>16</td>
<td>2000</td>
<td>37.981</td>
</tr>
<tr>
<td>4</td>
<td>Qg7-13</td>
<td>2197</td>
<td>97072</td>
<td>M1M2Z1Z2</td>
<td>4</td>
<td>2000</td>
<td>3.054</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M1Z1</td>
<td>4</td>
<td>2000</td>
<td>80.495</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M2Z2</td>
<td>4</td>
<td>2000</td>
<td>3.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z1Z2</td>
<td>4</td>
<td>2000</td>
<td>3.797</td>
</tr>
</tbody>
</table>
### SATISFIABLE instance

<table>
<thead>
<tr>
<th>Instance</th>
<th>Variable</th>
<th>Clauses</th>
<th>HyParSAT (seconds)</th>
<th>MiniSat (seconds)</th>
<th>ManySAT (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hanoi4.caf</td>
<td>718</td>
<td>4934</td>
<td>1.710 s</td>
<td>0.040</td>
<td>0.056</td>
</tr>
<tr>
<td>aim-200-1_6-no-1-1</td>
<td>200</td>
<td>320</td>
<td>0.056</td>
<td>0.001</td>
<td>0.04</td>
</tr>
<tr>
<td>aim-200-1_6-no-1-2</td>
<td>200</td>
<td>320</td>
<td>0.093</td>
<td>0.001</td>
<td>0.004</td>
</tr>
<tr>
<td>bw_large_a</td>
<td>459</td>
<td>4675</td>
<td>0.112</td>
<td>0.001</td>
<td>0.008</td>
</tr>
<tr>
<td>bw_large_b</td>
<td>1087</td>
<td>13772</td>
<td>0.241</td>
<td>0.008</td>
<td>0.016</td>
</tr>
<tr>
<td>bw_large_c</td>
<td>3016</td>
<td>50457</td>
<td>0.826</td>
<td>0.040</td>
<td>0.060</td>
</tr>
<tr>
<td>bw_large_d</td>
<td>6325</td>
<td>131973</td>
<td>400.019</td>
<td>0.200</td>
<td>0.208</td>
</tr>
<tr>
<td>qg2-08</td>
<td>512</td>
<td>148597</td>
<td>480</td>
<td>1.480</td>
<td>2.408</td>
</tr>
<tr>
<td>qg7-09</td>
<td>729</td>
<td>22060</td>
<td>0.11</td>
<td>0.008</td>
<td>0.012</td>
</tr>
<tr>
<td>qg2-07</td>
<td>343</td>
<td>68083</td>
<td>0.354</td>
<td>0.012</td>
<td>0.020</td>
</tr>
<tr>
<td>qg7-13</td>
<td>2197</td>
<td>97072</td>
<td>3.03</td>
<td>0.228</td>
<td>0.076</td>
</tr>
<tr>
<td>hanoi5</td>
<td>1931</td>
<td>14468</td>
<td>Stopped (2hr)</td>
<td>3.1802</td>
<td>0.836</td>
</tr>
<tr>
<td>vmpc_27</td>
<td>729</td>
<td>96849</td>
<td>Stopped (2hr)</td>
<td>413.93</td>
<td>1033.69</td>
</tr>
<tr>
<td>vmpc_28</td>
<td>784</td>
<td>108080</td>
<td>Stopped (2hr)</td>
<td>141.057</td>
<td>8095.92</td>
</tr>
<tr>
<td>vmpc_31</td>
<td>901</td>
<td>146909</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
</tr>
<tr>
<td>f600</td>
<td>600</td>
<td>2550</td>
<td>Stopped (12hr)</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
</tr>
<tr>
<td>g125_17</td>
<td>2125</td>
<td>66272</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
</tr>
<tr>
<td>g125_18</td>
<td>2250</td>
<td>70163</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
</tr>
<tr>
<td>g250_15</td>
<td>3750</td>
<td>239365</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
</tr>
<tr>
<td>g250_29</td>
<td>7550</td>
<td>454622</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
<td>Stopped (2hr)</td>
</tr>
<tr>
<td>md5_47_4</td>
<td>65604</td>
<td>273506</td>
<td>Stopped (3hr)</td>
<td>418.218</td>
<td>1118.48</td>
</tr>
<tr>
<td>md5_48_1</td>
<td>66892</td>
<td>279248</td>
<td>Stopped (3hr)</td>
<td>556.619</td>
<td>2685.73</td>
</tr>
</tbody>
</table>

### UNSATISFIABLE Instance

<table>
<thead>
<tr>
<th>Instance</th>
<th>Variable</th>
<th>Clauses</th>
<th>HyParSAT (seconds)</th>
<th>MiniSat (seconds)</th>
<th>ManySAT (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aim-200-1_6-no-3</td>
<td>200</td>
<td>320</td>
<td>0.054</td>
<td>0.001</td>
<td>0.004</td>
</tr>
<tr>
<td>aim-200-1_6-no-4</td>
<td>200</td>
<td>320</td>
<td>0.09</td>
<td>0.001</td>
<td>0.01</td>
</tr>
<tr>
<td>dhbois100</td>
<td>300</td>
<td>860</td>
<td>0.11</td>
<td>Error</td>
<td>0.004</td>
</tr>
<tr>
<td>qg4-08</td>
<td>512</td>
<td>9685</td>
<td>1.027</td>
<td>0.016</td>
<td>0.063</td>
</tr>
<tr>
<td>qg2-09</td>
<td>729</td>
<td>16732</td>
<td>854.713</td>
<td>1.404</td>
<td>4.668</td>
</tr>
<tr>
<td>qg7-10</td>
<td>1000</td>
<td>33735</td>
<td>0.217</td>
<td>0.004</td>
<td>0.016</td>
</tr>
<tr>
<td>qg7-11</td>
<td>1331</td>
<td>49534</td>
<td>7.629</td>
<td>0.016</td>
<td>0.056</td>
</tr>
<tr>
<td>qg7-12</td>
<td>1728</td>
<td>76327</td>
<td>37.981</td>
<td>0.120</td>
<td>0.332</td>
</tr>
<tr>
<td>ssa6288-047</td>
<td>10410</td>
<td>34238</td>
<td>0.996</td>
<td>0.004</td>
<td>0.016</td>
</tr>
<tr>
<td>ndhf_nits_13_UNSAT</td>
<td>2730</td>
<td>271255</td>
<td>Stopped (3hr)</td>
<td>Stopped (3hr)</td>
<td>390.944</td>
</tr>
<tr>
<td>q_query_3_l48_lambda</td>
<td>34656</td>
<td>174528</td>
<td>Stopped (3hr)</td>
<td>76.112</td>
<td>393.573</td>
</tr>
</tbody>
</table>
HyParSAT, a completely new approach that combines divide & conquer scheme with portfolio scheme.

Implemented in 100% Java for SMP computing resources.

Highly Configurable and adaptable to the available cores.

Unachievable true parallelism due to inherent complexity.

MiniSAT occasionally outperforms parallel solvers.

HyParSAT is not up to the mark as opposed to the best solvers in the industry.

HyParSAT may perform better after certain improvements.
Questions