SAT-Based CEGAR Method for the Hamiltonian Cycle Problem Enhanced by Cut-Set Constraints

Ryoga Ohashi
Kobe University, Kobe, Japan

Takehide Soh
Kobe University, Kobe, Japan

Daniel Le Berre
Univ. Artois, CNRS, UMR 8188 CRIL, Lens, France

Hidetomo Nabeshima
Yamanashi University, Kofu, Japan

Mutsunori Banbara
Nagoya University, Nagoya, Japan

Katsumi Inoue
National Institute of Informatics, Tokyo, Japan

Naoyuki Tamura
Kobe University, Kobe, Japan

Abstract

In this paper, we propose an enhancement to the SAT-based counterexample-guided abstraction refinement (CEGAR) approach for solving the Hamiltonian Cycle Problem (HCP). Many SAT-based methods for HCP have been proposed, including a CEGAR-based method that repeatedly solves a relaxed version of HCP strengthened by counterexamples. However, when the counterexample space – represented by the full set of subcycle partitions – is large, it becomes difficult to find a solution. To address this, we introduce *cut-set constraints* in the refinement step, replacing traditional subcycle blocking constraints. Our evaluation shows that these cut-set constraints achieve equal or better reduction in the counterexample space, making it easier to find valid solutions. We further assessed performance using all 1001 instances from the FHCP challenge set and confirmed that the proposed method solved 937 instances within 1800 seconds, outperforming both the existing eager and CEGAR encodings (which solved at most 666 instances). This demonstrates the effectiveness of incorporating cut-set constraints into SAT-based CEGAR approaches.

2012 ACM Subject Classification Computing methodologies → Discrete space search

Keywords and phrases Hamiltonian Cycle Problem, SAT Encoding, CEGAR

Digital Object Identifier 10.4230/LIPIcs.SAT.2025.24

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.15621774`

Funding This work was financially supported by JSPS KAKENHI (23K11047), and by ROIS NII Open Collaborative Research 2024 (24FP04).
Takehide Soh: JSPS KAKENHI Grant Number 23K11047.
Mutsunori Banbara: JSPS KAKENHI Grant Number 25K03097.
Katsumi Inoue: JSPS KAKENHI Grant Number JP25K03190, JST CREST Grant Number JP-MJCR22D3.

1 Introduction

The *Hamiltonian Cycle Problem (HCP)* asks whether a given graph contains a cycle that visits each node exactly once. HCP is one of the 21 NP-complete problems identified by Karp in 1972 [10]. Due to its close link with the Traveling Salesman Problem (TSP), efficient methods for HCP have remained a key research focus. As a result, *FHCP Challenge* [7] was recently held as a year-long challenge aimed at tackling particularly difficult HCP instances.

The *Satisfiability (SAT) Problem* is another NP-complete problem, deciding whether there exists an assignment of truth values to propositional variables that makes a given logical formula true. Despite the theoretical complexity of SAT, SAT solvers are very efficient in practice [2, 6] on a wide range of combinatorial problems reduced to SAT, including HCP [11, 14, 13, 15, 8, 16]. SAT-based methods for solving HCP usually depend on degree constraints and connectivity constraints. Various SAT encodings for these constraints have been studied. Recently, binary adder encoding [15] showed good performance for selected 7 instances of the FHCP Challenge. Combined with degree constraints, it assigns a total order to each node connected in a solution graph, and a binary adder manages the order. Subsequent work [8] utilizes the Chinese remainder theorem (CRT) to further shrink the number of clauses generated by binary adder encoding. Another approach is vertex elimination [16], which avoids no-sub-cycle constraints unlike other methods. However, these methods suffer from scalability issues or excessive clause generation on challenging FHCP instances, making a more efficient SAT-based method a key challenge.

To overcome the limitations of existing SAT-based methods, in this paper, we revisit the SAT-based CEGAR method [13]. This approach uses a SAT solver to solve a relaxed version of the HCP that consists solely of degree constraints. If the obtained solution contains subcycles (counterexamples), the method iteratively adds subcycle blocking constraints to eliminate them; otherwise, it returns the obtained solution as a Hamiltonian cycle. This approach has the advantage of not needing to encode upfront the connectivity constraint, which tends to result in a large number of clauses. However, for difficult instances, the issue is that the number of counterexamples found is huge, preventing the SAT solver from finding a Hamiltonian cycle. The main idea of the proposed method is to use *cut-sets*, which are sets of edges that connect subcycles to other subcycles, and add constraints to include these edges in the solution rather than relying on traditional methods that only add constraints to eliminate the detected subcycles. This improvement allows for the elimination of more subcycles compared to previous methods, leading to better performance. To evaluate the effectiveness of our proposed method, we conducted experiments using all 1001 benchmark instances from the FHCP Challenge. We also compared our approach with recently proposed SAT-based methods [13, 8, 16] to further assess its performance.

2 Preliminaries

Hamiltonian cycle problem (HCP). Let $G = (V, E)$ be a graph where V is a set of n nodes and E is a set of edges. Let C be a simple cycle in G. We also denote a cycle by a sequence of nodes $(v_{i_1}, v_{i_2}, \dots, v_{i_k})$, where $v_{i_j} \in V$ are all distinct and $\{v_{i_j}, v_{i_{j+1}}\} \in E$ for all $1 \leq j < k$, with $\{v_{i_k}, v_{i_1}\} \in E$. We also sometimes treat a cycle C as a set of edges. We use $|C|$ as the size of the cycle C and use $V(C)$ as the set of nodes in a cycle C, i.e., the set of nodes consisting of the cycle C. We also denote by G_C the subgraph of G induced by $V(C)$, i.e., $G_C = G[V(C)]$. The *Hamiltonian cycle problem* (HCP) is the problem of finding a cycle C s.t. $V(C) = V$.

Constraints of HCP. Let $G' = (V, E')$ be a subgraph (or a solution graph) of G where $E' \subseteq E$. Let x_{ij} $(i \neq j)$ be a Boolean variable for each edge $\{i, j\} \in E$, which is equal to 1 when $\{i, j\} \in E'$ is true. Then, constraints of HCP can be written as follows: $\Psi_{DEG} := \bigwedge_{i=1}^{n} \sum_{\{i,j\} \in E} x_{ij} = 2$ and $\Psi_{CON} := \bigwedge_{\substack{S \subset V \\ 2 \leq |S| \leq n-2}} \sum_{i,j \in S} x_{ij} \leq |S| - 1$. The *degree constraints* Ψ_{DEG} force that, for each node, its degree is exactly two in G'. Consequently, it forces each node to belong to a subcycle in G'. The *connectivity constraint* Ψ_{CON} prohibits subcycles, i.e., cycles on proper subsets of V. As mentioned above, compared to degree constraints (which can be encoded with a linear number of clauses to the number of nodes), connectivity constraints are more expensive in SAT encoding; in the worst case of a naive encoding, an exponential number of clauses is necessary.

Algorithm 1 SAT-based CEGAR for HCP.

```
Input: Graph $G = (V, E)$
Output: Hamiltonian cycle (if exists), or “No”
1: $\Psi :=$ DegreeConstraint($G$)
2: while $\Psi$ is satisfiable do
3:     $P :=$ Decode the model of $\Psi$ found        ▷ $P$ is a subcycle partition of $G$
4:     if $|P| == 1$ then
5:         return $P$                                  ▷ Hamiltonian cycle found
6:     end if
7:     $\Psi := \Psi \wedge$ RefinementConstraints($P$)
8: end while
9: return “No”
```

Subcycle Partition. Consider a set of cycles $P = \{C_1, C_2, \dots, C_k\}$. We say P is a *subcycle partition* of a given graph $G = (V, E)$ iff $V(C_1) \cup V(C_2) \cup \cdots \cup V(C_k) = V$ and $V(C_i) \cap V(C_j) = \varnothing$ for all $1 \leq i < j \leq k$. In other words, each node of G belongs to exactly one cycle in the set P, and together, these cycles cover every node in the graph without overlap. Clearly, a subcycle partition P contains a Hamiltonian cycle iff $|P| = 1$. Let Π be a function that maps a graph G to the set of all its subcycle partitions. Define $\Pi^{>1}$ as another function that maps a graph G to all its subcycle partitions excluding Hamiltonian cycles, i.e., $\Pi^{>1}(G) = \{\, P \in \Pi(G) \mid |P| > 1\}$.

Cut-set. A *cut* (S, T) is a partition of the node set V of a graph $G = (V, E)$ into two disjoint subsets S and T. The *cut-set* of a cut (S, T) is defined as $\{(u, v) \in E \mid u \in S,\ v \in T\}$, which is the set of edges having one endpoint in S and the other endpoint in T.

3 Existing SAT-based CEGAR for HCP [13]

Unlike typical SAT-based methods, SAT-based *Counterexample-Guided Abstraction Refinement* (CEGAR) relaxes constraints and incrementally uses a SAT solver on a partial HCP encoding, instead of fully encoding connectivity into CNF at once.

The algorithm of SAT-based CEGAR for HCP is presented in Algorithm 1. In Line 1, the formula Ψ is initialized with the degree constraints Ψ_{DEG}. Line 2 checks satisfiability repeatedly. If satisfiable, Line 3 decodes a subcycle partition P from the obtained model. If Line 4 detects that P is a single cycle, Line 5 immediately returns it as a Hamiltonian cycle. Otherwise, Line 7 adds the refinement constraints derived from P to refine Ψ, and Line 8 resumes the checking. Finally, when no further solution can be found, Line 9 concludes that no Hamiltonian cycle exists.

If we initialize the abstraction using only degree constraints, the counterexamples are always a subcycle partition $P = \{C_1, C_2, \dots, C_m\}$ of G. Then, in the existing SAT-based CEGAR [13], `RefinementConstraints`(P) is implemented by `SubCycleBlocking`(P) as follows:

$$\texttt{SubCycleBlocking}(P) := \bigwedge_{C \in P} \bigvee_{\{u,v\} \in C} \neg x_{u,v} \tag{1}$$

An illustration of a possible counterexample is shown in Figure 1. In this example, the subcycle partition consists of $C_1 = (1, 2, 3, 10, 11, 12)$ and $C_2 = (4, 5, 6, 7, 8, 9)$. For this

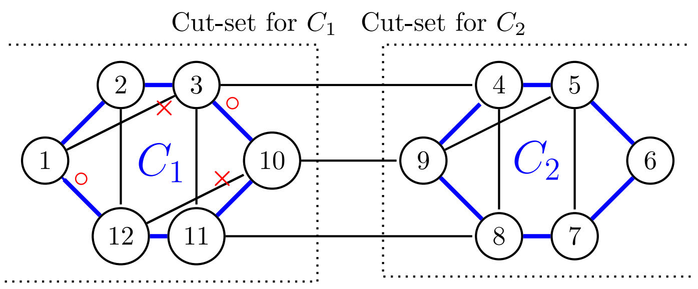


Figure 1 A counterexample consists of the subcycle partition $P = \{C_1, C_2\}$ where $C_1 = (1, 2, 3, 10, 11, 12)$ and $C_2 = (4, 5, 6, 7, 8, 9)$. The dashed line represent the cut-sets between the set of nodes $V(C_i)$ (i.e., S) and the rest of the graph $V \setminus V(C_i)$ (i.e., T), as explained in Section 4.1. The × marks indicate the deleted edges {3,1} and {12,10}, while the ∘ marks indicate the added edges {3,10} and {1,12} according to the `MergeCycles` operation explained in Section 4.3.

counterexample, the existing SAT-based CEGAR refines constraints Ψ by two blocking clauses. In the case of the given example, the following two clauses are generated.

$$\neg x_{1,2} \vee \neg x_{2,3} \vee \neg x_{3,10} \vee \neg x_{10,11} \vee \neg x_{11,12} \vee \neg x_{12,1}$$
$$\neg x_{4,5} \vee \neg x_{5,6} \vee \neg x_{6,7} \vee \neg x_{7,8} \vee \neg x_{8,9} \vee \neg x_{9,4} \tag{2}$$

Note that, even in the worst case, we do not always need to block all subcycle partitions since each subcycle is forbidden independently. For instance, if we refine the constraints for the subcycle partition $\{C_1, C_2\}$, then we do not need to care $\{C_1, (4, 8, 9), (5, 6, 7)\}$ since C_1 cannot appear after the first refinement. And it is not necessary to block all subcycles, e.g. $(2, 3, 11, 12)$ since the remaining nodes cannot construct any subcycle partition.

From another perspective, this CEGAR approach can be regarded as a method for turning the following eager encoding into a lazy encoding.

$$\Psi_{SCB} := \bigwedge_{P \in \Pi^{>1}(G)} \bigwedge_{C \in P} \bigvee_{\{u,v\} \in C} \neg x_{u,v} \tag{3}$$

Instead of the connectivity constraints Ψ_{CON} presented in Section 2, the conjunction $\Psi_{DEG} \wedge \Psi_{SCB}$ ensures connectivity, forcing any solution graph to form a Hamiltonian cycle.

4 Proposed SAT-based CEGAR Enhanced by Cut-Set Constraints

This paper proposes an improvement to the existing SAT-based CEGAR method. While the basic algorithm remains the same, we modify how refinement constraints are added.

4.1 Cut-Set Constraints

In the following, we will denote the cut-set of (S, T) simply as $\Delta(S)$. Our idea is to replace the base constraint Ψ_{SCB} shown in the equation (3) by the following constraint[1]:

$$\Psi_{CSF} := \bigwedge_{P \in \Pi^{>1}(G)} \bigwedge_{C \in P} \bigvee_{\{u,v\} \in \Delta(V(C))} x_{u,v} \tag{4}$$

Then, the following proposition holds.

[1] Cut-set based constraints have been studied in the context of the TSP solving (e.g., [4]). We adapt these ideas to the setting of SAT-based CEGAR for HCP.

▶ **Proposition 1.** Any model of $\Psi_{DEG} \wedge \Psi_{CSF}$ forms a Hamiltonian cycle of G.

Proof. The constraint Ψ_{DEG} ensures that any solution graph can be decomposed into a subcycle partition of G. Meanwhile, the constraint Ψ_{CSF} ensures that for each subcycle in the subcycle partition, there is at least one edge connecting it to its complement. This forbids any subcycle partition in $\Pi^{>1}(G)$ that is not a single cycle. Consequently, any model of $\Psi_{DEG} \wedge \Psi_{CSF}$ necessarily constructs a Hamiltonian cycle. ◀

We then propose to implement the function `RefinementConstraints`(P) of Algorithm 1 by `CutSetForcing`(P) as follows (see also Figure 1 for an example):

$$\texttt{CutSetForcing}(P) := \bigwedge_{C \in P} \bigvee_{\{u,v\} \in \Delta(V(C))} x_{u,v} \tag{5}$$

For the same counterexample $P = \{C_1, C_2\}$ shown in Figure 1, the following clause is generated by `CutSetForcing`(P).

$$x_{3,4} \vee x_{9,10} \vee x_{8,11} \tag{6}$$

Note that there is a single clause since this example has the partition $|P| = 2$, and there is only one cut-set constraint for two subcycles. However, in general, we need the $|P|$ clauses as same as `SubCycleBlocking`(P).

4.2 Properties of Cut-Set Constraints

The advantage of using the proposed cut-set constraints instead of the existing subcycle blocking constraints is its ability to better prune the counterexample space $\Pi^{>1}(G)$. In general, the following proposition holds.

▶ **Proposition 2.** Given a graph $G = (V, E)$ and a subcycle partition P, the number of subcycle partitions that `CutSetForcing`(P) prunes from $\Pi^{>1}(G)$ is *equal to or greater than* the number that `SubCycleBlocking`(P) prunes.

Proof. Both clauses commonly forbid individual subcycles contained in P. Thus, all subcycle partitions that contain those subcycles are pruned from $\Pi^{>1}$, which can be represented by $\Theta_1 = \{P' \in \Pi^{>1}(G) \mid \{C\} \subseteq P' \wedge C \in P\}$. Clearly, `SubCycleBlocking`$(P)$ cannot prune subcycle partitions other than Θ_1. On the other hand, `CutSetForcing`(P) additionally prunes $\Theta_2 = \{P' \in \Pi^{>1}(G) \mid P'' \subseteq P' \wedge P'' \in \bigcup_{C \in P} (\Pi(G_C) \setminus \{C\})\}$ where G_C is a subgraph of G induced by C. ◀

To illustrate the example, we assign the following symbols $C_1, \ldots, C_{12}$ to the subcycles in Figure 1: $C_1 = (1,2,3,10,11,12)$, $C_2 = (4,5,6,7,8,9)$, $C_3 = (1,2,3)$, $C_4 = (10,11,12)$, $C_5 = (1,2,12)$, $C_6 = (3,10,11)$, $C_7 = (4,8,9)$, $C_8 = (5,6,7)$, $C_9 = (1,2,12,10,11,3)$, $C_{10} = (1,2,12,11,10,3)$, $C_{11} = (1,2,3,11,10,12)$, $C_{12} = (4,9,5,6,7,8)$.

Again, suppose that $P = \{C_1, C_2\}$ is obtained as a counterexample. Then, two refining clauses in formulae (2) and (6) commonly forbid the following subcycle partitions, which include an element of $P = \{C_1, C_2\}$. The pruned partitions Θ_1 become as follows:

$$\{\{C_1, C_2\}, \{C_1, C_7, C_8\}, \{C_1, C_{12}\}, \{C_2, C_3, C_4\}, \{C_2, C_5, C_6\}, \{C_2, C_9\}, \{C_2, C_{10}\}, \{C_2, C_{11}\}\}$$

In addition, the proposed constraint in formula (6) prohibits the following subcycle partitions, which include an element of $\{\{C_3, C_4\}, \{C_5, C_6\}, \{C_9\}, \{C_{10}\}, \{C_{11}\}\} = \Pi(G_{C_1}) \setminus \{C_1\}$ or $\{\{C_7, C_8\}, \{C_{12}\}\} = \Pi(G_{C_2}) \setminus \{C_2\}$ – subcycle partitions of G induced by C_1 or C_2.

$$\begin{aligned}\Theta_2 =& \{\{C_3, C_4, C_7, C_8\}, \{C_5, C_6, C_7, C_8\}, \{C_9, C_7, C_8\}, \{C_{10}, C_7, C_8\}, \{C_{11}, C_7, C_8\}, \\ & \{C_3, C_4, C_{12}\}, \{C_5, C_6, C_{12}\}, \{C_9, C_{12}\}, \{C_{10}, C_{12}\}, \{C_{11}, C_{12}\}\}\end{aligned}$$

Algorithm 2 Procedure of Merging Subcycles.

```
Input: Subcycle Partition $P = \{C_1, C_2, \dots, C_m\}$
Output: Smaller Subcycle Partition $P = \{C'_1, C'_2, \dots, C'_{m'}\}$
1: while there exists a pair $(C_i, C_j) \in P$ such that MergeCycles$(C_i, C_j) \neq ()$ do
2:     $C_{\text{merged}} :=$ MergeCycles$(C_i, C_j)$
3:     $P := (P \setminus \{C_i, C_j\}) \cup \{C_{\text{merged}}\}$
4: end while
5: return $P$
```

When subcycles in a counterexample contain multiple subcycle partitions, Θ_2 becomes exponentially large. The reason is that existing methods prune only subcycle partitions containing the counterexample subcycle. In contrast, the proposed method enforces a cut-set, which prunes *all* subcycle partitions enclosed by the subgraph induced by that subcycle. In the above case, the difference arises because the subgraph induced by C_1 has multiple subcycle partitions – C_1, C_9, C_{10}, C_{11}, C_3, C_4, and C_5, C_6 – similarly for C_2.

In other words, if the subgraph induced by the counterexample subcycle does not have multiple subcycle partitions, then $\Theta_2 = \varnothing$, and there is no difference between the existing approach and the proposed approach. In order to use the pruning capability of the proposed method – i.e., to increase the cardinality of Θ_2 – we perform subcycle merging, as described in the next section.

4.3 Merging Subcycles for Improving Runtime

The previous section suggests that, when the induced subgraph of a discovered subcycle has other subcycle partitions, it prunes a larger number of subcycle partitions. We thus introduce a post-processing method that merges subcycles to utilize the advantage of the proposed refinement constraints. The post-processing procedure is shown in Algorithm 2. Its input is a subcycle partition, and it returns a smaller partition if possible. This algorithm searches for a pair of cycles that can be merged using a function `MergeCycles`. If such a pair is found, the algorithm replaces them with the merged cycle. This process is repeated until no mergeable pair remains. Let $C_i = (v_1, v_2, \dots, v_k)$ and $C_j = (u_1, u_2, \dots, u_l)$ be two subcycles to merge. Like the 2-opt method in TSP [3], `MergeCycles` are realized as follows.

$$\texttt{MergeCycles}(C_i, C_j) :=$$
$$\begin{cases} \texttt{Swap}\big(C_i, C_j, \{s, t\}, \{s+1, t+1\}\big), & \text{if } \exists\, s, t \text{ such that } v_s \in V(C_i),\ u_t \in V(C_j), \\ & \{v_s, u_t\} \in E,\ \{v_{s+1}, u_{t+1}\} \in E, \\ \texttt{Swap}\big(C_i, C_j, \{s, t+1\}, \{s+1, t\}\big), & \text{if } \exists\, s, t \text{ such that } v_s \in V(C_i),\ u_t \in V(C_j), \\ & \{v_s, u_{t+1}\} \in E,\ \{v_{s+1}, u_t\} \in E, \\ (), & \text{otherwise.} \end{cases}$$

Here, `Swap` forms a single cycle combining C_i and C_j by replacing the edges $\{v_s, v_{s+1}\}$ and $\{u_t, u_{t+1}\}$ with two given edges. For example, suppose that the subcycles $(1, 2, 3)$ and $(10, 11, 12)$ are found in the instance graph shown in Figure 1. Then we can merge the two subcycles by replacing the edges $\{3, 1\}$ and $\{12, 10\}$ with $\{3, 10\}$ and $\{1, 12\}$ that will create the larger subcycle $(1, 2, 3, 10, 11, 12)$.

4.4 Incorporating Existing Techniques for Efficiency

Conversion to Directed Graph. For simplicity, we have explained the method using undirected graphs so far. However, the method can be naturally extended to handle directed graphs. In fact, as shown in previous work [13], experiments suggest that applying the method to directed graphs yields better performance. Therefore, in the subsequent sections, we use the directed-graph version of the method. When an undirected graph $G = (V, E)$ is given, we introduce a set of auxiliary arcs $A = \{(i, j),\ (j, i) \mid \{i, j\} \in E\}$. And, the degree constraint becomes $\Psi_{DEG} = \bigwedge_{i=1}^{n} \sum_{(i,j)\in A} x_{ij} = 1 \wedge \bigwedge_{j=1}^{n} \sum_{(i,j)\in A} x_{ij} = 1$. The refinement constraint becomes as follows. Here, Δ^{in} and Δ^{out} are incoming and outgoing arcs in the cut-set, respectively.

$$\texttt{CutSetForcing}(P) := \bigwedge_{C_i \in P} \left(\bigvee_{(u,v)\in\Delta^{in}(C_i)} x_{u,v} \wedge \bigvee_{(u,v)\in\Delta^{out}(C_i)} x_{u,v} \right) \tag{7}$$

Hint Constraints. Because a large number of cycles of two nodes occur when the undirected graph is converted to a directed graph, an additional constraint is introduced to prevent them [13]. We refer to it as "2loop." Furthermore, symmetry is broken by directing the edges to uniquely determine the orientations of the cycles [8]. We refer to it as "asym."

5 Experiments

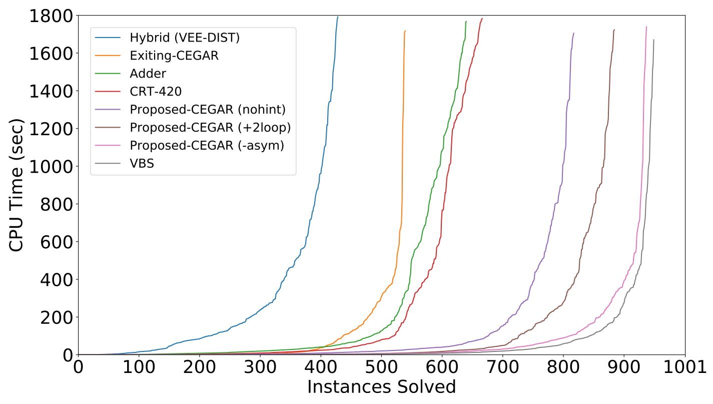
5.1 Set Up

Computer. Core i5 12400 (2.5GHz) CPU with 6 cores, 64GB of RAM. A single thread is used for each run in the experiments. Timelimit is 1800 seconds per instance.

Benchmarks. Recent studies [15, 8] used 7 instances selected from the FHCP Challenge Set[2] [7]. In contrast, we used all 1001 instances provided in the benchmark set to conduct comprehensive experiments.

Compared Methods. We compared the proposed method with an existing CEGAR approach [13] (Existing-CEGAR), as well as three recently proposed SAT encodings (Adder [15], CRT-420 [8], Hybrid (VEE-DIST) [16], respectively) for the Hamiltonian cycle problem. For Adder and CRT-420, we used an implementation available on GitHub[3]. For VEE-DIST, we use the recent version of Picat[4] (v3.8 with Kissat). Note that Existing-CEGAR is implemented using Rustsat [9] as well as the proposed CEGAR. In addition to those tools, we also tried to compare another implementation of the existing CEGAR[5], but could not run it reliably (418 instances were aborted by OS signals). Therefore, we do not include its performance.

[2] `https://sites.flinders.edu.au/flinders-hamiltonian-cycle-project/fhcp-challenge-set/`
[3] `https://github.com/marijnheule/ChineseRemainderEncoding`
[4] `https://picat-lang.org/`
[5] `https://github.com/kfazekas/incremental-examples`



Hybrid [16]	428
E. CEGAR [13]	539
Adder [15]	640
CRT-420 [8]	666
P. CEGAR (nohint)	817
P. CEGAR (+2loop)	884
P. CEGAR (-asym)	937
VBS	949

Figure 2 Cactus Plot of Proposed Method and SAT-based Prior Work.

Implementation. To encode the degree constraints in CNF, we have used sequential counter encoding [12]. We have implemented the proposed method on Rustsat. The implementation is available [6]. We use CaDiCaL [1] for all methods except Picat.

5.2 Result

Ablation study of the proposed approaches. We first evaluate the three acceleration options "2loop," "asym," "merge" described in Section 4.3 and Section 4.4. The number of solved instances is shown in the table below where the symbol + denotes "nohint" plus one additional option, and the symbol - denotes "all" minus some individual option. The Virtual Best Solver (VBS) is a hypothetical solver which represents the best possible performance among all evaluated options.

	# Ins.	nohint	+2loop	+asym	+merge	-merge	-asym	-2loop	all	VBS
Total	1001	817	884	822	858	895	**937**	863	936	948

The results show that "2loop" and "merge" are essential, and "-asym" performs best.

Comparisons with Existing SAT-based Methods. Figure 2 presents a cactus plot, where each approach is represented by a curve. On the x-axis, we have the number of instances solved within a specified time limit, and on the y-axis, the required timelimit for solving those instances is shown. Among individual approaches, the proposed CEGAR variants solved more instances (937) within the timeout. Meanwhile, the existing CEGAR method solved significantly fewer instances than the proposed approaches. Because the existing CEGAR method includes "2loop," the difference compared to "+2loop" can be viewed as the impact of the proposed cut-set constraints. Note that CRT-420 is also a type of abstraction method and can occasionally return counterexamples. For this figure, we plotted only the problems for which a correct solution was returned, totaling 666. Even if we consider counterexamples of CRT-420 as valid solutions, the number of instances solved would be 733, whereas the proposed method successfully solved more instances.

[6] `https://doi.org/10.5281/zenodo.15621774`

6 Conclusion

In this paper, we propose an enhancement to the existing SAT-based counterexample-guided abstraction refinement (CEGAR) approach to solving the Hamiltonian cycle problem (HCP). Our key contribution is to replace the conventional constraints for subcycle blocking with cut-set constraints. By doing so, we can prune a larger portion of the counterexample space, especially in cases where subcycles contain nested subpartitions. In addition, we introduced an acceleration technique, merging subcycles, which further speeds up the proposed method. In our experiments, we confirmed that the proposed method exhibited improved performance compared to existing CEGAR and eager encoding methods. While the existing approach solved 666 out of 1001 FHCP challenge set instances, the proposed method solved 937 instances within 1800 seconds. Future work includes a more efficient implementation using IPASIR-UP [5] and using parallelization to achieve further speed-ups.

References

1 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

2 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. doi:10.3233/FAIA336.

3 Georges A Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

4 George B. Dantzig, Delbert R. Fulkerson, and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. doi:10.1287/OPRE.2.4.393.

5 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. Satisfiability modulo user propagators. *J. Artif. Intell. Res.*, 81:989–1017, 2024. doi:10.1613/JAIR.1.16163.

6 Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artif. Intell.*, 301:103572, 2021. doi:10.1016/J.ARTINT.2021.103572.

7 Michael Haythorpe. FHCP Challenge Set: The first set of structurally difficult instances of the Hamiltonian cycle problem, 2019. arXiv:1902.10352.

8 Marijn J. H. Heule. Chinese remainder encoding for Hamiltonian cycles. In Chu-Min Li and Felip Manyà, editors, *Proc of 24th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021)*, volume 12831 of *LNCS*, pages 216–224. Springer, 2021. doi:10.1007/978-3-030-80223-3_15.

9 Christoph Jabs. RustSAT: A library for SAT solving in Rust. In *Proc of 28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025)*, volume 341 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SAT.2025.15.

10 Richard M. Karp. *Reducibility among Combinatorial Problem*, pages 85–103. Springer, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

11 Steven D. Prestwich. SAT problems with chains of dependent variables. *Discret. Appl. Math.*, 130(2):329–350, 2003. doi:10.1016/S0166-218X(02)00410-9.

12 Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In Peter van Beek, editor, *Proc. of 10th International Conference on Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, Berlin, Heidelberg, 2005. Springer. doi:10.1007/11564751_73.

13 Takehide Soh, Daniel Le Berre, Stéphanie Roussel, Mutsunori Banbara, and Naoyuki Tamura. Incremental SAT-based method with native Boolean cardinality handling for the Hamiltonian cycle problem. In Eduardo Fermé and João Leite, editors, *Proc. of 14th European Conference on Logics in Artificial Intelligence*, pages 684–693. Springer, 2014. doi:10.1007/978-3-319-11558-0_52.

14 Miroslav N. Velev and Ping Gao. Efficient SAT techniques for absolute encoding of permutation problems: Application to Hamiltonian cycles. In Vadim Bulitko and J. Christopher Beck, editors, *Proc. of 8th Symposium on Abstraction, Reformulation, and Approximation (SARA 2009)*. AAAI, 2009. URL: http://www.aaai.org/ocs/index.php/SARA/SARA09/paper/view/837.

15 Neng-Fa Zhou. In pursuit of an efficient SAT encoding for the Hamiltonian cycle problem. In Helmut Simonis, editor, *Proc. of 26th International Conference on Principles and Practice of Constraint Programming (CP 2020)*, volume 12333 of *Lecture Notes in Computer Science*, pages 585–602. Springer, 2020. doi:10.1007/978-3-030-58475-7_34.

16 Neng-Fa Zhou. Encoding the Hamiltonian cycle problem into SAT based on vertex elimination (short paper). In Paul Shaw, editor, *Proc. of 30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *LIPIcs*, pages 40:1–40:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CP.2024.40.