Solving QNP and FOND$^+$ with Generating, Testing and Forbidding

Zheyuan Shi, Hao Dong, Yongmei Liu*
Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
{shizhy5,dongh33}@mail2.sysu.edu.cn, ymliu@mail.sysu.edu.cn

Abstract

Qualitative Numerical Planning (QNP) extends classical planning with numerical variables that can be changed by arbitrary amounts. FOND$^+$ extends Fully Observable Non-Deterministic (FOND) planning by introducing explicit fairness assumptions, resulting in a more expressive model that can also capture QNP as a special case. However, existing QNP and FOND$^+$ solvers still face significant scalability challenges. To address this, we propose a novel framework for solving QNP and FOND$^+$ by **generating** strong cyclic solutions of the associated FOND problem, **testing** their validity, and **forbidding** non-solutions in conducting further searches. For this, we propose a procedure called SIEVE*, which generalizes the QNP termination testing algorithm SIEVE to determine whether a strong cyclic solution is a FOND$^+$ solution. Additionally, we propose several optimization techniques to further improve the performance of our basic framework. We implemented our approach based on the advanced FOND solver PRP; experimental results show that our solver shows superior scalability over the existing QNP and FOND$^+$ solvers.

1 Introduction

Fully Observable Non-Deterministic (FOND) planning extends classical planning to handle non-deterministic actions. In this context, strong-cyclic (SC) planning seeks to find a policy that ensures any reachable state has a trajectory leading to the goal. In contrast, strong planning finds policies that are guaranteed to reach the goal. It turns out that strong cyclic solutions are policies that are guaranteed to achieve the goal under the fairness assumption, which states that if an action occurs infinitely often, then any of its possible effects occurs infinitely often. Building on these, Qualitative Numerical Planning (QNP) [Srivastava *et al.*, 2011] further introduces non-deterministic numeric effects, while FOND$^+$ [Rodriguez *et al.*, 2022] generalizes SC, strong, and QNP planning by incorporating explicit fairness assumptions.

*Corresponding author

There are a number of existing solvers for QNP and FOND$^+$. Srivastava *et al.* [2011] proposed a generate-and-test method for solving a QNP problem, which translates the QNP problem into a FOND problem, solves it in the strong-cyclic setting and verifies termination using the SIEVE algorithm. However, there is no efficient implementation of this idea. Later, Bonet and Geffner [2020] proposed a method called qnp2fond, which compiles QNP into a more complex FOND problem and solves it with existing FOND solvers like PRP [Muise *et al.*, 2012]. Another recent QNP solver, DSET [Zeng *et al.*, 2022], introduced an algorithm that directly searches for a solution on the AND/OR graph induced by the QNP problem. FOND-ASP [Rodriguez *et al.*, 2022] provides a unified framework for solving QNP and FOND$^+$ problems by reducing them to Answer Set Programming (ASP) [Lifschitz, 2008], characterizing solutions as policies where all reachable states "terminate" under conditional fairness assumptions.

While existing solvers perform well on small problems, they often struggle with large-scale problems. For example, qnp2fond introduces a large number of additional actions and variables, leading to overly complex and hard-to-parse encodings. DSET lacks effective heuristics, resulting in poor scalability on large QNPs. As the only efficient FOND$^+$ solver currently, FOND-ASP frequently encounters timeouts or memory issues due to its need to encode all reachable states, leading to large and costly representations.

Motivated by the idea of building an efficient QNP and FOND$^+$ solver based on the advanced PRP solver which uses a key technique of forbidden state action pairs (fsaps), we propose a novel Generate-Test-and-Forbid (GTF) framework for solving QNP and FOND$^+$ problems. Our approach iteratively generates SC solutions by adapting PRP, tests their validity under explicit fairness assumptions, and conducts further searches by forbidding existing state-action assignments (referred to as steps). For this purpose, we propose a testing algorithm for FOND$^+$ called SIEVE* that generalizes SIEVE, and show that the solutions of a FOND$^+$ problem are the strong-cyclic solutions of the underlying FOND problem that pass the SIEVE* test. If a generated solution is invalid, the framework refines the search by forbidding specific steps, guiding the solver toward valid solutions.

To improve the efficiency of this process, we introduce several optimization techniques: 1) Pruning with key steps.

By identifying essential state-action assignments that must be preserved for any valid solution, we prune unnecessary branches in the search space. 2) Precomputing vital fsaps. We precompute some vital fsaps and forbid them in advance. 3) Heuristics for ordering forbidden steps. We leverage insights from SIEVE* and numeric planners to rearrange the order of forbidding steps, guiding the search toward branches that are more likely to contain valid solutions.

Based on our proposed approach, we implemented three variants of QNP and FOND$^+$ solvers. Through extensive experiments on existing and newly constructed problems, we demonstrate that our method shows superior performance over existing solvers on large-scale QNP problems and the majority of FOND$^+$ problems, and performs comparably to state-of-the-art solvers on small-scale QNP problems.

2 Preliminaries

2.1 FOND Planning

We follow the representation of fully observable non-deterministic (FOND) problems from [Muise *et al.*, 2012]. Let $\mathcal{V}$ be a finite set of variables, each with a finite domain D_v^+ that includes $\perp$ to denote undefined values. A **partial state** is a function mapping each $v \in \mathcal{V}$ to a value in D_v^+, while a **complete state** (or simply **state**) assigns every variable a defined value (i.e., no $\perp$). A partial state s **entails** s' (written $s \models s'$) if s agrees with s' on all variables defined in s'. The **update** $s \oplus s'$ applies s' to s, taking values from s' where defined, and from s otherwise.

Definition 1. A FOND planning problem is a tuple $\langle \mathcal{V}, s^0, s^*, \mathcal{A} \rangle$, where $\mathcal{V}$ is a finite set of variables, the initial state s^0 is a complete state, the goal s^* is a partial state, and $\mathcal{A}$ is a set of actions. Each action in $\mathcal{A}$ is made up of two parts: $\mathbf{Pre}_a$, a partial state that describes the condition under which a may be executed; and $\mathbf{Eff}_a$, a finite set of partial states that describe the possible outcomes of the action.

An action a is non-deterministic if $|\mathbf{Eff}_a| > 1$, and deterministic if $|\mathbf{Eff}_a| = 1$. It is applicable in a partial state s if $s \models \text{Pre}_a$. The progression of a partial state s w.r.t. an action a and effect $e \in \text{Eff}_a$, denoted $Prog(s, a, e)$, is the updated partial state $s \oplus e$ when a is applicable in s, and undefined otherwise. The set of successor states after executing action a in s is denoted by: $F(a, s) = \{Prog(s, a, e) \mid e \in \text{Eff}_a\}$.

Definition 2. A **policy** π for a FOND problem P is a partial function mapping non-goal states into applicable actions.

Given a policy π, a state trajectory $s_0, s_1, \ldots$ (finite or infinite) is called a **π-trajectory** if s_0 is the initial state, and for all $i \geq 0$, $s_{i+1} \in F(a_i, s_i)$ where $a_i = \pi(s_i)$. A π-trajectory is **maximal** if it is infinite, or it is finite and the last state s_n is the first goal state in the sequence or $\pi(s_n) = \perp$. A state s is **reachable** by π if there is a π-trajectory $s_0, \ldots, s$; and a state s' is **reachable** from s by π if there is a π-trajectory $s_0, \ldots, s, \ldots, s'$.

There are three types of solutions in the FOND setting [Cimatti *et al.*, 2003]:

Definition 3. *Let P be a FOND problem.*

1. A policy π is a **weak solution** of P if there exists a π-trajectory reaching the goal.
2. A policy π is a **strong solution** of P if every maximal π-trajectory is finite and goal-reaching.
3. A policy π is a **strong cyclic (SC) solution** of P if every state reachable by π can reach the goal.

In the rest of this section, we will use the Boolean version of FOND problems where all variables are Boolean.

2.2 QNP

Given a set of Boolean atoms At and a set of non-negative numerical variables V, we use $\mathcal{L}$ to denote the class of all consistent sets of literals of the form p and $\neg p$ for $p \in At$, and $X > 0$ and $X = 0$ for $X \in V$. A set of literals is said to be consistent if it does not contain both p and $\neg p$ for any Boolean atom p, and does not contain both $X > 0$ and $X = 0$ for any numerical variable X.

Definition 4. A qualitative numerical planning (QNP) problem Q is a tuple $\langle At, V, I, O, G \rangle$ where At is a set of Boolean atoms, V is a set of non-negative numerical variables, $I \in \mathcal{L}$ is the initial state, $G \in \mathcal{L}$ is the goal condition, and O is a set of actions. Every $o \in O$ has a set of preconditions $pre(o) \in \mathcal{L}$, and effects $eff(o)$. Boolean effects of $eff(o)$ contain literals of the form p and $\neg p$ for $p \in At$. Numerical effects of $eff(o)$ contain special atoms of the form $X\uparrow$ or $X\downarrow$ for $X \in V$ which increase or decrease X by an arbitrary amount. The literal $X > 0$ is a precondition of all actions with $X\downarrow$ effects.

A sound and complete two-step method for solving QNPs was formulated by Srivastava *et al.* [2011]: the QNP problem Q is converted into a standard FOND problem $P = T_D(Q)$ and its (strong-cyclic) solution is checked for termination. To get $T_D(Q)$, a Boolean atom p_X is introduced for each numerical variable X. Then literals $X > 0$ and $X = 0$ are replaced by literals p_X and $\neg p_X$, respectively. Effect $X\uparrow$ is replaced by effect p_X, and effect $X\downarrow$ is replaced by a non-deterministic choice of p_X and $\neg p_X$.

To check the termination of a policy π for $P = T_D(Q)$, Srivastava *et al.* [2011] proposed a sound and complete algorithm called SIEVE by repeatedly removing edges from the transition graph of π until the graph becomes acyclic or no additional edges can be removed. SIEVE identifies first the strongly connected components (SCCs) of the graph, then tries to remove an edge from an SCC. An edge can be removed from an SCC if there is a variable X s.t. the edge decrements X and no edge in the SCC increments X.

2.3 FOND$^+$

Definition 5. A FOND$^+$ problem $P_c = \langle P, C \rangle$ is a FOND problem P extended with a set C of (conditional) fairness assumptions of the form A_i/B_i, $i = 1, ..., n$, where each A_i is a set of non-deterministic actions in P, and each B_i is a set of actions in P disjoint from A_i.

Definition 6. A π-trajectory τ for a FOND$^+$ problem P_c is fair if the following holds: for any fairness assumption A_i/B_i in P_c, and for any action $a \in A_i$, if a occurs infinitely often on τ and actions from B_i occurs finitely often, then any of a's possible effects occurs infinitely often on τ.

Besides, we say an action a is **adversarial** if it is not in any A_i of any fairness assumption $\langle A_i/B_i\rangle$, and is **fair** if there is a fairness assumption $\langle A_i/\varnothing\rangle$ where $a \in A_i$. Here we use $\varnothing$ to represent the empty set.

Definition 7. A policy π solves a FOND$^+$ problem if all **maximal π-trajectories** that are **fair** reach the goal.

Rodriguez *et al.* [2022] showed that strong, strong cyclic and QNP planning can all be reduced to FOND$^+$ planning:

1. The **strong solutions** of a **FOND** problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \varnothing\rangle$.
2. The **strong-cyclic solutions** of a **FOND** problem P are the solutions of the FOND$^+$ problem $P_c = \langle P, \{A/\varnothing\}\rangle$, where A is the set of all non-deterministic actions in P.
3. The **solutions** of a **QNP** problem Q are the solutions of the FOND$^+$ problem $P_c = \langle P, C\rangle$ where $P = T_D(Q)$ and C is the set of fairness assumptions A_i/B_i, one for each numerical variable x_i in Q, such that A_i contains all the actions in Q that decrement x_i, and B_i contains all the actions in Q that increment x_i.

3 SIEVE* Testing for FOND$^+$

We propose a testing algorithm for FOND$^+$, called SIEVE*, which generalizes the QNP-specific SIEVE, and show that FOND$^+$ solutions are precisely the strong cyclic solutions of the underlying FOND problem that pass the SIEVE* test.

The main idea of SIEVE* is as follows: Like SIEVE, given a policy, SIEVE* repeatedly removes edges from the transition graph of the policy until the graph becomes acyclic or no more edges can be removed, and the condition for removing an edge is a generalization of that for SIEVE. As shown in Alg. 1, SIEVE* takes as input a graph g and a set C of conditional fairness assumptions. SIEVE* first checks for the strong connectivity of g, and if it is not strongly connected, we compute the SCCs of g following Tarjan's algorithm [Tarjan, 1972] (line 2), and run SIEVE* on these sub-graphs recursively; if g is strongly connected, then we repeatedly remove edges from g when conditions 1) and 2) are satisfied (line 7). If g is acyclic, it is considered to pass the test (line 10). If no edge is removed, the graph g is said to fail the test (line 12). Otherwise, SIEVE* is recursively applied to the graph after edge removals (line 13).

We now illustrate the algorithm with an example.

Example 1. Consider a FOND$^+$ problem with a single fairness assumption $A/\varnothing$, where $A = \{a_j, a_k\}$. Fig. 1 illustrates two state transition graphs g_{left} and g_{right} for different scenarios of a policy π, with only three states s_i, s_j, and s_k and the associated transitions displayed. Let $lr = left$ or $right$.

SIEVE*(g_{lr}) identifies the SCC (line 2), denoted as $g_{\text{lr}}^{\text{ijk}}$, containing nodes s_i, s_j, and s_k. It then removes the edge (s_j, s_i), as $\pi(s_j)$ is in A and $F(\pi(s_j), s_j)$ contains nodes outside $g_{\text{lr}}^{\text{ijk}}$ (line 7). Despite this removal, $g_{\text{lr}}^{\text{ijk}}$ remains cyclic, leading the algorithm to call SIEVE* again (line 13). At this point, the SCC obtained by SCCs-of($g_{\text{lr}}^{\text{ijk}}$) in SIEVE*($g_{\text{lr}}^{\text{ijk}}$) only contains nodes s_i and s_k, denoted as $g_{\text{lr}}^{\text{ik}}$. However, in the left case, no edge satisfies both conditions 1) and 2),

Algorithm 1: SIEVE*

Input: Graph g, fairness assumptions C
Output: "True" for passing, "False" otherwise

1 **if** *g is not strongly connected* **then**
2 **foreach** *$g' \in$ SCCs-of (g)* **do**
3 **if** *SIEVE*(g',C) = "False"* **then**
4 **return** *"False"*;
5 **return** *"True"*;
6 **repeat**
7 remove an edge (s, s') in g if **1)** there exists $A_i/B_i \in C$ such that $\pi(s) \in A_i$ and no action in B_i appears in g and **2)** at least one state in $F(\pi(s), s)$ is not in g.
8 **until** *no edge can be removed*;
9 **if** *g is acyclic* **then**
10 **return** *"True"*;
11 **if** *no edge was removed* **then**
12 **return** *"False"*;
13 **return** *SIEVE*(g,C)*;

so SIEVE*($g_{\text{left}}^{\text{ik}}$) returns False, causing SIEVE*($g_{\text{left}}^{\text{ijk}}$) then SIEVE*($g_{\text{left}}$) to return False (line 4). In contrast, in the right case, $F(\pi(s_k), s_k)$ contains s_j not in $g_{\text{right}}^{\text{ik}}$, allowing edge (s_k, s_i) to be removed, thus converting $g_{\text{right}}^{\text{ik}}$ into acyclic, and SIEVE*($g_{\text{right}}^{\text{ik}}$) returns True. So SIEVE*($g_{\text{right}}^{\text{ijk}}$) returns True (line 5) and we need to check other SCCs in SCCs-of(g_{right}). Actually, in the left graph, there is a fair π-trajectory $(s_0, \ldots, s_i, s_k, s_k, s_i, \ldots)$ that loops infinitely in $g_{\text{left}}^{\text{ik}}$, indicating π is not a FOND$^+$ solution according to Def. 7.

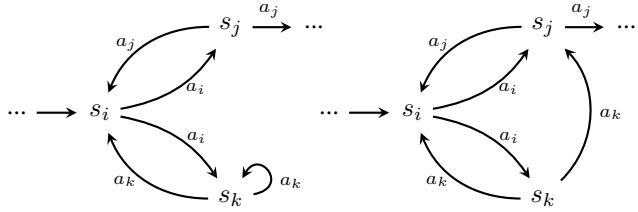

Figure 1: Two state transition graphs g_{left} and g_{right}, where s_i, s_j, and s_k are non-goal states, and a_i, a_j, and a_k are non-deterministic actions. The initial states s_0 are in the "..." leading to s_i.

We have the following nice characterization of FOND$^+$ solutions. We use $\mathcal{G}(\pi)$ to denote the **transition graph** of π.

Theorem 1. *A policy π is a solution to a FOND$^+$ problem $P_c = \langle P, C\rangle$ iff π is a strong cyclic solution of P and **SIEVE***$(\mathcal{G}(\pi), C)$ *returns "True".*

Sketch. $\Rightarrow$: Assume π is not a strong cyclic solution of P. We use π-deadend to refer to a state s that is reachable by π but cannot reach the goal. Then there must exist an SCC where all states are π-deadends. We can use this SCC to construct a fair maximal π-trajectory that is not goal-reaching. So π is a strong cyclic solution of P. Assume π fails to pass SIEVE*.

Then there must exist an SCC g_{sc} of $\mathcal{G}(\pi)$ s.t. SIEVE*(g_{sc}) returns "False". Again, we can use this SCC to construct a fair maximal π-trajectory that is not goal-reaching.

$\Leftarrow$: Assume π is not a solution of P_c. Then there exists a fair maximal π-trajectory τ that cannot reach the goal. From τ, we can get an SCC of $\mathcal{G}(\pi)$ where no edge can be removed by SIEVE*. Thus π fails to pass the SIEVE* test. □

Note when a sub-graph of $\mathcal{G}(\pi)$ fails the SIEVE* test, $\mathcal{G}(\pi)$ will also fail the test (as the left case in Fig. 1 shows). This makes it possible for early detection of failure of a partial policy and serves as a pruning mechanism during the search for a solution, which will be introduced in Section 5.1.

Also note that SIEVE is a special case of SIEVE*. By the reduction of QNP to FOND$^+$ at the end of Section 2.3, the edge removal condition used in the SIEVE algorithm corresponds to condition **1)** in SIEVE*. Condition **2)** always holds in the QNP setting, which can be shown by contradiction, though we will not prove it here.

4 The Generate-Test-and-Forbid Framework

To solve FOND$^+$ problems, we aim to generate an SC solution that passes SIEVE*. In this section, we propose the PRP-based generate-test-and-forbid (GTF) framework as the basic framework for solving FOND$^+$, and prove its soundness and completeness.

To solve FOND problems, PRP builds an SC solution by repeatedly repairing a weak solution generated by a classical planner. A key technique used by PRP is the recording of deadends as forbidden **partial** state-action pairs (fsaps) to avoid them in all subsequent searches.

In this work, we adapt PRP as the underlying strong cyclic generator and further use fsaps for two purposes: (1) to prevent the generation of SC solutions that use vital fsaps (which will be identified in Section 5.2), and (2) to generate all SC solutions by incrementally forbidding **complete** state-action pairs, which we refer to as steps.

Definition 8. A **state**-action pair $\langle s, a\rangle$ is a **step** if $s \models \text{Pre}_a$. Given a policy π, its **step set** $\Psi(\pi)$ is the set of state-action pairs $\langle s, \pi(s)\rangle$ where s is reachable by π and $\pi(s) \neq \bot$.

Given a policy π, for any π', if there exists $\langle s, a\rangle \in \Psi(\pi)$ such that $\langle s, a\rangle \notin \Psi(\pi')$, then π' is different from π. Thus, by searching without using the steps in $\Psi(\pi)$, we can obtain solutions distinct from π. We refer to the steps that are not allowed to be used as Forbidden Steps (FSteps). A strong cyclic solution π **forbids** step set $fsteps$ if $\Psi(\pi)$ does not include any step from $fsteps$.

Definition 9. Given a FOND problem P and a step set $fsteps$, we use $FSC_P(fsteps)$ to denote the set of all strong cyclic solutions of P that forbid $fsteps$: $FSC_P(fsteps) = \{\pi \mid \pi \text{ is an SC solution of } P, \Psi(\pi) \cap fsteps = \varnothing\}$.

Clearly, $FSC_P(\varnothing)$ is the set of strong cyclic solutions of P. The following proposition states that to enumerate all SC solutions forbidding a step set $fsteps$, it suffices to find one such solution π, and then recursively enumerate all SC solutions obtained by additionally forbidding each step in $\Psi(\pi)$:

Algorithm 2: FOND$^+$ Solver. Entry Program

Input: FOND$^+$ problem $P_c = \langle P, C\rangle$
Output: A FOND$^+$ solution π or "None"

```
1  F ← ∅;
2  ⋆K ← ∅;
3  ⋆PreComputeVFsaps(F);
4  return GTF(P_c, F, ⋆K);
```

Algorithm 3: FOND$^+$ Solver. GTF

Input: FOND$^+$ problem $P_c = \langle P, C\rangle$,
fsaps F,
⋆key steps K
Output: A FOND$^+$ solution forbidding F or "None"

```
1  π ← FindSC(P,F);
2  if there is no SC solution then
3      return "None";
4  if SIEVE*(G(π),C) = "True" then
5      return π;
6  ⋆Rearrange(Ψ(π));
7  foreach step ∈ Ψ(π) do
8      ⋆if step ∈K then
9          continue
10     if π' ← GTF(P_c, F ∪ {step}, ⋆K) then
11         return π';
12     ⋆K.add(step);
13     ⋆if SIEVE*(G(K),C) = "False" then
14         return "None";
15     ⋆F.AddBlockingFSteps(step);
16 return "None";
```

Proposition 1. For a FOND problem P, if $\pi \in FSC_P(fsteps)$, then we have $FSC_P(fsteps) = \{\pi\} \cup \bigcup_{\langle s,a\rangle \in \Psi(\pi)} FSC_P(fsteps \cup \{\langle s, a\rangle\})$.

The solving framework consists of two parts: the entry program for initializing the solving process (Alg. 2) and the generate-test-and-forbid process GTF (Alg. 3). In this section, we do not consider the lines and variables marked with ⋆, which concern optimization techniques and will be discussed in the next section. Given a set of FSteps F, the process GTF(P_c, F) returns a FOND$^+$ solution in $FSC_P(F)$. Thus, by initializing F to an empty set, the entire framework computes a FOND$^+$ solution within the entire space of strong cyclic solutions.

Algorithm 3 illustrates how GTF works. Given P_c and F, we first generate an SC solution in $FSC_P(F)$ (line 1) and test with SIEVE* (line 4). For generating, we amend the sound and complete FOND solver PRP to generate an SC solution forbidding F, which is actually searching for the "π" in Prop. 1 with $fsteps = F$. When π fails SIEVE*, we forbid each $step$ in $\Psi(\pi)$ and search in $FSC_P(F \cup \{step\})$ (line 10). When all $steps$ have been forbidden and no solution is found, GTF returns "None" in line 16.

We now present the soundness and completeness results.

Theorem 2. *For a given FOND$^+$ problem $P_c = \langle P, C\rangle$ and a set of steps F, if GTF(P_c, F) returns π, then π is a solution of P_c forbidding F; and if GTF(P_c, F) returns "None", there is no solution of P_c forbidding F.*

Proof. This follows from Theorem 1, Proposition 1, and the soundness and completeness of PRP. □

Corollary 1. *For a given FOND$^+$ problem $P_c = \langle P, C\rangle$, if GTF$(P_c, \varnothing)$ returns π, then π solves P_c; and if GTF$(P_c, \varnothing)$ returns "None", then P_c is unsolvable.*

Note this framework is not limited to generating a single FOND$^+$ solution. By replacing "return π" in line 5 with "store π", we can collect all valid solutions.

5 Optimization Techniques

In this section, we focus on the $\star$-marked lines in the algorithm which are all designed for accelerating the FOND$^+$ solving process. The soundness remains as these techniques don't modify the conditions of the algorithm to return a policy, i.e., a strong cyclic solution that passes SIEVE*; and we are going to show the completeness is also not affected.

5.1 Prune with Key Steps

In the basic version of GTF, we do nothing when forbidding a step leads to no solution in line 10. Indeed, we have the following observation, which can be proved by contradiction:

Proposition 2. *If GTF$(P_c, F \cup \{step\})$ = "None", then for any $F' \supseteq F$, π = GTF(P_c, F') implies $step \in \Psi(\pi)$.*

We refer to such a step as a **key** step (w.r.t. F), which means that this step is required for any potential FOND$^+$ solution within $FSC_P(F)$. We use a set K to store these key steps (line 12), and pass them to subsequent calls of GTF (line 10). The third input K of the GTF procedure means that K is a set of key steps that must be included in any solution returned by GTF.

We develop two pruning techniques. The first is called "Skip Key" (line 8-9). When we find that the step to be forbidden has been identified as a key step, we skip the current iteration and proceed to the next iteration of the loop. The second is called "Test Keys" (line 13-14). As $K \subseteq \Psi(\pi)$ for any potential solution π, if $\mathcal{G}(K)$ fails SIEVE*, π also fails SIEVE* (according to Theorem 1).

Additionally, we add some extra steps that we call BlockingFSteps to F to prevent other applicable actions to be picked in the state of a key step (line 15) . Specifically, for a key step $\langle s, a\rangle$, we let $F \leftarrow F \cup \{\langle s, a'\rangle \mid s \models \text{Pre}_{a'} \wedge a' \neq a\}$.

Completeness reserves with the above techniques due to Proposition 2.

5.2 Precompute Vital Fsaps

To enhance FOND$^+$ solving, we further identify certain partial state–action pairs that lead to invalid solutions, referred to as Vital Fsaps (VFsaps), and forbid them during the search process. Specifically, a pair $\langle s, a\rangle$ is considered a vital fsap if, for any state s' entailing s, executing action a causes the SIEVE* test to fail.

In this work, we precompute a subset of VFsaps that induce one-node SCCs where no edge can be removed: $\{\langle s, a\rangle \mid a$ is adversarial, there is $e \in \text{Eff}_a$ $s.t.$ $\text{Prog}(s, a, e) = s\}$. These steps are precomputed and incorporated into the global fsaps F (line 3 in the entry program) to exclude invalid policies while ensuring completeness is preserved.

5.3 Rearrange the Order for Forbidding Steps

Note that the soundness and completeness are not affected by the order of forbidding steps. In this part, we propose different strategies to rearrange (line 6) the order of forbidding steps to improve performance.

Breadth-First Forbidding (BFF) means to rearrange the steps $\Psi(\pi)$ according to the order in which the states are visited in a breadth-first search of $\mathcal{G}(\pi)$. It can be proved that with the technique "AddBlockingFSteps" proposed in Section 5.1, we guarantee when an SC solution is generated by $FindSC$ in line 1, it is different from all other SC solutions generated before. However, it can still be inefficient to find a solution with this strategy due to the large search space of SC solutions. The following forbidding strategy enables the integration of heuristics:

Futile-First Forbidding (3F): We refer to the steps that might be harmful (e.g., by leading to a fair trajectory not reaching the goal) or useless (e.g., redundant for achieving the goal) as "futile steps". Futile-First Forbidding sorts steps in a specific order based on certain heuristics, guiding GTF to forbid futile steps first, aiming to find a solution as quickly as possible.

In this work, we introduce two kinds of heuristics for Futile-First Forbidding. The first, **3FF**, is obtained from the **F**ailure result of SIEVE* testing for a policy, i.e., we forbid the steps in an SCC whose edges cannot be removed. The second, **3FN**, involves translating the QNP problem into a **N**umerical planning problem. Actions unused in the resulting solution are considered redundant and are prioritized for forbidding. Specifically, in this work, we transform the problem by fixing all amounts of numerical changes to 1 and replacing any initial non-zero numerical values with 1, then use Metric-FF [Hoffmann, 2003] to compute such a solution.

6 Implementation and Experiments

We implemented three solvers in C++ using the above three rearrangement strategies: GTF-BFF (FOND$^+$ solver), GTF-3FF (FOND$^+$ solver) and GTF-3FN (QNP solver).[1]

Note that in this section, when we mention FOND$^+$ domains or problems, we specifically refer to those that cannot be represented as QNPs. We evaluate the performance of our three solvers on QNPs in comparison with DSET [Zeng *et al.*, 2022], FOND-ASP [Rodriguez *et al.*, 2022], and three solvers using qnp2fond translator [Bonet and Geffner, 2020], each paired with different underlying FOND solvers for SC planning: PRP [Muise *et al.*, 2012], (FOND-)SAT [Geffner and Geffner, 2018], and PR2 [Muise *et al.*, 2024]. For FOND$^+$ planning, we compare GTF-BFF and GTF-3FF against FOND-ASP. All experiments were run on an Ubuntu 20.04 Linux machine with an Intel Core i9-10980XE CPU

[1]Codes and data – https://github.com/sysulic/GTF4FONDX.

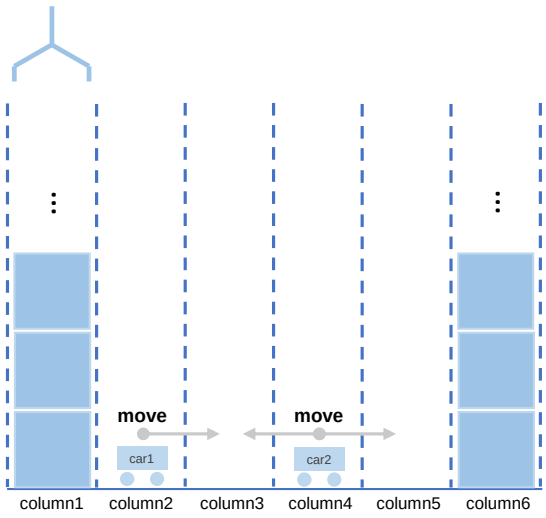

Figure 2: An example of the FOND$^+$ versions of the BlocksColumns domain. Moving car 2 may resulting it moving to column 3 or column 5, while moving car 1 will move it to column 3.

(3.00 GHz). Each instance was allocated a maximum of 8 GB of memory and a runtime limit of 30 minutes.

6.1 Domains

The QNP instances with small numbers of actions, features, and reachable states are categorized as Tiny-domains, including: $blocks\_clear$, $blocks\_on$, $gripper$, $delivery$, $delivery2$, $q1$, $q2$ (unsolvable), $q3$, $gripper2$ and $rewards$ from Bonet and Geffner [2020]; $Gripper1u$ (unsolvable) and $Nest3u$ (unsolvable) from Zeng *et al.* [2022]; and 9 instances in $qnp1$ from Rodriguez *et al.* [2022]. Other existing QNP domains (each including one or more instances) include: $Nests$ and $Nests\_u$ (all instances are unsolvable) from Zeng *et al.* [2022]; $qnp2$ from Rodriguez *et al.* [2022]; $GripperAbs$, $FerryAbs$, $LogisticsAbs$, $ZenotravelAbs$, $NomysteryAbs$, and $FloortileAbs$ from Dong *et al.* [2025]. Note that domains from [Dong *et al.*, 2025] are actually bounded QNP problems, where the change in any numerical variable caused by an action is exactly one. However, we found that some of them can still be solved by certain QNP solvers. Existing FOND$^+$ domains are all from Rodriguez *et al.* [2022]: $qnp2\text{-}f11$, $qnp2\text{-}f01$ (unsolvable), $football$ and $football\_u$ (unsolvable).

We propose two new QNP domains and further extend them to FOND$^+$ versions.

BlocksColumns This domain consists of several columns, each containing some blocks, with robot cars initially positioned in the columns except for the rightmost one. The objective is to move these cars to the rightmost column. Additionally, there are grippers that can move between columns and are capable of picking up and dropping blocks in a column, provided no car is present in that column. The cars can also move to an adjacent column if that column contains no blocks. A column that contains no blocks is considered to have sufficient space to hold multiple cars simultaneously. In this domain, numerical variables are used to represent the number of blocks in each column, which increase when blocks are dropped and decrease when blocks are picked up. One solution is to move the blocks from the columns to the right of the cars into the columns to their left, thereby clearing a path for the cars. Once the path is clear, the cars can be moved to the rightmost column.

By making the effect of car-moving non-determinisitic, we construct two FOND$^+$ domains: **BlocksColumns-Fair (BC-Fair)** and **BlocksColumns-Adv (BC-Adv)**, as shown in Fig. 2. In both domains, the objective and the possible actions of grippers are the same as the domain BlocksColumns. The only difference is that car movements are non-deterministic: when a car is moved, it may go left or right. However, if the adjacent column in one direction contains blocks, that direction is blocked, and the car will move to the other direction instead. The difference between the two domains is that in BC-Fair, the move action is assumed to be fair, while in BC-Adv there is no such fairness assumption.

HallsFloors This domain is adapted from the Halls domain [Bonet *et al.*, 2009; Lin *et al.*, 2022]. The environment consists of multiple floors and a single agent. Each floor contains an uncertain number of interconnected halls arranged in a rectangular ring. Some floors are connected via staircases, which are located at the four corners of the rectangle. Four numeric variables represent the agent's distance to the four sides of the rectangular floor layout. The agent can move between halls and is considered to have reached a corner when two of these distance variables are simultaneously zero. A floor is considered visited once the agent has visited all four of its corners. The goal is to visit all floors. In the QNP version, the positions of the staircases are fixed initially, and the agent can use a staircase if it is located at the same corner. In contrast, in the FOND$^+$ version, called **HF-ND**, the staircases are hidden and must be revealed by executing the "discover-stair" action after the agent has visited the floor. This action is non-deterministic and causes a staircase to appear at one of the four corners.

6.2 Results

The information about the size of these domains and overall solving results are shown in Tables 1 (QNP) and 2 (FOND$^+$). A bold line separates the existing domains from the new ones. For each domain and solver, we report the coverage of the solver, that is, the ratio of instances solved by the solver. For tiny domains, we also report the average solving time in the parentheses when the solver can solve all instances. Figure 3 illustrates the overall coverage performance (the ratio of all instances solved) of various solvers over time for QNP/FOND$^+$.

As shown in Fig. 3, GTF-3FN achieves the best performance in QNP planning, consistently solving the highest proportion of instances across all time limits, with GTF-3FF following closely. GTF-BFF and DSET also perform well but fall behind the top two methods. In contrast, FOND-ASP and solvers using qnp2fond fail to solve even half of the QNP instances. For FOND$^+$ planning, where GTF-3FN is not applicable, GTF-3FF outperforms all other methods, achieving

Domains	Size (Average) |F|	|O|	#RStates	**GTF (ours)** BFF	3FF	3FN	FOND ASP	DSET	qnp2fond PRP	PR2	SAT
Tiny-domains(21)	5.0	5.5	9.8	**1(0.09s)**	**1(0.09s)**	1(0.10s)	1(0.43s)	**1(0.09s)**	1(35.93s)	0.86	0.86
Nests(7)	7.0	7.0	290.3	**1.00**	**1.00**	**1.00**	0.00	**1.00**	0.00	0.00	0.00
Nests_u(7)	7.0	7.0	146.1	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00
qnp2(9)	7.0	7.0	228.1	**1.00**	**1.00**	**1.00**	0.89	**1.00**	0.89	0.67	0.44
GripperAbs(8)	10.0	11.4	3102.6	0.50	0.88	**1.00**	0.50	0.88	0.88	0.88	0.25
FerryAbs(2)	10.0	24.0	577.0	0.50	**1.00**	**1.00**	0.50	0.50	**1.00**	0.50	0.50
LogisticsAbs(1)	22.0	36.0	7e+04	**1.00**	**1.00**	**1.00**	0.00	0.00	**1.00**	**1.00**	0.00
ZenotravelAbs(20)	160.2	6164.1	≥1e+07	**1.00**	**1.00**	**1.00**	0.00	0.05	0.35	0.35	0.10
NomysteryAbs(20)	192.4	3572.9	≥5e+06	**0.70**	**0.70**	**0.70**	0.00	0.60	0.05	0.05	0.00
BlocksColumns(30)	35.0	60.0	≥4e+05	0.23	0.83	**0.93**	0.03	0.27	0.03	0.00	0.03
HallsFloors(12)	26.9	32.8	≥2e+07	**1.00**	**1.00**	**1.00**	0.08	0.58	0.08	0.08	0.08
Overall(137)	64.3	1440.8	≥4e+06	0.75	0.91	**0.94**	0.31	0.58	0.36	0.31	0.21

Table 1: Performance comparison of various solvers across QNP domains. For each domain, the number of instances, average sizes (features |F|, actions |O|, and reachable states #RStates), and the coverage performance of different solvers are presented.

Domains	Size (Average) |F|	|O|	#RStates	**GTF (ours)** BFF	3FF	FOND ASP
qnp2-f11(9)	9.0	9.0	910.4	**1.00**	**1.00**	0.67
qnp2-f01(9)	7.0	7.0	228.1	**1.00**	**1.00**	0.89
football_u(10)	61.0	196.5	2860.0	0.10	0.00	**0.30**
football(10)	61.0	196.5	2860.0	**1.00**	**1.00**	0.30
BC-Fair(30)	45.0	51.3	1e+06	0.23	**1.00**	0.03
BC-Adv(30)	45.0	51.3	1e+06	0.00	**0.17**	0.03
HF-ND(12)	49.7	55.5	≥7e+06	**1.00**	**1.00**	0.17
Overall(110)	42.4	71.1	≥2e+06	0.44	**0.68**	0.22

Table 2: Coverage Performance across FOND$^+$ domains.

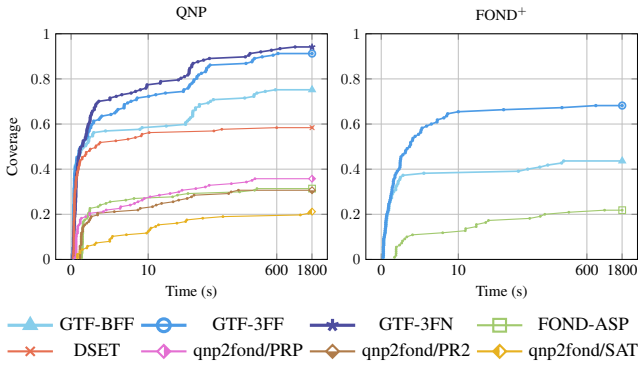

Figure 3: Overall Coverage over time for QNP/FOND$^+$.

a coverage of 68% (as shown in Table 2). Meanwhile, GTF-BFF solves fewer than half of the instances, and FOND-ASP lags further behind with a final coverage of only 22%.

The superior performance of our method stems from several factors: (1) building on top of an advanced FOND solver, which enables the use of existing heuristics from SC and classical planning; (2) the low memory cost of the generate-and-test method compared to FOND-ASP; and (3) the effectiveness of the proposed heuristics, derived from failure analysis in SIEVE* and solutions provided by a numerical planner in the QNP setting.

We also found that our method struggles with certain problems, as shown in Tables 1 and 2, highlighting potential areas for future improvement. For the QNP domain $NomysteryAbs$, our underlying solver encounters difficulty finding a strong-cyclic (SC) solution when the problem size becomes too large. Additionally, in solvable FOND$^+$ domains, GTF-3FF still requires excessive SC search efforts for $BC\text{-}Adv$, suggesting the need for more effective strategies to reorder or prioritize search. Furthermore, the failure in the unsolvable problem $football\_u$ demonstrates cases where our method performs poorly: there are exponentially many SC solutions, yet none of them solves the FOND$^+$ problem.

7 Conclusion

In this paper, based on the advanced FOND solver PRP, we propose a generate-test-and-forbid framework for solving QNP and FOND$^+$ problems. Moreover, we propose several optimization techniques such as pruning with key steps, precomputing vital fsaps, and heuristics for ordering forbidden steps. Based on our proposed approach, we implemented three variants of QNP and FOND$^+$ solvers. Experimental results on existing and newly constructed domains demonstrate that our method shows superior scalability over existing solvers. In the future, we are interested in exploring further heuristics to improve the performance of our solvers, and efficient solving techniques for QNP or FOND$^+$ problems with large space of strong cyclic policies but few solutions.

Acknowledgments

We thank the annonymous reviewers for helpful comments. We acknowledge support from the Natural Science Foundation of China under Grant No. 62076261.

References

[Bonet and Geffner, 2020] Blai Bonet and Hector Geffner. Qualitative numeric planning: Reductions and complexity. *J. Artif. Intell. Res.*, 69:923–961, 2020.

[Bonet *et al.*, 2009] Blai Bonet, Héctor Palacios, and Hector Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS-09*, 2009.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[Dong *et al.*, 2025] Hao Dong, Zheyuan Shi, Hemeng Zeng, and Yongmei Liu. An automatic sound and complete abstraction method for generalized planning with baggable types. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence, AAAI-25*, 2025.

[Geffner and Geffner, 2018] Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling, ICAPS-18*, 2018.

[Hoffmann, 2003] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res.*, 20:291–341, 2003.

[Lifschitz, 2008] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence, AAAI-08*, 2008.

[Lin *et al.*, 2022] Xiaoyou Lin, Qingliang Chen, Liangda Fang, Quanlong Guan, Weiqi Luo, and Kaile Su. Generalized linear integer numeric planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling, ICAPS-22*, 2022.

[Muise *et al.*, 2012] Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling, ICAPS-12*, 2012.

[Muise *et al.*, 2024] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. PRP rebooted: Advancing the state of the art in FOND planning. In *Proceedings of the 38th Annual AAAI Conference on Artificial Intelligence, AAAI-24*, 2024.

[Rodriguez *et al.*, 2022] Ivan D. Rodriguez, Blai Bonet, Sebastian Sardiña, and Hector Geffner. FOND planning with explicit fairness assumptions. *J. Artif. Intell. Res.*, 74:887–916, 2022.

[Srivastava *et al.*, 2011] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence, AAAI-11*, 2011.

[Tarjan, 1972] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[Zeng *et al.*, 2022] Hemeng Zeng, Yikun Liang, and Yongmei Liu. A native qualitative numeric planning solver based on AND/OR graph search. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-22*, 2022.