A Native Qualitative Numeric Planning Solver Based on AND/OR Graph Search

Hemeng Zeng, **Yikun Liang** and **Yongmei Liu**[*]
Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
ymliu@mail.sysu.edu.cn

Abstract

Qualitative numeric planning (QNP) is classical planning extended with non-negative real variables that can be increased or decreased by some arbitrary amount. Existing approaches for solving QNP problems are exclusively based on compilation to fully observable nondeterministic planning (FOND) problems or FOND$^+$ problems, i.e., FOND problems with explicit fairness assumptions. However, the FOND-compilation approaches suffer from some limitations, such as difficulties to generate all strong cyclic solutions for FOND problems or introducing a great many extra variables and actions. In this paper, we propose a simpler characterization of QNP solutions and a new approach to solve QNP problems based on directly searching for a solution, which is a closed and terminating subgraph that contains a goal node, in the AND/OR graphs induced by QNP problems. Moreover, we introduce a pruning strategy based on termination tests on subgraphs. We implemented a native solver DSET based on the proposed approach and compared the performance of it with that of the two compilation-based approaches. Experimental results show that DSET is faster than the FOND-compilation approach by one order of magnitude, and comparable with the FOND$^+$-compilation approach.

1 Introduction

Qualitative numerical planning (QNP) is classical planning extended with non-negative real variables and non-deterministic actions that increase or decrease the values of these variables by some arbitrary amount. QNP was firstly introduced by Srivastava et al. [2011] to model the problems of planning with loops, where actions may have to be repeated unknown times to reach the goal.

Recently, QNP has been widely used for generalized planning because it is decidable and provides a useful abstraction model for generalized planning. Generalized planning studies the computation of solutions that generalize over multiple planning instances [Levesque, 2005]. Some works [Bonet and Geffner, 2018; Bonet *et al.*, 2019] learn potential features (numerical and boolean) and abstract actions from solutions of sample instances, and then abstract the generalized planning problems into QNP problems. Illanes and McIlraith [2019] consider a class of generalized planning problems called quantified planning problems, and abstract such problems into QNP problems based on the idea of quantifying over sets of similar objects.

Existing works for solving QNP problems are exclusively based on compilation to fully observable non-deterministic planning (FOND) problems. First of all, Srivastava et al. [2011] propose a generate-and-test approach. They show that a QNP problem Q can be easily compiled into a FOND problem P whose strong cyclic solutions that terminate are solutions to Q. They introduce the SIEVE algorithm for testing whether a solution terminates. However, the generate-and-test method has two limitations: firstly, it is difficult to compute all solutions for P by using off-the-shelf FOND planners; secondly, the generate is done independently of the test, with the result that more solutions than necessary are generated. To save from the termination test, Bonet and Geffner [2020] propose a translator **`qnp2fond`** to compile a QNP problem Q into a FOND problem P' with extra variables and actions. They show that if Q is solvable, then a solution π' solves P'. However, it remains open whether the flat solution for π', i.e., the solution obtained from π' by ignoring the extra variables and actions, is a solution to the original problem Q. Moreover, introducing extra variables and actions increases the search space exponentially.

Very recently, Rodriguez et al. [2021] propose the so-called FOND$^+$ planning, that is, FOND planning with explicit fairness assumptions. They show that QNP is a special class of FOND$^+$ planning, where the fairness assumption is that for any numerical variable x, if the value of x is increased infinitely often but decreased only finitely often, then eventually x becomes 0. They develop a FOND$^+$ solver FOND-ASP by reducing FOND$^+$ planning to answer set programming (ASP) [Brewka *et al.*, 2011].

AND/OR graphs have been widely used in planning because the state spaces of many planning problems can be formalized as AND/OR graphs which can be traversed by off-the-shelf heuristic search algorithms, such as AO* [Nilsson, 1982] and LAO* [Hansen and Zilberstein, 2001]. To solve

[*]Corresponding Author

a FOND problem P, Bercher and Mattmüller [2009] introduce an approach that traverses the AND/OR graph induced by P using AO* to compute a strong solution. Mattmüller et al. [2010] find strong cyclic solutions for FOND problems by performing LAO* search.

In this paper, we propose a new approach to solve a QNP problem Q based on directly searching for a solution in the AND/OR graph induced by Q. Our approach is based on a simpler characterization of QNP solutions and a key observation about termination of policies. Srivastava et al. [2011] characterize a solution to a QNP problem as a strong cyclic solution that terminates. We give a simpler characterization that a solution to a QNP problem is a closed and terminating policy with a reachable goal node. We make the key observation that any extension of a non-terminating policy is also non-terminating. Thus, we search over all policies by a backtracking algorithm and on identifying a non-terminating one, we prune all its extensions.

Based on the proposed approach, we implemented a QNP solver DSET, and compared the performance of DSET with that of the compilation-based approach of Bonet and Geffner [2020], using two typical FOND solvers PRP [Muise *et al.*, 2012] and FOND-SAT [Geffner and Geffner, 2018]. Experimental results show that on solvable domains, DSET is faster than the compilation-based solvers by one order of magnitude; and on unsolvable domains, DSET takes a few seconds, while the other solvers time out after 30 minutes. The main reason behind the performance gap is that the compilation to FOND problems introduces many extra variables and actions so that the search space for solutions is exponentially increased. We also compared the performance of DSET with that of FOND-ASP. It turns out that the two solvers have comparable performance on most domains since they consider the same search space for QNP solutions.

2 Preliminaries

In this section, we introduce QNP problems, AND/OR graphs, and the existing characterization of QNP solutions.

Given a set of propositional variables F and a set of non-negative numerical variables V, we refer to p and $\neg p$ for $p \in F$ as the F-literals, and refer to $v > 0$ and $v = 0$ for $v \in V$ as the V-literals. A pair of complementary literals has the form $\{p, \neg p\}$ for $p \in F$, or $\{v > 0, v = 0\}$ for $v \in V$. A set of literals is consistent if it does not contain complementary literals. Let $\mathcal{L}_X$ (resp. $\mathcal{L}_F$) denote the set of all consistent sets of literals from $F \cup V$ (resp. F).

Definition 1. A QNP problem is a tuple $Q = \langle F, V, I, G, O\rangle$ where F and V are sets of propositional and non-negative numerical variables, $I \in \mathcal{L}_X$ denotes the initial condition, $G \in \mathcal{L}_X$ denotes the goal condition, O is a set of actions. Every $a \in O$ has a set of preconditions $pre(a) \in \mathcal{L}_X$, propositional effects $eff(a) \in \mathcal{L}_F$ and numerical effects $N(a)$ which only contain special atoms of the form $inc(v)$ or $dec(v)$ (often abbreviated as $v\uparrow$ or $v\downarrow$) to increase or decrease v by some arbitrary amount for $v \in V$. Actions with the $dec(v)$ effect must feature the precondition $v > 0$ for any variable $v \in V$.

We distinguish between states and qualitative states (qstates): a state $\bar{s}$ is an assignment of values to all variables, a qstate s is an element of $\mathcal{L}_X$ that contains a literal of each variable. $\bar{s}$ satisfies s if the assignment of values for variables in $\bar{s}$ satisfies all literals in s. Usually, given a QNP problem $Q = \langle F, V, I, G, O\rangle$, I is an initial qstate and G is a partially specified qstate corresponding to a set of goal qstates.

Definition 2. Given a QNP problem $Q = \langle F, V, I, G, O\rangle$, an instance of Q is a quantitative planning problem whose initial state, which specifies a non-negative real number for each numerical variable, satisfies I, and the actions and goal condition are the same as Q.

A simple QNP problem Q_{nest_2} is shown as follows where $\langle C; E\rangle$ denotes an action with preconditions C and effects E.

Example 1. $Q_{nest_2} = \langle F, V, I, G, O\rangle$ with $F = \emptyset$, $V = \{x, y\}$, $I = \{x > 0, y > 0\}$, $G = \{x = 0, y = 0\}$ and $O = \{a, b, c\}$ where $a = \langle x > 0, y = 0; x\downarrow, y\uparrow\rangle$, $b = \langle y > 0; y\downarrow\rangle$, $c = \langle y > 0; y\downarrow, x\uparrow\rangle$. An instance of Q_{nest_2} is one where $x = 3$ and $y = 4$ in the initial state.

An action a is applicable in a qstate if it contains $pre(a)$. An action a is applicable in a state $\bar{s}$ if it satisfies $pre(a)$. If action a is applicable in state $\bar{s}$, a state $\bar{s}'$ is a successor state of $\bar{s}$, if the following conditions hold: $\bar{s}'$ contains $eff(a)$, for each $inc(v) \in N(a)$, $\bar{s}'(v) > \bar{s}(v)$, for each $dec(v) \in N(a)$, $\bar{s}'(v) < \bar{s}(v)$, and for each variable v not appearing in $eff(a)$ or $N(a)$, $\bar{s}'(v) = \bar{s}(v)$.

Definition 3. Given a QNP problem Q, a policy π for Q is a partial mapping from qstates to applicable actions.

Given a policy π and a state $\bar{s}$, let s be the qstate satisfied by $\bar{s}$, we use $\pi(s)$, if it is defined, to represent the action a assigned to $\bar{s}$, and $a(\bar{s})$ to represent the set of all possible successor states after applying a in $\bar{s}$. Given a policy π, a sequence of states $\bar{s}_0, \bar{s}_1, \ldots$ (finite or infinite) is called a π-trajectory if for $i \geq 0$, $\bar{s}_{i+1} \in a_i(\bar{s}_i)$ where $a_i = \pi(s_i)$.

Definition 4. For $\epsilon > 0$, a π-trajectory is ϵ-bounded, if for any action performed, if it decreases a numerical variable v, then either the old value of v is $< \epsilon$ and the new value equals 0 or the amount of decrease is $\geq \epsilon$.

Definition 5. Given a QNP problem Q, a policy π solves an instance of Q if for any $\epsilon > 0$, every ϵ-bounded π-trajectory started at the initial state is goal reaching. A policy π solves Q if it solves every instance of Q.

A policy π that solves Q_{nest_2} in Example 1 is given by: $\pi(\{x > 0, y > 0\}) = b$; $\pi(\{x > 0, y = 0\}) = a$; $\pi(\{x = 0, y > 0\}) = b$.

Definition 6. Given a QNP problem Q, a policy π terminates if for any instance of Q, for any $\epsilon > 0$, every ϵ-bounded π-trajectory started at the initial state is finite.

The qstate space of a QNP problem can be formalized as an AND/OR graph. We now introduce basic terminologies of AND/OR graphs.

An AND/OR graph $\mathcal{G} = \langle N, C\rangle$ consists of a set of nodes N and a set of connectors C, where a connector is a pair $\langle n, M\rangle$, connecting the incoming node $n \in N$ to a nonempty set of outgoing nodes $M \subseteq N$. A k-connector is a connector with exactly k outgoing nodes. Usually, $\mathcal{G}$ contains a distinguished initial node $n_0 \in N$ and a set of goal nodes $N_g \subseteq N$.

A path in $\mathcal{G}$ is a sequence of nodes successively linked by connectors. A subgraph of $\mathcal{G}$ is an AND/OR graph $\mathcal{G}' = \langle N', C' \rangle$ such that $N' \subseteq N$ and C' only contains connectors from C whose involved nodes (incoming and outgoing nodes) are contained in N'. Usually, we require that the initial node n_0 is contained in N' as the initial node n'_0 and the set of goal nodes $N'_g = N' \cap N_g$. In this paper, we restrict our attention to subgraphs where every non-goal node has at most one outgoing connector.

We say a subgraph $\mathcal{G}'$ is *closed* if every non-goal node in $\mathcal{G}'$ has exactly one outgoing connector. We say a subgraph $\mathcal{G}'$ is *proper* if for every non-goal node n' in $\mathcal{G}'$, there is a finite path starting at n' and ending in a goal node.

The qstate space of a QNP problem Q can be formalized as an AND/OR graph $\mathcal{G} = \langle N, C \rangle$. Specifically, nodes in N correspond to qstates of Q, the initial node corresponds to the initial qstate of Q and the goal nodes correspond to the goal qstates of Q. For an action a applicable in a qstate s, the application of a to s results in a set of qstates, denoted as $a(s)$. The action with $inc(v)$ results in a qstate with literal $v > 0$. The action with $dec(v)$ results in two qstates, one with literal $v > 0$ and the other with $v = 0$. There is a connector $\langle s, a(s) \rangle \in C$ for each qstate s and action a applicable in s. The AND/OR graph induced by Q_{nest_2} is shown in Figure 1 where the two nodes s_0 and s_g respectively correspond to the initial qstate and the goal qstate and the labels of connectors correspond to the names of actions in Q_{nest_2}.

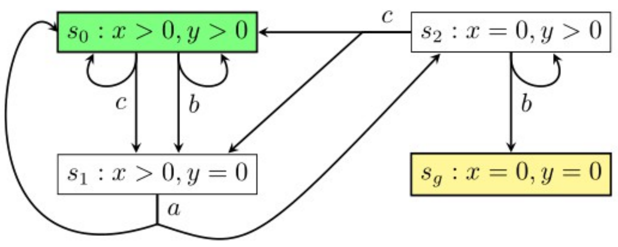

Figure 1: The AND/OR graph induced by Q_{nest_2}

Obviously, given a QNP problem Q and the AND/OR graph $\mathcal{G}$ induced by Q, there is a one-to-one correspondence between policies for Q and subgraphs of $\mathcal{G}$. Thus, in the paper, we will use "policy" and "subgraph" interchangeably.

A closed and proper subgraph is called a strong cyclic solution. The following result was proved by Srivastava et al. [2011]:

Theorem 1. *Given a QNP problem Q, a policy π solves Q iff it is a strong cyclic solution that terminates.*

Srivastava et al. [2011] introduce a sound and complete algorithm SIEVE, that verifies whether a policy π for Q terminates. Given G, the qstate transition graph induced by Q and π, SIEVE iteratively removes edges from G until G becomes acyclic or no additional edges can be removed as shown in Algorithm 1. SCC is the abbreviation for the strongly connected component of a graph.

Algorithm 1: SIEVE

```
  Input: $G$
1 repeat
2     Compute the SCCs of $G$;
3     Choose an SCC $g$ and a variable $v$ that is
        decreased but not increased in $g$;
4     if there is no such $v$ to choose then
5         return "Non-terminating";
6     Remove the edges $(s, s')$ such that $s$ and $s'$ are in
        $g$ and $\pi(s)$ has a $dec(v)$ effect;
7 until $G$ is acyclic;
8 return "Terminating";
```

3 Theoretic Foundation

In this section, we introduce the theoretic foundation for our algorithm, i.e., a simpler characterization of QNP solutions and a key observation about termination of policies.

The result of Srivastava et al. [2011] characterizes a solution to a QNP problem as a proper and closed subgraph that terminates. We now give a simpler characterization that a solution to a QNP problem is a closed subgraph that contains a goal node and terminates. Thus we replace the "properness" condition by the much relaxed condition of "presence of a goal node".

Theorem 2. *Given a QNP problem Q and the AND/OR graph $\mathcal{G}$ induced by Q, a subgraph $\mathcal{G}'$ of $\mathcal{G}$ is a solution to Q iff $\mathcal{G}'$ contains a goal node, is closed, and terminates.*

Proof. By Theorem 1, the only-if direction is easy since properness implies that there is a goal node. To prove the if direction, we only need to prove every non-goal node in $\mathcal{G}'$ is goal-reaching. We call nodes in $\mathcal{G}'$ that cannot reach a goal node deadends. Assume to the contrary that $\mathcal{G}'$ contains some deadends. Clearly, deadends can only reach deadends. Thus, there exists a SCC of deadends, otherwise, we get infinitely many deadends, contradicting that $\mathcal{G}$ is a finite graph. We call a SCC g a closed one if the outgoing nodes of all connectors in g are in g. Then there exists a closed SCC of deadends, otherwise, we get infinitely many SCCs, or we get a cycle of SCCs, which can be merged into a SCC, contradicting that each on the cycle is a SCC. Let g be a closed SCC of deadends. We now prove that this contradicts that $\mathcal{G}'$ passes the SIEVE test. For any variable v that is decreased in g, there is an edge (s, s') in g where $v > 0$ in s and $v = 0$ in s'; since s' is reachable from s in g, v must be increased in g. Thus no edge can be removed from g by SIEVE, so $\mathcal{G}'$ cannot pass the SIEVE test. □

Finally, we make the following observation:

Proposition 1. *Given a QNP problem Q, any policy that extends a non-terminating policy is also non-terminating.*

Proof. Let π be a non-terminating policy. Then there exists an infinite ϵ-bounded π-trajectory σ for some instance of Q and $\epsilon > 0$. Let π' extend π. Then σ is also a π'-trajectory. Thus π' is also non-terminating. □

4 Find Solutions for QNP Problems

With the above theoretic foundation, we propose a new approach that solves a QNP problem Q directly by searching subgraphs of the AND/OR graph $\mathcal{G}$ induced by Q. It will enumerate terminating subgraphs of $\mathcal{G}$ until a proper one that contains a goal node is found. When a non-terminating subgraph is identified, it will not be further expanded and thus pruning is done.

4.1 Avoid Non-terminating Solutions

An important part of our approach is to generate terminating subgraphs, which improves the basic idea of the generate-and-test approach by [Srivastava *et al.*, 2011]. The basic generate-and-test approach solves a QNP problem Q by iteratively generating and verifying the termination of strong cyclic solutions (also called candidate solutions) of Q one by one until a terminating one is found or all candidate solutions are enumerated and verified to be non-terminating. However, it is inefficient due to the independence between the generation procedure and the termination test procedure because candidate solutions which have been tested and fail to terminate are not fully utilized to avoid generating non-terminating candidate solutions. As shown by Proposition 1, any subgraph that extends a non-terminating subgraph is non-terminating. Therefore, if the termination of subgraphs can be verified in advance, the generation of subgraphs containing non-terminating ones can be avoided.

4.2 Detailed Algorithm

The pseudocode of our approach is shown in Algorithm 2. $\mathcal{G}$ is the AND/OR graph representing the qstate space of Q. The size of $\mathcal{G}$ is exponential with the number of variables in Q. s_0 is the initial node. S_g is the set of goal nodes. $\mathcal{G}'$ is a subgraph corresponding to a policy for Q and initialized as containing only s_0. A node in $\mathcal{G}'$ is said to be expanded if one of its outgoing connectors is chosen and added into $\mathcal{G}'$. $open$ is a stack storing the unexpanded nodes in $\mathcal{G}'$ and every node in $open$ is reachable from s_0. $closed$ is an ordered list storing the expanded nodes in $\mathcal{G}'$. The nodes in $closed$ are arranged in the order they are expanded.

In Algorithm 2, we will repeatedly pop the top node s from $open$ and try to expand s until $open$ is empty. If s is a non-goal node, it will try to expand s. When expanding, we ensure that $\mathcal{G}'$ is always terminating. A connector c is legal with respect to $\mathcal{G}'$ *iff* $\mathcal{G}'$ still terminates when adding c to it. Backtracking will be performed when failed to expand $\mathcal{G}'$ to satisfy conditions of terminating and containing a goal node.

When trying to expand s in Algorithm 3, we examine each unvisited connector of s, test if it is legal and mark it as visited. If a legal connector c is found, it will be added to $\mathcal{G}'$. S_c denotes the set of all outgoing nodes of c. We will add s to the end of $closed$ meaning that s is the latest expanded node. In order to backtrack correctly, it is necessary to record the incoming connector when pushing a node into $open$. We record $InConnector(s')$ for $s' \in S_c$ if s' has not been expanded and is not in $open$, and then push s' into $open$.

When failed to expand s, meaning that there is no legal connector for s with respect to $\mathcal{G}'$, we undo the marking of all connectors of s (to prepare for backtracking), then we backtrack or exit depending on whether s is the initial node as shown in Algorithm 2 Line 9-12. $s = s_0$ means that we cannot backtrack thus we return "Unsolvable". $s \neq s_0$ means that we have to examine the next outgoing connector of the last expanded node, which is the last node in $closed$. So backtracking will be performed as shown in Algorithm 4.

Algorithm 2: QNP Solver

```
Input: $\mathcal{G}$, $s_0$, $S_g$
Output: a solution graph $\mathcal{G}'$ or "Unsolvable"
1  $\mathcal{G}' \leftarrow \langle\{s_0\}, \emptyset\rangle$;
2  $open \leftarrow \{s_0\}$;
3  $closed \leftarrow \{\}$;
4  repeat
5      if $open \neq \emptyset$ then
6          $s \leftarrow open$.pop();
7          if $s \notin S_g$ then
8              EXPAND($s$);
9              if failed to expand then
10                 if $s = s_0$ then
11                     return "Unsolvable";
12                 else BACKTRACK;
13     else
14         if $\mathcal{G}'$ contains at least one goal node then
15             return $\mathcal{G}'$;
16         else BACKTRACK;
```

When $open$ is empty, meaning that $\mathcal{G}'$ has no unexpanded nodes, $\mathcal{G}'$ is closed. As shown in Algorithm 2 Line 13-16, we will exit or backtrack depending on whether $\mathcal{G}'$ contains goal nodes. If $\mathcal{G}'$ contains at least one goal node, it means $\mathcal{G}'$ satisfies all the conditions in Theorem 2 and it is a solution of Q. If there is no goal node in $\mathcal{G}'$, we will backtrack and try to re-expand the last expanded node as shown in Algorithm 4.

As shown by Bonet and Geffner [2020], the complexity of solution existence for QNP problems is EXP-Complete. However, in the worst case, the time complexity of Algorithm 2 is doubly exponential with the number of variables of the given QNP problem since it will enumerate all subgraphs of the induced AND/OR graph.

4.3 Soundness and Completeness

Theorem 3 (Soundness of Algorithm 2)**.** *Given a QNP problem Q, if Algorithm 2 returns a solution graph $\mathcal{G}'$, then $\mathcal{G}'$ corresponds to a policy that solves Q.*

Proof. Proving soundness of Algorithm 2 is equivalent to proving that the solution graph $\mathcal{G}'$ returned by Algorithm 2 satisfies all three conditions in Theorem 2.

Firstly, Algorithm 2 returns $\mathcal{G}'$ if the condition that $\mathcal{G}'$ contains at least one goal node and $open = \emptyset$ is satisfied. $open$ is empty means that all non-goal nodes in $\mathcal{G}'$ are successfully expanded and have exactly one connector, so $\mathcal{G}'$ is closed. Secondly, Algorithm 3 guarantees that only the legal connectors can be added into $\mathcal{G}'$ and thus $\mathcal{G}'$ always terminates. □

Algorithm 3: EXPAND(s)

```
1  $C_s \leftarrow$ the set of unvisited connectors of $s$ ;
2  foreach $c \in C_s$ do
3      $G \leftarrow$ Graph($\mathcal{G}' + c$);
4      mark $c$ as visited;
5      if SIEVE(G) = "Terminating" then
6          add $c$ to $\mathcal{G}'$;
7          add $s$ to the end of $closed$;
8          $S_c \leftarrow$ the set of outgoing nodes of $c$;
9          foreach $s' \in S_c$ do
10             if $s' \notin open \wedge s' \notin closed$ then
11                 $InConnector(s') \leftarrow c$;
12                 $open$.push($s'$);
13         return success;
14 $open$.push($s$);
15 mark all connectors of $s$ as unvisited;
16 return failure;
```

Algorithm 4: BACKTRACK

```
1  $s \leftarrow$ remove the last node of $closed$;
2  $c \leftarrow$ the assigned connector of $s$;
3  $S_c \leftarrow$ the set of outgoing nodes of $c$;
4  foreach $s' \in S_c$ do
5      if $InConnector(s') = c$ then
6          $InConnector(s') \leftarrow NULL$;
7          remove $s'$ from $open$;
8  remove $c$ from $\mathcal{G}'$;
9  $open$.push($s$);
```

Theorem 4 (Completeness of Algorithm 2)**.** *Given a QNP problem Q, if Q is solvable, Algorithm 2 will return a solution graph corresponding to one of the policies that solve Q.*

Proof. When expanding $\mathcal{G}'$ in Algorithm 3, the pruning strategy is based on SIEVE. We ignore the pruning strategy since it performs pruning only when any expanding leads to a non-terminating $\mathcal{G}'$, which can not be a part of the solution graph of Q according to Proposition 1.

We prove that Algorithm 2 enumerates all closed subgraphs, in which there is exactly one outgoing connector for each non-goal node. Algorithm 3 assigns one of the outgoing connectors to s_0 and add it to $\mathcal{G}'$. All non-goal nodes s reachable from s_0 in $\mathcal{G}'$ are pushed into $open$, and then will be assigned one outgoing connector and added to the end of $closed$, until $\mathcal{G}'$ is closed. In this process, all outgoing connectors of s are assigned to s one by one. When all outgoing connectors of s have been enumerated, it will backtrack to assign next outgoing connector of the last expanded node in $closed$. When all connectors of s_0 are enumerated and $open$ is empty, Algorithm 2 enumerates all closed subgraphs. If Q is solvable, it will return a solution graph of Q, one of all closed subgraphs that it enumerates. □

Domain	$\|f_Q\|$	$\|A_Q\|$	$\|f_F\|$	$\|A_F\|$	time(ms)
BlocksClear	2	4	20	16	7.2
BlocksOn	5	7	47	36	6.2
ChoppingTree	2	2	2	2	3.4
Cornera	2	4	29	23	2.9
Delivery$_1$	4	4	54	39	4.1
Delivery$_2$	4	5	54	40	7.3
Delivery$_3$	4	7	54	42	3.1
Gripper$_1$	4	5	54	44	3.3
Nest$_2$	2	2	29	19	10.2
Nest$_3$	3	3	47	33	3.9
Nest$_{10}$	10	10	285	243	11.0
Q$_1$	4	4	4	4	2.6
Q$_3$	4	4	41	29	5.7
Rewards	2	2	29	19	2.9
ShovelingSnow	3	3	47	30	7.7
TestOn	3	3	35	24	24.0
Logistic$_1$	20	12	517	439	13.4
Logistic$_2$	20	13	517	440	14.0
Nest$_3$u	3	3	47	36	7.8
Nest$_{10}$u	10	10	285	253	13.3
Gripper$_1$u	4	5	54	44	2.8
Q$_2$	4	4	41	29	5.5

Table 1: The size of QNP problems and the corresponding FOND problems obtained by **qnp2fond** and the time for translation. Problems in the top half part are solvable while the rest are not. $|f_Q|$ and $|A_Q|$ are the number of variables and actions in QNP problems respectively. $|f_F|$ and $|A_F|$ are the number of variables and actions in the corresponding FOND problems respectively.

5 Implementation and Experiments

Based on the proposed approach, we implemented a QNP solver DSET[1], meaning Direct Search and Early Termination-testing. We compared the performance of DSET with the existing compilation-based approach that compiles a QNP problem Q into a FOND problem P via a translator **qnp2fond** [Bonet and Geffner, 2020], which provides a sound and complete translation from QNP problems to FOND problems without termination testing. We solve the resulting FOND problems by two off-the-shelf FOND solvers: PRP [Muise *et al.*, 2012] and FOND-SAT [Geffner and Geffner, 2018]. We also compared with FOND-ASP solver proposed by Rodriguez et al. [2021] which solves QNP problems as FOND$^+$ problems by providing sets of fairness assumptions. We do not compare with the generate-and-test approach proposed by Srivastava et al. [2011] because there are no off-the-shelf FOND solvers that can enumerate all strong cyclic solutions and it is not simple to amend one solver to do so.

All the experiments were conducted on a Linux machine with a 2.9GHz Intel 10700 CPU and 4GB of memory.

5.1 Experimental Domains

We adopt the following domains:

- *ChoppingTree*, *Nest*$_2$, *Nest*$_3$ and *ShovelingSnow* are from Srivastava et al. [2011]. *BlocksOn*, *Delivery*$_2$,

[1]DSET is available at https://github.com/sysulic/DSET

	DSET		FOND-SAT		PRP		FOND-ASP	
Domain	time(s)	Policy size	time(s)	Policy size	time(s)	Policy size	time(s)	Policy size
BlocksClear	**0.0158**	**2**	0.1007	4	0.1067	4	0.0440	**2**
BlocksOn	**0.0117**	**7**	1.9778	11	0.2253	11	0.0471	**7**
ChoppingTree	**0.0127**	**2**	0.0658	3	0.0617	3	0.0430	**2**
Cornera	**0.0165**	**2**	0.2008	6	0.0875	5	0.5200	**2**
Delivery$_1$	**0.0139**	**6**	2.3177	11	0.2658	16	0.0523	**6**
Delivery$_2$	**0.0217**	**6**	2.1919	11	0.4689	16	0.1010	**6**
Delivery$_3$	**0.0287**	**6**	2.3840	11	0.4873	16	0.0586	**6**
Gripper$_1$	**0.0174**	**10**	114.9385	16	0.2463	28	0.1030	**10**
Nest$_2$	**0.0140**	**3**	0.3720	9	0.1621	7	0.0450	**3**
Nest$_3$	**0.0179**	**7**	3.5786	14	0.6258	15	0.0777	**7**
Nest$_{10}$	**5.1119**	**1023**	–	–	–	–	–	–
Q$_1$	**0.0126**	4	0.0735	4	0.0833	4	0.0465	**3**
Q$_3$	**0.0144**	6	0.3989	6	0.1260	7	0.0437	**4**
Rewards	**0.0155**	**2**	0.1886	6	0.1475	5	0.0408	**2**
ShovelingSnow	**0.0116**	**4**	1.0984	11	0.2461	10	0.0436	**4**
TestOn	**0.0122**	**3**	0.3167	7	0.1058	6	0.0433	**3**
Logistic$_1$	**15.6873**	**105**	–	–	–	–	58.2216	127
Logistic$_2$	–	–	–	–	–	–	–	–
Nest$_3$u	**0.0147**	–	–	–	22.0863	–	0.0460	–
Nest$_{10}$u	**2.0198**	–	–	–	–	–	21.2016	–
Gripper$_1$u	**0.0187**	–	–	–	743.6844	–	0.0586	–
Q$_2$	**0.0097**	–	–	–	1200.2490	–	0.0430	–

Table 2: Summary of the results for QNP problems by four solvers. The dashed line in the columns of time denotes out of time or memory while in the columns of size denotes “not applicable”.

Gripper$_1$ and *Rewards* are from Bonet et al. [2019]. *BlocksClear*, *Delivery*$_1$, *Delivery*$_3$, Q_1, Q_2 and Q_3 are from Bonet and Geffner [2020] and Q_2 is an unsolvable domain.

- To test the performance of DSET on domains of larger size, we design three new solvable domains: *Nest*$_{10}$ with a 10-fold nested loop, which is similar to *Nest*$_2$ and *Nest*$_3$, *Logistics*$_1$, which is a QNP abstraction of the classical Logistics problem by grouping indistinguishable objects, and *Logistics*$_2$, which adds to *Logistics*$_1$ an extra action.
- Considering that a QNP solver means that it not only returns a solution for solvable problems, but also informs about unsolvability for unsolvable problems, just like SAT solvers, we add three unsolvable domains *Nest*$_3$*u*, *Nest*$_{10}$*u* and *Gripper*$_1$*u* by modifying the corresponding domains. More specifically, the preconditions and effects of some actions are rewritten so as to make the domains unsolvable.

Table 1 shows the size of the benchmark QNP problems and the corresponding FOND problems generated by **`qnp2fond`** and the time for translation. In most cases, the corresponding FOND problems P have a much larger size than the original QNP problems Q because **`qnp2fond`** adds many extra variables and actions involving the stack operations for each numerical variable to guarantee that the solution of P corresponds to a solution of Q. Generally speaking, the number of stack-related variables and actions in P is cubic polynomial in the number of numerical variables in Q.

5.2 Main Results

We compared the performance of the four solvers, each of which is evaluated in terms of solving time (the running time needed to solve a problem) and policy size (the number of qstate-action pairs in the solution) as shown in Table 2. The timeout is set to 1800 seconds to prevent a solver from running indefinitely. To eliminate the interference of background processes, for each problem, we run each solver 10 times, which is an empirical value, and take the average as the reported running time. The running time in the columns of FOND-SAT and PRP is the sum of the running time of the translator **`qnp2fond`** and the FOND solver.

The solving time of DSET is less than that of FOND-SAT and PRP by more than an order of magnitude on most solvable QNP problems. The reason is that DSET searches for solutions in a much smaller qstate space and performs pruning when a non-terminating subgraph is found, while FOND-SAT and PRP struggle with a much larger qstate space, especially for those hard problems like *Nest*$_{10}$, *Logistic*$_1$ and *Logistic*$_2$. The running time of DSET increases with the size of QNP problems. *Logistic*$_1$ can be solved within 16 seconds; however, with just one extra action, *Logistic*$_2$ makes DSET run out of time. Since DEST is a naive enumeration method, these two domains are designed to test the border of scalability. When QNP problems are unsolvable, DSET returns “Unsolvable” within seconds while PRP and FOND-SAT usually take a much longer time and even timeout. For most QNP problems, solvable or unsolvable, the running time of DSET is comparable with that of FOND-ASP since they search or compute solutions in the same qstate space. When computing

Nest$_{10}$, FOND-ASP runs out of memory.

The policies returned by DSET usually have a smaller size. A smaller policy is not necessarily more efficient, but it is more succinct. The reason why PRP and FOND-SAT return larger policies is that the FOND problems generated by **qnp2fond** require that a numerical variable v can be decreased only after v is pushed into the stack and v can be increased if v is not in the stack. If a numerical variable is in the stack, it must be popped before it can be increased. With this mechanism, PRP and FOND-SAT will return policies of larger size since some *Push* or *Pop* actions must be applied before a numerical variable is decreased or increased.

In the following part, we focus on the analysis of one typical unsolvable QNP problem to illustrate why DSET returns "Unsolvable" much more quickly than PRP and FOND-SAT.

Example 2. $Q_2 = \{F, V, I, O, G\}$ is an unsolvable QNP problem where $F = \{p, g\}$, $V = \{x, y\}$, $I = \{p, \neg g, x > 0, y > 0\}$, $G = \{g\}$ and $O = \{a_1, a_2, a_3, a_4\}$ where $a_1 = \langle p, x > 0; \neg p, x \downarrow\rangle$, $a_2 = \langle \neg p; p, x \uparrow\rangle$, $a_3 = \langle x = 0; g\rangle$ and $a_4 = \langle y = 0; g\rangle$.

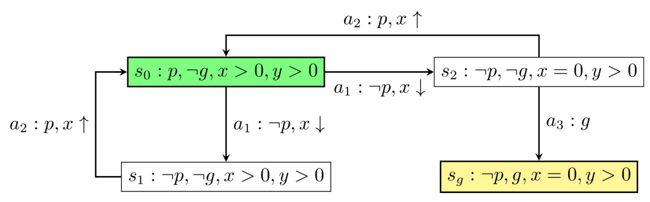
$a_2 : p, x \uparrow$
$s_0 : p, \neg g, x > 0, y > 0$
$s_2 : \neg p, \neg g, x = 0, y > 0$
$a_1 : \neg p, x \downarrow$
$a_2 : p, x \uparrow$
$a_1 : \neg p, x \downarrow$
$a_3 : g$
$s_1 : \neg p, \neg g, x > 0, y > 0$
$s_g : \neg p, g, x = 0, y > 0$

Figure 2: The Qstate Transition Graph of Q_2

As shown in Figure 2, a_4 is inapplicable in any qstates reachable from s_0 since there is no action with the effect $y \downarrow$ and thus the precondition of a_4 will never be satisfied. Moreover, it can be easily verified that Q_2 has no solution because the action a_2, which must be used to restore the precondition of a_1 in the loop, increases x at the same time. Therefore, x will be decreased and increased in the loop forever and the policy will not terminate.

It took DSET 0.0097 seconds to return "Unsolvable" for Q_2, which is negligible compared with the running time of PRP and FOND-SAT. It took PRP 1200.2490 seconds to return "Unsolvable" because the corresponding FOND problem of Q_2, has a much larger size and makes it time-consuming for PRP. Specifically, PRP can prune the qstate space significantly by locating and propagating the deadends, which is based on the relevance between qstates. However, **qnp2fond** introduces a lot of stack-related variables and thus results in weaker relevance between qstates.

5.3 Evaluation of the Pruning Strategy

Considering that the dataset shown in Table 1 is limited, to measure the impact of pruning non-terminating policies ahead of time, we set the number of numeric variables M and the number of actions N, randomly generate a set of 500 QNP problems $Q = \langle F, V, I, G, O\rangle$ where $|F| = 3$, $|V| = M$, $|O| = N$, as follows. I contains a literal of each variable in $F \cup V$; G and $pre(a)$, $a \in O$, contain a literal of each variable in $F \cup V$ with probability 0.5; *eff*(a) contains a literal of each variable in F with probability 0.5, and contains the atom $inc(v)$ or $dec(v)$ of each variable in V with probability 0.5. Each literal in I, G and O is produced by randomly choosing from a pair of complementary literals of the corresponding variable. Each atom in *eff*(a) is produced by randomly choosing from a pair of $inc(v)$ and $dec(v)$ of the corresponding variable v.

Domain	$\lvert V_Q\rvert$	$\lvert A_Q\rvert$	pruning(s)	no pruning(s)
Random$_1$	8	12	0.3594	0.0753
Random$_2$	10	20	1.3115	8.8865

Table 3: Additional experiments. $|V_Q|$ and $|A_Q|$ are the number of numeric variables and actions in QNP problems respectively.

As shown in Table 3, we randomly generate 2 sets of 500 QNP problems using the method described above; compute the average running time of DSET on each set and compare the results with and without the pruning strategy. It turns out that there is a trade-off between pruning but doing termination-testing on non-closed subgraphs and non-pruning but only doing termination-testing on closed subgraphs. On Random$_1$, the pruning strategy does not pay off; however, pruning shows its significant advantage on Random$_2$.

6 Conclusions

In this paper, we first propose a simpler characterization that a QNP solution is a closed and terminating policy with a reachable goal node, and make the observation that any extension of a non-terminating policy is also non-terminating. Based on the above characterization and observation, we propose a native QNP solver DSET, which works by formalizing the qstate space of a QNP problem Q into an AND/OR graph $\mathcal{G}$ and performing a subgraph search algorithm on $\mathcal{G}$ for solutions with a pruning strategy that involves termination tests of subgraphs. Experimental results show that on solvable QNP problems, the running time of DSET is less than that of the compilation-based approach proposed by Bonet and Geffner[2020] using FOND solvers PRP and FOND-SAT by more than an order of magnitude. For unsolvable QNP problems, DSET returns "Unsolvable" much more quickly than PRP and FOND-SAT. Meanwhile, DSET is comparable with the FOND$^+$ solver FOND-ASP by Rodriguez et al. [2021].

However, DSET is a naive enumeration method and thus suffers from scalability as shown by the *Logistic*$_2$ domain and classical planning instances since QNP extends classical planning. In the future, We will try to improve scalability of DSET with heuristic search. We are also interested in expanding the work presented in this paper to solve QNP problems where numeric variables are increased or decreased by restricted amounts rather than by some arbitrary amount. Such problems can function as better abstraction models for generalized planning.

Acknowledgments

We acknowledge support from the Natural Science Foundation of China under Grant No. 62076261.

References

[Bercher and Mattmüller, 2009] Pascal Bercher and Robert Mattmüller. Solving non-deterministic planning problems with pattern database heuristics. In *KI 2009: Advances in Artificial Intelligence*, pages 57–64, 2009.

[Bonet and Geffner, 2018] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4667–4673, 2018.

[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.*, 2019] Blai Bonet, Guillem Francès, and Hector Geffner. Learning features and abstract actions for computing generalized plans. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 2703–2710, 2019.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[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)*, pages 88–96, 2018.

[Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artif. Intell.*, 129(1-2):35–62, 2001.

[Illanes and McIlraith, 2019] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 7610–7618, 2019.

[Levesque, 2005] Hector J. Levesque. Planning with loops. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 509–515, 2005.

[Mattmüller *et al.*, 2010] Robert Mattmüller, Manuela Ortlieb, Malte Helmert, and Pascal Bercher. Pattern database heuristics for fully observable nondeterministic planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 105–112, 2010.

[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)*, 2012.

[Nilsson, 1982] Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer, 1982.

[Rodriguez *et al.*, 2021] Ivan D. Rodriguez, Blai Bonet, Sebastian Sardiña, and Hector Geffner. Flexible FOND planning with explicit fairness assumptions. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS)*, pages 290–298, 2021.

[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)*, pages 1010–1016, 2011.