Flexible Strategies for State Space Exploration

LUUK ALFING, University of Twente, The Netherlands

Exploring large state spaces is a common challenge in many domains, requiring systematic strategies to solve complex problems. Existing tools are often designed around specific strategies or goals, limiting their reusability. This study proposes the development of a flexible, unified framework for state space exploration, allowing for customization of exploration strategies based on various strategy features. The framework emphasizes modularity and extensibility, and is evaluated through the implementation of multiple configurations, tested across different state space exploration problems.

Additional Key Words and Phrases: State space exploration, Flexible exploration framework, Feature model, Strategy features, State space analysis

1 Introduction

State space exploration is the technique of searching through a graph structure consisting of states, which represent a configuration of a system at a specific moment in time, and transitions connecting one state to another, which represent a change made to the configuration of a system.

State space exploration is used in many disciplines of computer science, such as model checking [16] and program analysis, where it plays a crucial role in efficiently navigating large and complex state spaces [19]. Moreover, it is relevant in other fields, including game theory [21], physics [30], and medical applications [3, 5, 6].

Within the diversity and broad range of state space exploration, a problem occurs. Every unique exploration problem requires its own strategy to solve it or find the desired outcome. As noted by Ganai et al. [15], who designed a hybrid search strategy, no single strategy is optimal to efficiently solve the wide variety of problems. Most of the time, when a new problem needs a solution using state space exploration, an efficient strategy has to be researched and implemented, as existing strategies all have their own advantages, disadvantages, and proper applications [22].

This is also evident in the current state of the art on this topic, which consists primarily of research carried out to test or develop efficient algorithms for a specific exploration problem [11, 14] or to compare different algorithms or variations [8]. Instead, this research will focus on the creation of a flexible state space exploration framework that supports multiple strategies and features, so as to have an appropriate exploration strategy for diverse state space exploration problems.

To address the wide variety, relevant features will be identified and transformed into a flexible framework that is capable of exploring a state space exploration problem based on selected features. This gives rise to the following overarching research question:

> How can a flexible framework support the wide range of existing strategies for solving state space exploration problems?

This can be divided into the following three sub-questions:

(1) What are the essential exploration features required for a flexible state space exploration framework?
(2) How can different exploration strategies be abstracted and integrated into a unified framework?
(3) How well does a flexible exploration framework perform when applying different strategies to different state space problems?

Chapter 2 will provide background on state space exploration strategies and feature models. In Chapter 3, existing exploration frameworks and their limitations will be highlighted. The feature identification and framework design and implementation will be discussed in Chapter 4. Next, Chapter 5 will evaluate the flexible strategies framework across multiple problems. Finally, Chapter 6 discusses the findings, limitations, and contribution of this research.

2 Background

2.1 State Space

The process of state space exploration can be formalized as the search over a state space. In this research, a state space is defined as:

$$\mathcal{S} = (S, A, T, s_0, G)$$

with:
- S, the set of all possible states.
- A, the set of all available actions that lead to a transition.
- $T \subseteq S \times A \times S$, the set of transitions.
- $s_0 \in S$, the initial state.
- G, the set of goal states.

Each transition $(s, a, s') \in T$ represents a labeled move from a state s to a successor state s' via a action a. The goal of exploration is typically to determine whether a state in G is reachable from the initial state s_0, making the reached goal states the solutions. Moreover, a solution can be a trace, being a valid sequence of transitions that leads to the goal state. Such a solution can be defined as:

$$\rho = \langle t_1, t_2, \ldots, t_n \rangle \mid \forall i \in \{1, \ldots, n\},\ t_i = (s_{i-1}, a_i, s_i) \in T,\ s_n \in G$$

The exploration of a state space can be viewed as progressing through layers of states, where each layer contains all states at the corresponding depth. In other words, all states that are reachable from the initial state s_0 by the same fixed number of transitions. Formally, the depth of a state $s \in \mathcal{S}$, denoted as $d(s)$, can be defined as the length of the shortest path from the initial state s_0 to s:

$$d(s) = \min\{n \in \mathbb{N}_0 \mid \exists \rho = \langle (s_0, a_1, s_1), \ldots, (s_{n-1}, a_n, s_n = s) \rangle \subseteq T\}$$

And a specific layer, denoted as L_k can be defined as the set of all states at depth k:

$$L_k = \{s \in S \mid d(s) = k\}$$

TScIT 43, July 4, 2025, Enschede, The Netherlands

To solve state space exploration problems efficiently, especially when the state space is large, heuristic functions are often used to guide the search [28]. A heuristic is a function $h(s)$ that estimates the cost from a given state s to a goal state $g \in G$, which can be defined as:

$$h(s) : S \rightarrow \mathbb{R}_{\geq 0}$$

A heuristic function has two important properties [7, 24]:

- *Admissibility*: a heuristic should never overestimates the true minimal cost $\delta(s)$ from any state s to a goal, which can be defined as:

$$\forall s \in \mathbb{S}, \quad h(s) \leq \delta(s)$$

- *Consistency*: a heuristic is consistent if for every state s and its successor s', the estimated cost satisfies:

$$h(s) \leq c(s, s') + h(s')$$

where $c(s, s')$ denotes the cost of transitioning from s to s'.

2.2 Strategies

Various space state exploration problems have given rise to the introduction of a wide range of exploration strategies. The strategies all take a different approach to reach the goal of the exploration. Below, a handful of the most well-known strategies will be highlighted, ranging from simple and intuitive strategies to more complex ones.

2.2.1 Breadth First Search. One of the most common exploration strategies in state space exploration is Breadth First Search (BFS). The BFS strategy searches through the state space by fully exploring one layer before continuing to explore deeper in the state space [22]. As a result, BFS is an effective strategy to find the shortest path (the least number of steps) to the goal.

2.2.2 Depth First Search. Another common exploration strategy is Depth First Search (DFS), which contrasts with BFS in its traversal behavior. Instead of exploring a state space layer by layer, DFS explores one path as deeply as possible before backtracking and exploring alternative branches [22]. Therefore, DFS is a viable alternative to BFS that often uses less memory but does not guarantee a shortest path [20].

2.2.3 Greedy Search. Both BFS and DFS are uninformed strategies, which means that they do not use additional knowledge about the problem to solve it. A strategy that does use such additional knowledge is greedy search. It does this by using a heuristic function that approximates the distance from a state to the goal. This value is then used to prioritize exploring the states that are estimated to be closest to the goal [28].

2.2.4 A Search.* Similar to greedy search, the A* search strategy uses a heuristic function to estimate the distance to the goal and with that approximate the best states to explore. However, in contrast to greedy search, it combines this heuristic function with a cost function. This cost function represents the cost to reach a state (the distance from the initial state s_0), and allows the A* exploration strategy to reliably find the shortest paths to the goal while using the advantages of a heuristic function [13, 25].

2.2.5 Symbolic State Space Exploration. An alternative to previously discussed explicit state space exploration is symbolic state space exploration. Rather than exploring individual states, symbolic state space exploration operates on symbolic representations of sets of states [8, 9]. As a result, symbolic state space exploration can be an effective strategy to solve extremely large state space exploration problems [8, 17].

2.3 Feature Modeling

Creating feature models is a technique commonly used in software product line engineering to model the variability and reusability in a system or complete product line [1, 2]. Feature model diagrams visualize the features of a system in a hierarchal structure, where each level of the diagram becomes increasingly detailed. Moreover, they define relationships between the features such as mandatory, optional, and alternative, as well as constraints over the features such as requires, indicating that one feature requires another feature or a specific option of that feature [2].

This study uses feature models to provide a clear and concise overview of the differences and variability present in state space exploration strategies, which can be used to design a flexible exploration framework.

Various studies have been conducted to develop a full understanding of feature models and their potential [1, 4, 26]. These efforts include approaches to transform feature models into concrete implementation [1], as well as techniques to analyze feature models [4]. Both act as crucial steps in this study, specifically in implementing the flexible framework and in creating and validating the design of the feature model.

3 Related Work

State space exploration has been a relevant topic of research for many years [15, 16, 19]. As already described, a lot of these works look into a single optimized strategy, however, studies related to a flexible strategies framework have been performed as well

Firstly, Kattenbelt [20] argues that most model checking tools are highly specialized and optimized for a specific purpose, making it extremely difficult to reuse or extend on them for other related problems that require a different approach, and therefore proposes a modular approach to model checking. Their modularity is more about how tools interact, whereas this framework focuses on how strategies are built.

Another related work is [17], in which Heijblom investigates how features of models can be used to determine the optimal exploration strategy. While their research aims to find optimal strategies for various exploration problems, this study aims to design a framework that allows for the configuration of search strategies.

Finally, in their study, Rasmussen et al. [23] use agent-based methods to build adaptable exploration strategies. While the implementation differs, this study also aims at creating adaptable strategies.

4 Methodology and Approach

4.1 Feature Model

The first step in this study is to identify the relevant exploration features and to create a feature model diagram based on those features. To do this, existing search strategies, some of which have been discussed in Section 2.2, have been analyzed to discover the features that make each of these strategies unique. The resulting feature model diagram can be found in Figure 1, and will be explained in more detail.

4.1.1 Search goal. An essential part of every exploration strategy is the search goal: every state space exploration problem aims to reach a state part in the set G, as defined in Section 2.1. However, this goal can be completely different for different problems, depending on the specific problem to be solved. For example, a goal could be a specific state [13] or transition, or it could be to satisfy or violate some property about a state or transition [25]. The only consistency between state space exploration problems is that the goal should be about states or transitions, as these are the building blocks of a state space. Therefore, in this research, the search goal can be any boolean expression regarding states and transitions, specifically:

- State s, the id of a state.
- Transition t, the label of a transition.
- Loop, whether a transition is a loop from a state to itself.
- Outdegree n, the number of outgoing transitions from a state.

With these a goal expression can be created like this:

(Transition "finish" & Loop) | Outdegree 0

Meaning, find states with a transition to itself called finish, or with no outgoing transitions.

4.1.2 Next State Selection. The main difference between the strategies discussed in Section 2.2 is their next state selection policy, which is the next feature identified in this study. The next state selection can be divided into two categorizes, *informed* and *uninformed* search. Informed search uses a heuristic function that approximates how good a state is to explore, so that the most promising states can be prioritized. The following uninformed state selection policies are identified for this study:

- Oldest, explore the least recently discovered state next.
- Newest, explore the most recently discovered state next.
- Random, choose a random discovered state to explore next.

For the informed state selection policies, the main component is the heuristic function. Another component is the state acceptance, which describes whether a state can be explored based on its heuristic value, and is further highlighted in Section 4.1.4. Some informed search strategies also use a cost function, which represents the distance from a state to the initial state s_0. Therefore, informed state selection consists of a heuristic function and an optional cost function: $f(s) = h(s)$ or $f(s) = g(s) + h(s)$

Here, s represents a state and $f(s)$ the function that returns a value that shows how promising the state is, with the lowest value being the most promising. The function $g(s)$ represents the cost function $c(s_0, s)$ and $h(s)$ the heuristic function as defined in Section 2.1.

There are numerous heuristic functions that can be used depending on the precise kind of state space exploration problem and structure of the states. Depending on the structure of the states, different heuristics are possible, as the more information a state contains the more information can be used to estimate the most promising state. For example, a state can have an id that simply states the order in which it was generated, or an id that provides more information, or a state could even be a graph structure that also contains its own states and transitions. Some examples of heuristics are as follows:

- Node-Edge-Node Count, compare the number of transitions inside a state to those of the goal state.
- Transitions Count, the number of outgoing transitions of a state [29].
- Rule Violation Count, the number of properties that do not hold in a state, but are satisfied in the goal state [25].
- Hamming Distance, number of bits that differ between an encoded state and the encoded goal state [13].

4.1.3 Frontier Size. The next identified feature is the size of the frontier. The frontier refers to the set of states that are discovered but not yet (fully) explored, meaning there may still be more outgoing unexplored transitions from those states to explore. Thus, the frontier size is the (maximum) number of states in that set. These are the options for the frontier size feature:

- Single, a frontier size of 1.
- Fixed, a frontier size of some number greater than 1.
- Complete, no limit on the size of the frontier.

A frontier size of 1 allows for a form of linear search, in which the strategy will always solely look at a single state to further explore, leaving no room for backtracking. A fixed frontier size results in the so-called *beam search*, in which the frontier size is used to create an optimal balance between finding the best solutions and being efficient with memory [28].

4.1.4 State Acceptance. When using an informed exploration strategy, the next state selection is based on the most promising unexplored state, as described in Section 4.1.2. The goal of this is to not waste time exploring states that have a low chance of finding a solution. This creates another component, namely state acceptance, with the following two options:

- Allow Worse, also explore states that seem less promising than the already explored states.
- Strictly Better, only explore new states that seem more promising than the already explored state.

When accepting strictly better states, states will not only be ranked on their heuristic value to determine which one to explore first, but also states with a lower heuristic value than currently explored states will not be explored at all. According to research [28], this can be an effective way to improve efficiency but offers no guarantees of finding a solution, as exploration could get stuck in local extrema where a state lacks better successors, preventing progress toward the goal.

4.1.5 Result. Another relevant feature identified in this study is the type of result. Depending on the problem being explored, it may be interesting to find out how often a goal occurs, where it occurs, or how it occurs. Therefore, the result feature has these options:

- State, return the states in which the goal condition is met.
- Trace, return the full path to the states in which the goal condition is met.

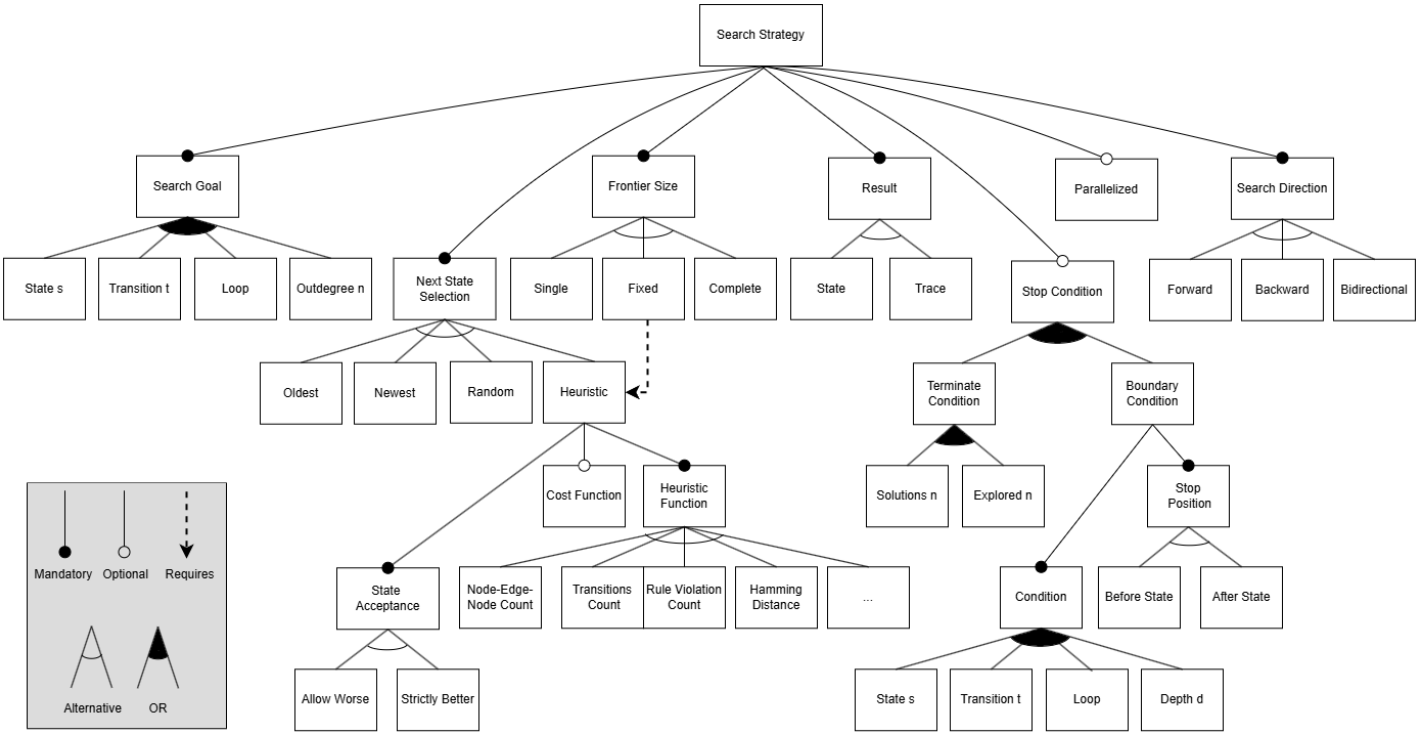

Fig. 1. Feature Model Diagram

4.1.6 Stop Condition. Another important feature is the stop condition. To efficiently search through a state space, it is beneficial to know when to stop searching in a specific direction or completely. For example, if you know that after a particular transition there will not be any chance to find a solution, then there is no reason to keep searching further behind that transition. There are two types of stop condition in this research, namely the terminate condition, which stops the search entirely when this condition is met, and the boundary condition, which prevents the search from continuing from the state in which the condition is met. The options for the terminate condition are:

- Solutions n, stop the search when a certain number of solutions are found.
- Explored n, stop the search when a certain number of states are explored.

The boundary conditions has two components, namely the condition and the stop position, which is explained in the next section. Options for the boundary conditions are:

- State s, do not continue searching after a state with a certain id.
- Transition t, do not continue searching after a transition with a certain label.
- Loop, do not continue searching after a transition from a state to itself.
- Depth d, do not continue searching after reaching a certain depth in the state space graph.

Similarly to the search goal, the options for the terminate condition and the boundary condition can be combined to create an expression.

4.1.7 Stop Position. As explained above, the boundary condition determines when successors of a state should not be explored. However, this gives rise to the question, should exploration be stopped while exploring the state that met the boundary condition or after that state? In other words, can a state in which a boundary condition is met be a solution? Therefore, the stop position component is identified with the following two options:

- Before State, the state that meets the stop condition cannot be part of the solutions, even if it meets the goal condition criteria.
- After State, the state that meets the stop condition can be part of the solutions if it also meets the criteria of the goal condition.

4.1.8 Parallelized. To improve the efficiency of an exploration strategy, research has indicated that parallelizing it could be an option [9, 12, 14]. That means that multiple parts of the exploration process will be run simultaneously on multiple cores or machines. This is an optional feature without multiple options and therefore can either be included or not included in the exploration strategy.

4.1.9 Search Direction. The final identified feature in this study is the search direction. Usually a search is performed from the starting state to a goal state, but that is not the only option. Research has indicated that searching from both the initial state and the goal state can improve the performance of some exploration strategies [7]. The search direction feature has the following options:

- Forward, search from the initial state to a goal state.
- Backward, search from a goal state to the initial state.
- Bidirectional, search from both the initial state and a goal state and find a solution by meeting in the middle.

4.1.10 categorization The features identified in this section can be divided into two main categories, which can be found in Table 1. The first category are the features that explain the precise way the strategy searches through the state space. They describe which states will be explored and in what order. Together, these features represent the part of the strategy that can be applied to any problem.

The second category of features consists of those that describe how the precise manner of search, as defined by the first category, behaves on a specific problem. They describe what the strategy should search for, when it should stop searching, and what type of result is expected for a specific state space exploration problem.

Table 1. Feature Categorization

Category 1	**Category 2**
Next State Selection	Search Goal
Frontier Size	Result
State Acceptance	Terminate Condition
Parallelized	Boundary Condition
Search Direction	Stop Position

4.2 Validation Table

The feature model aims to provide a flexible and scalable framework to create an exploration strategy by selecting between its described features. To ensure its completeness and practical relevance, an essential criterion is that it should also support existing strategies.

Validating this capability of a feature model is not only crucial for this study, but also represents an effective analysis technique for feature models in general, as explored in previous work [4].

To properly validate the feature model, various exploration strategies have been selected, some of which are described in Section 2.2. Additionally, other strategies are selected which can be found in Table 2 Since these strategies can be applied to any exploration problem, they are classified according to the features in the first category in Table 1.

Table 2. Validation Strategies

Strategy	**Abbreviation**	**Sources**
Random Walkthrough	RW	[12, 24]
Hill Climbing	HC	[28]
Greedy Search	Greedy	[28]
A* Search	A*	[22, 28]
Reverse A* Search	Rev-A*	[27]
Beam Search	Beam	[24, 28]
Bidirectional BFS	Bi-BFS	[7, 24]
Linear Search	Linear	[10]
Parallel DFS	P-DFS	[12]

The result of the validation can be found in Table 3, in which an empty cell is used to illustrate that a particular feature is not applicable to that specific strategy or that any option is possible.

Table 3. Feature Model Validation Table

Feat. / Strat.	**Next State Selection**	**Frontier Size**	**State Acceptance**	**Parallelized**	**Search Direction**
BFS	Oldest	Complete		*Excluded*	Forward
DFS	Newest	Complete		*Excluded*	Forward
RW	Random	Complete		*Excluded*	Forward
HC	Heuristic	Single	Strictly Better	*Excluded*	Forward
Greedy	Heuristic	Complete	Allow Worse	*Excluded*	Forward
A*	Cost + Heuristic	Complete	Allow Worse	*Excluded*	Forward
Rev-A*	Cost + Heuristic	Complete	Allow Worse	*Excluded*	Backward
Beam	Heuristic	Fixed	Allow Worse	*Excluded*	Forward
Bi-BFS	Oldest	Complete		*Excluded*	Bi-directional
Linear		Single	Allow Worse	*Excluded*	Forward
P-DFS	Newest	Complete		Included	Forward

4.3 Implementation

As discussed in Chapter 2.3, there are approaches to transform feature models to code implementation. These approaches typically involve translating the feature model into an intermediate representation that can then be transformed more directly into an actual implementation [1]. However, due to the relative simplicity of the feature model design in this research, this approach has not been followed for the implementation of the flexible framework.

Instead, to implement this tool, a general explorer is created that calls methods in interfaces representing the features, together with implementations of the interfaces representing the options of those features, and flags are checked to alter the exploration strategy for the basic features. The implementation is available on GitLab [1].

4.3.1 Explorer. One of the main components of the exploration tool is the explorer. The explorer is responsible for exploring the state space using the selected strategy, and its main method can be found in Algorithm 1. The explorer searches through the state space by looping over the states provided by the next state selector and then checking three conditions, which represent the following features:

(1) The search goal.
(2) The terminate condition.
(3) The boundary condition.

Moreover, to integrate the stop position, the currently explored state gets added to a list of excluded states when the boundary condition is met and the stop position is set to stop before the state, to ensure that the state can be excluded from the final result.

[1] https://gitlab.utwente.nl/s2995808/flexible-state-space-exploration-strategies

Algorithm 1 Explore state space

```
result ← ∅
excluded ← ∅
steps ← 0
while transition ← selectNext ≠ null do
    steps ← steps +1
    state ← getTargetState(transition)
    if isGoal(state, transition, steps, |result|) then
        result ← result ∪ {state}
    end if
    if isTerminate(state, transition, steps, |result|) then
        break
    end if
    if isBoundary(state, transition, steps, |result|) then
        if stopBeforeState then
            excluded ← excluded ∪ {state}
        end if
        continue
    end if
    addToFrontier(state)
end while
result ← result \ excluded
return result
```

4.3.2 State Selector. The state selector task is to provide the next state to explore. To implement this behavior, an interface was created with select next as its main method. The next state selection options, as identified in the feature model (Figure 1), all implement this next state method. An example of these implementations is that of the oldest next state selector, which can be found in Algorithm 2 and looks similar to the other next state selection options.

Algorithm 2 Select next state

```
while frontierState ← first state in frontier ≠ null do
    next ← getSuccessor(frontierState)
    if next = null then
        remove the first state from frontier
        continue
    end if
    return next
end while
return null
```

4.3.3 Excluded Features. There are two features identified and included in the feature model, which got excluded from the implementation. Due to time constraints, not all identified features could be taken into account. These features are parallelized and search direction, as these features would have taken significant effort to implement compared to the other features, and these features are not unique to an existing search strategy but instead can be seen as additions to strategies to possibly make them more efficient.

4.3.4 Conditions. The next important component to discuss are the conditions. Since the search goal and stop conditions are modeled using an OR-relation, a condition may combine multiple of these options into a single expression. To support this, the system had to be implemented in a flexible way that allows for any combination of these options, which is done by creating an ANTLR grammar[2]. An example of a condition expression using this grammar could be the following boundary condition:

(Transition "end" & !Loop) | Depth 10

The conditions in the implementation allow for even greater flexibility and customization than originally envisioned in the feature model. The available options for the terminate condition, boundary condition, and search goal often overlap and could be applied to any of the three. Therefore, the implementation allows for any combination of all available options for each condition type.

4.3.5 Configuration. The configuration of the search strategy is split into two parts, following the categorization specified in Table 1. The first part is the strategy configuration, where the features from the first category are set using the format: $Feature : option$, which are then parsed to configure the system accordingly. An example of a strategy configuration for greedy search:

$State\ selector : heuristic$
$Frontier\ size : complete$
$State\ acceptance : allow\ worse$

The second part is the problem configuration, which sets features from the second category. An example of the ferryman problem [25]:

$Search\ goal : Transition\ "finish"\ \&\ loop$
$Terminate\ condition : solutions\ 1$
$Boundary\ condition : Transition\ "edible"$
$Stop\ Position : before$
$Result\ type : trace$

Then they are tied together using the final search configuration:

$Problem\ configuration\ name : ferryman$
$Strategy\ configuration\ name : greedy$
$State\ space\ generator\ name : AutGenerator$

4.3.6 Generation. Finally, there is the generation component. When exploring a state space, generally, the state space is not known from the start and should be generated while exploring. The tool implemented in this research focuses on exploration and flexible strategies to explore state spaces. However, it still requires the generation aspect, which is integrated in this application by providing an interface that could be implemented to communicate with a tool that is specialized in the generation of state spaces. This interface includes methods for providing the initial state, successors of a state, and the heursitic value of a state.

An implementation of this generation interface is provided. This implementation does not integrate any other tool into the system. Instead, it expects a file containing the entire state space in .aut format[3], representing the transition system of the state space.

5 Results

5.1 Evaluation

To understand the performance of the developed flexible framework, multiple strategies have been tested on various state space exploration problems. This section will describe these problems and how they will be evaluated.

[2] https://www.antlr.org/
[3] https://cadp.inria.fr/man/aut.html

5.1.1 Dining Philosophers Problem. The first exploration problem that will be used to evaluate the framework is the dining philosophers problem. In this problem, a number of philosophers are sitting around a table. These n philosophers are hungry and have to pick up two forks to be able to start eating. There are also n forks, which means that philosophers have to compete against each other for the required forks [29]. A philosopher can either pick up the fork on their left, pick up the fork on their right, or place the forks back on the table.

This problem simulates the concurrency of programs [20], and the goal is to find deadlocks, which are states without outgoing transitions. There are two solutions in this problem, namely, each philosopher picks up the fork on their left, resulting in no philosopher being able to take the fork on their right, and each philosopher picking up the fork on their right. The goal of this evaluation is to find these two solutions while exploring the least number of states.

Since the goal is to find states with no outgoing transitions, the heuristic function used by the informed strategies is the transition count, which compares the number of outgoing transitions of a state. The states with the lowest number of outgoing transitions are considered the most promising states, as they may have a higher change of leading to a state without outgoing transitions [29].

5.1.2 Pancake Problem. The next problem is the pancake problem. In this problem, there is a stack of n pancakes, each of a different size, that has to be sorted with the largest pancakes at the bottom. To achieve this, the stack of pancakes can be flipped from any level, resulting in the flipping of the entire stack of pancakes above that level [18].

The pancake problem has a single solution, which is the state in which the stack of pancakes is sorted with the largest pancake at the bottom. Therefore, to evaluate this problem, the search goal is to explore as few states as possible to find this solution.

For this problem the informed strategies use the so-called gap heuristic:

> "Its heuristic value is the number of stack positions for which the pancake at that position is not of adjacent size to the pancake below:" [18]

$$h^{gap}(s) = |\{i \mid i \in \{1, \ldots, n\},\ |s_i - s_{i+1}| > 1\}|$$

5.1.3 Queens Problem. The final problem is the queens problem. In the queens problem, n queens have to be organized on a $n \times n$ chessboard in such a way that no queens attack each other [24]. The problem starts with a queen in the first row of each column, and a solution can be found by moving queens to different rows in their respective columns.

The number of solutions for this problem varies with the board size n. Since this problem will be solved by moving the queens' positions to optimize the layout towards a goal state, the search goal for this problem is limited to finding a single solution. Therefore, the focus will once again be on the number of explored states to find one of the solutions.

The heuristic used for this problem is "the number of pairs of queens that are attacking each other, either directly or indirectly" [24]. By minimizing the number of attacking queens pairs, the goal of a board without any attacking queen pairs may be reached sooner.

5.2 Analysis

To conclude the performance of the flexible framework, the strategies described in Table 3 have been applied to the state space exploration problems described in the previous section. The result of the problems can be found in Tables 4, 5, and 6. With n the n-size problem is denoted, and max shows the theoretical maximum amounts in the entire state space. The values for the random walkthrough strategy are gathered by exploring the state space three times and computing the average, and beam search is used with a frontier size set to 10. The results are analyzed below, organized per problem.

5.2.1 Dining Philosophers Problem. The results of the dining philosopher problem show that every strategy, except for hill climbing, has managed to find both goal states that are present in the state space. Concerning the number of states explored, the three strategies greedy search, A* search, and beam search perform clearly better than the others. These are the informed search strategies, which indicates that the transition count heuristic that is used in this problem is an effective heuristic for this exploration problem.

A* search seems to be the most effective strategy for this problem, resulting in the lowest number of explored states for 6, 8, and 9 philosophers. However, when comparing the results of each strategy between the different problem sizes, beam search does not increase as rapidly compared to other strategies when n increases. This could indicate that for larger state spaces of the dining philosopher problem, beam search possibly becomes the most optimal strategy.

Table 4. Performance Metrics for the n-Dining Philosophers Problem

Metric	**Solutions**				**States Explored**			
n	6	7	8	9	6	7	8	9
Max	**2**	**2**	**2**	**2**	**3 405**	**11 910**	**40 827**	**137 784**
BFS	2	2	2	2	3 405	11 910	40 827	137 784
DFS	2	2	2	2	1 868	5 170	23 230	62 276
RW	2	2	2	2	990	1 757	6 678	19 830
HC	0	0	0	0	6	7	8	9
Greedy	2	2	2	2	130	100	167	327
A*	2	2	2	2	83	114	108	186
Beam	2	2	2	2	88	171	228	329

5.2.2 Pancake Problem. When analyzing the pancake problem, the most important metric is the number of states explored, since there is only one solution to each of the n-pancake problems in this study, and except for hill climbing all strategies managed to find this solution in every exploration.

The three uninformed strategies BFS, DFS and random walkthrough all performed really poorly on this problem, especially compared to the informed strategies. This shows that the pancake problem requires an appropriate heuristic to be solved efficiently, and that the gap heuristic is such heuristic.

Greedy search turned out to be the most optimal strategy for the pancake problem, achieving the best results across all tested n-pancake problems, followed by beam search.

Table 5. Performance Metrics for the n-Pancakes Problem

Metric	Solutions				States Explored			
n	6	7	8	9	6	7	8	9
Max	**1**	**1**	**1**	**1**	**4 332**	**35 282**	**322 562**	**3 265 922**
BFS	1	1	1	1	3 884	29 234	64 642	1 420 616
DFS	1	1	1	1	2 201	5 441	79 429	935 203
RW	1	1	1	1	2 583	23 965	229 517	1 012 268
HC	0	0	0	0	9	3	2	6
Greedy	1	1	1	1	10	13	16	20
A*	1	1	1	1	308	493	119	1 112
Beam	1	1	1	1	58	73	62	96

5.2.3 Queens Problem. The queens problem has led to some interesting results. Firstly, once again, the uninformed strategies performed significantly worse than the informed strategies and especially scaled extremely bad when testing on higher numbers of n, as can be noticed from the results of the 6-queens and 7-queens problem. Therefore, it can be again be concluded that a good heuristic has been selected for this problem.

Secondly, for the 4-queens and 5-queens problem, hill climbing, greedy search, and A* search all managed to find a solution while exploring the minimum number of required states. Whereas, none of them managed to achieve the same result when exploring the two larger 6-queens and 7-queens state spaces. Hill climbing did not even reach a solutions before stopping the exploration, which is a surprising result, as it was expected to be one of the more efficient strategies for exploring large variations of the n-queens problem [24]. This result indicates that with larger variant of this problem, hill climbing reaches local extrema, indicating the need for a more advanced optimization of this strategy [28]. Based on the results of this test, A* search seems to be the best strategy for the n-queens problem, closely followed by greedy search.

Lastly, beam search shows some interesting results in this problem, it managed to find a solution in $(n-1) \times 10 + 1$ explored states, meaning it searches $n-1$ of its frontier size and then finds it. This indicates that a smaller frontier size may be better for this problem.

Table 6. Performance Metrics for the n-Queens Problem

Metric	Solutions				States Explored			
n	4	5	6	7	4	5	6	7
Max	**2**	**10**	**4**	**40**	**3 075**	**62 511**	**1 399 685**	**34 588 847**
BFS	1	1	1	1	1 082	20 242	448 022	10 394 246
DFS	1	1	1	1	1 038	2 317	228 680	9 639 509
RW	1	1	1	1	2 298	3 379	247 411	1 100 055
HC	1	1	0	0	4	5	10	8
Greedy	1	1	1	1	4	5	28	34
A*	1	1	1	1	4	5	10	30
Beam	1	1	1	1	31	41	51	61

6 Conclusion

In this study, the impact of a flexible framework for state space exploration problems was explored. Relevant exploration features were identified and integrated into a feature model diagram that captures the important aspects of a state space exploration strategy.

This led to the development of a flexible framework capable of addressing diverse state space problems in a configurable and adaptable manner. By allowing both the problem and the strategy to be defined through configuration, the framework reduces the need to balance trade-offs between efficiency, optimality, and scalability. Its flexibility allows strategies to be adapted for each unique problem, improving applicability across various domains and problems.

6.1 Discussion

The results of this research demonstrate the potential of a flexible framework for state space exploration problems. However, this flexibility comes with its own trade-offs. The high level of configurable parts leads to a more complex setup process, which requires users to have a solid understanding of the problem and strategies, and may increase the time needed to identify optimal feature selections.

Moreover, the framework focuses on the exploration of state spaces. Another important aspect of solving state space exploration problem is the generation of state spaces. Although the framework provides an interface to allow for the integration of state space generation tools, this remains a challenge for users, who require the knowledge to integrate such tools themselves or access to such tools to provide generated transition system manually.

While the defined set of features covers a broad range of exploration strategies, it may not include every specialized or highly optimized strategy, which may limit its effectiveness in some cases.

Finally, while the framework was tested on several problems, a broader validation across additional domains and problem types is necessary to fully understand its strengths and limitations.

Overall, the flexible framework represents a significant step towards unifying diverse state space exploration strategies under a single configurable framework. However, careful consideration of configuration complexity and performance trade-offs is necessary.

6.2 Future Work

One direction for future work is to expand the number of integrated features. The first step will be to include the parallelization and search direction features. These features were identified in this study but excluded from the implementation of the framework as discussed in Section 4.3.3, but could allow for more advanced and specialized configurations.

Another area of future work involves automated strategy selection, where exploration strategies are configured based on the problem, as investigated in [17]. Moreover, it could focus on the support of dynamic strategies that change during exploration.

Next, the usability of the framework could be improved. Introducing a visual tool or user interface, for example, would make the framework more accessible to a wider range of users.

Finally, evaluating the framework across a broader set of problems and domains would be an important next step. This would provide a deeper understanding of its strengths and limitations.

References

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. 2010. Comparing approaches to implement feature model composition. In *European Conference on Modelling Foundations and Applications*. Springer, 3–19. https://doi.org/10.1007/978-3-642-13595-8_3

[2] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. 2012. Feature model differences. In *Advanced Information Systems Engineering: 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings 24*. Springer, 629–645. https://doi.org/10.1007/978-3-642-31095-9_41

[3] Ahmed M Alaa and Mihaela van der Schaar. 2019. Attentive state-space modeling of disease progression. *Advances in neural information processing systems* 32 (2019).

[4] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information systems* 35, 6 (2010), 615–636. https://doi.org/10.1016/j.is.2010.01.00

[5] Juliana Bowles, Linda Brodo, Roberto Bruni, Moreno Falaschi, Roberta Gori, and Paolo Milazzo. 2024. Enhancing Reaction Systems with Guards for Analysing Comorbidity Treatment Strategies. In *Computational Methods in Systems Biology - 22nd International Conference, CMSB 2024, Pisa, Italy, September 16-18, 2024, Proceedings (Lecture Notes in Computer Science, Vol. 14971)*, Roberta Gori, Paolo Milazzo, and Mirco Tribastone (Eds.). Springer, 27–44. https://doi.org/10.1007/978-3-031-71671-3_3

[6] Linda Brodo, Roberto Bruni, Moreno Falaschi, Roberta Gori, and Paolo Milazzo. 2025. Attractor and Slicing Analysis of a T Cell Differentiation Model Based on Reaction Systems. In *From Data to Models and Back - 11th International Symposium, DataMod 2023, Eindhoven, The Netherlands, November 6-7, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 14618)*. Springer, 69–89. https://doi.org/10.1007/978-3-031-87217-4_4

[7] Jingwei Chen, Robert C Holte, Sandra Zilles, and Nathan R Sturtevant. 2017. Front-to-end bidirectional heuristic search with near-optimal node expansions. *arXiv preprint arXiv:1703.03868* (2017). https://doi.org/10.48550/arXiv.1703.03868

[8] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. 2006. The saturation algorithm for symbolic state-space exploration. *International Journal on Software Tools for Technology Transfer* 8 (2006), 4–25. https://doi.org/10.1007/s10009-005-0188-7

[9] Gianfranco Ciardo, Yang Zhao, and Xiaoqing Jin. 2009. Parallel symbolic state-space exploration is difficult, but what is the alternative? *Electronic Proceedings in Theoretical Computer Science* 14 (2009), 1–17. https://doi.org/10.4204/eptcs.14.1

[10] Sharlee Climer and Weixiong Zhang. 2004. A Linear Search Strategy using Bounds.. In *ICAPS*. 132–141.

[11] Thao Dang, Alexandre Donze, Oded Maler, and Noa Shalev. 2008. Sensitive state-space exploration. In *2008 47th IEEE Conference on Decision and Control*. IEEE, 4049–4054. https://doi.org/10.1109/CDC.2008.4739371

[12] Matthew B. Dwyer, Sebastian Elbaum, Suzette Person, and Rahul Purandare. 2007. Parallel Randomized State-Space Search. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 3–12. https://doi.org/10.1109/ICSE.2007.62

[13] Stefan Edelkamp, Shahid Jabbar, and Alberto Lluch-Lafuente. 2006. Heuristic Search for the Analysis of Graph Transition Systems. In *Graph Transformations*. Springer, 414–429. https://doi.org/10.1007/11841883_29

[14] Jonathan Ezekiel and Gerald Lüttgen. 2008. Measuring and evaluating parallel state-space exploration algorithms. *Electronic Notes in Theoretical Computer Science* 198, 1 (2008), 47–61. https://doi.org/10.1016/j.entcs.2007.10.020

[15] Malay K. Ganai, Chao Wang, and Weihong Li. 2010. Efficient state space exploration: Interleaving stateless and state-based model checking. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 786–793. https://doi.org/10.1109/ICCAD.2010.5653863

[16] Orna Grumberg, EM Clarke, and D Peled. 1999. Model checking. In *International Conference on Foundations of Software Technology and Theoretical Computer Science; Springer: Berlin/Heidelberg, Germany*. https://www.cs.cmu.edu/afs/cs/user/emc/www/papers/Books%20and%20Edited%20Volumes/Model%20Checking.pdf

[17] A.R. Heijblom. 2016. Using features of models to improve state space exploration. http://essay.utwente.nl/71574/

[18] Malte Helmert. 2010. Landmark Heuristics for the Pancake Problem.. In *Symposium on Combinatorial Search*. 109–110. https://ai.dmi.unibas.ch/papers/helmert-socs2010.pdf

[19] Inhye Kang and Insup Lee. 1994. State minimization for concurrent system analysis based on state space exploration. In *Proceedings of COMPASS'94-1994 IEEE 9th Annual Conference on Computer Assurance*. IEEE, 123–134. https://doi.org/10.1109/CMPASS.1994.318461

[20] M.A. Kattenbelt. 2006. Towards an explicit-state model checking framework. http://essay.utwente.nl/57280/

[21] Robert Arlen Levinson. 1993. *Exploiting the physics of state-space search*. Computer Research Laboratory, University of California, Santa Cruz.

[22] Maharshi J Pathak, Ronit L Patel, and Sonal P Rami. 2018. Comparative analysis of search algorithms. *International Journal of Computer Applications* 179, 50 (2018), 40–43.

[23] Jacob I Rasmussen, Gerd Behrmann, and Kim G Larsen. 2007. Complexity in simplicity: Flexible agent-based state space exploration. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 231–245.

[24] Stuart J. Russell and Peter Norvig. 2010. Artificial Intelligence: A Modern Approach. , 64–160 pages. https://people.engr.tamu.edu/guni/csce625/slides/AI.pdf

[25] Erik Snippe. 2011. Using heuristic search to solve planning problems in GROOVE. In *14th Twente Student Conference on IT, University of Twente*.

[26] Pim van den Broek, Ismênia Galvão, and Joost Noppen. 2010. Merging Feature Models. In *14th International Software Product Line Conference*, Vol. 2. Lancaster University, 83–89. https://research.utwente.nl/en/publications/merging-feature-models

[27] Zhou Weiteng, Han Baoming, Li Dewei, and Zheng Bin. 2013. Improved reversely a star path search algorithm based on the comparison in valuation of shared neighbor nodes. In *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*. IEEE, 161–164. https://doi.org/10.1109/ICICIP.2013.6568060

[28] Christopher Wilt, Jordan Thayer, and Wheeler Ruml. 2010. A comparison of greedy search algorithms. In *proceedings of the international symposium on combinatorial search*, Vol. 1. 129–136. https://doi.org/10.1609/socs.v1i1.18182

[29] Rosa Yousefian, Vahid Rafe, and Mohsen Rahmani. 2014. A heuristic solution for model checking graph transformation systems. *Applied Soft Computing* 24 (2014), 169–180. https://doi.org/10.1016/j.asoc.2014.06.055

[30] Qiang Zheng, Xiaoguang Yin, and Dongxiao Zhang. 2023. State-space modeling for electrochemical performance of Li-ion batteries with physics-informed deep operator networks. *Journal of Energy Storage* 73 (2023), 109244. https://doi.org/10.1016/j.est.2023.109244