ABSTRACT

In this paper we describe SYNERGY, which is a general-purpose AI planning system that is based on the genetic programming paradigm. Rather than reasoning about the planning world, SYNERGY uses selection, mutation, recombination and fitness measure to generate linear plans that solve conjunctive goals. We ran SYNERGY on several domains, and the experimental results show that our planner solves problem instances that are up to two orders of magnitude larger than the ones solved by UCPOP.

KEYWORDS: AI planning, genetic programming, conjunctive goals

INTRODUCTION

Artificial intelligence (AI) planning is known to be an extremely hard problem (see [2]), and it is generally accepted that most non-trivial planning problems are at least NP-complete. In order to cope with the combinatorial explosion of the search problem, AI researchers proposed a wide variety of solutions, from search control rules [3] to hierarchical planning [8] to skeletal planning [5]. More recently, we witnessed the occurrence of the stochastic planners, which trade in the completeness of the planner for the speed up of the search process (e.g., SatPlan [7] or PBR [1] are at least one order of magnitude faster than the classic planning systems).

In this paper we present SYNERGY, which is a general-purpose, stochastic AI planner based on the genetic programming paradigm [9]. Genetic programming (GP) is an automatic programming technique that uses evolution-like operations (e.g., reproduction, mutation, and cross-over) to generate and manipulate computer programs. Rather than reasoning about the planning-world, SYNERGY transforms a given AI planning problem-instance into an equivalent GP problem-instance in which each GP-individual is a complete candidate-plan, and it uses reproduction, mutation, cross-over, and fitness measure to generate linear plans that solve conjunctive goals. SYNERGY has an expressive power equivalent to UCPOP: it provides conditional effects, disjunctive preconditions, and both universal and existential quantifiers. We tested SYNERGY on several domains, and our preliminary results show that it solves problem instances that are one to two orders of magnitude larger than the ones solved by UCPOP.
GENETIC PROGRAMMING

Genetic programming represents a generalization of genetic algorithms in the sense that GP manipulates populations of computer programs. GP problem solvers (GPPS) can be seen as two-phase processes. First, they randomly generate an initial generation of programs that are expressed as function compositions. Second, GPPS use the current generation of programs to produce a new generation that is based on the principle of "survival of the fittest". In order to create the new generation, GPPS apply two main breeding operations: reproduction (a program from generation \( i \) is copied unchanged within generation \( i+1 \)) and cross-over (given two parent-programs from generation \( i \), each of them is broken in two components, and generation \( i+1 \) will contain two children-programs that are created by combining components coming from different parents).

A GP application has two main components: a set of building-blocks that is used to create the population of programs, and an evaluation function that measures the fitness of each individual program. In GP, there are two types of building-blocks: terminals and functions. Both can be seen as LISP functions, the only difference between them consisting of the fact that terminals are not allowed to take arguments, while functions take at least one argument. The individuals generated by the GPPS represent computer programs that are built by function composition over the set of terminals and functions.

GPPS use domain-specific fitness evaluation functions: they take as input a program \( P \), and their output represents a measure of how appropriate \( P \) is to solve the problem at hand. In order to estimate the fitness of a program, the evaluation function uses fitness cases of the form \(<\text{input-values}, \text{desired-output-values}>\): for the given input-values, a "perfectly fit" program should generate the desired-output-values.

AI PLANNING AS GENETIC PROGRAMMING

In order to solve an AI planning problem, SYNERGY uses an approach similar to the one described in [7]: it converts the given problem-instance \( A \) to an equivalent problem-instance \( B \) of a different nature, it solves \( B \) based on a stochastic algorithm, and it converts the result of \( B \) to a solution for \( A \). But while SATPLAN turns the planning problem into a SAT problem, SYNERGY converts it into a GP problem (see Figure 1).

![Figure 1. Overview of the SYNERGY System](image)

Most AI planners define the planning actions in a declarative, STRIPS-like manner [4]. In contrast, SYNERGY requires a procedural description of the planning operators. The
procedural approach was motivated by the use of the underlying GP problem solver that
must simulate the execution of each plan in order to evaluate its fitness.

The SYNERGY approach to AI planning has three main advantages. First, SYNERGY
is a highly parallelizable planner because the fitness evaluation of different plans can be
performed in parallel on different processors. Second, SYNERGY facilitates planning in
dynamic environments: as each plan has to be executed in simulation, it is easy to update
the planning-universe after the execution of each operation. Third, the planner provides a
flexible way to express goal priorities. Domain implementers can use weight-factors to
express the relative importance of each goal type, and, consequently, plans that solve a
larger number of higher-importance goals will tend to have a better fitness value.

In order to transform an AI planning problem-instance to an equivalent GP problem-
instance, SYNERGY has to generate the four main components of any GP problem-
instance: the fitness evaluation function, the fitness cases, and the sets of terminals and
functions. For a given AI planning problem-instance, SYNERGY creates a unique
fitness case that consists of two components: the initial world description and the goal
state. When SYNERGY evaluates the fitness of a plan \( P \), which is a linear sequence of
fully instantiatiated operators \(<p_1, p_2, \ldots, p_n>\), the planner sets the world status to the
given initial state, and it successively simulates the execution of each operator \( p_i \) in the plan
(NOTE: as the planner considers each GP individual to be a complete candidate-plan, in
order to avoid the occurrence of invalid plans SYNERGY makes the following
convention: each operator that has a failing precondition is considered to be a no-op).

After executing the plan \( P \), SYNERGY computes the fitness of \( P \) based on the formula

\[
\sum_{i=1}^{n} \text{FitnessEvaluationFunction}_{\text{Predicate}_i}(\text{CurrentState}, \text{Goals})
\]

As one can see from the formula above, each SYNERGY predicate must have a
corresponding fitness evaluation function. A trivial example would be the number of
unsatisfied goals, while more sophisticated functions could use heuristics like the
Manhattan distance, or weight-factors that express the importance of each type of goal.

SYNERGY must also generate the sets of terminals \( T \) and functions \( F \) that are used by
the GP problem solver. Both sets consist of two disjoint components: a user-defined one
and an additional one.

\[
T = T_U \cup T_A
\]

\[
F = F_U \cup F_A
\]

The set \( T_A \) is used to designate the planning objects and will be discussed in the next
section, while \( F_A \) is independent of the planning domain and contains the GP functions
\text{SEQ-2} and \text{SEQ-5} (i.e., \text{PROGN}-like Lisp functions that take 2, respectively 5 arguments).

The sets \( T_U \) and \( F_U \) are generated based on the user-defined planning operators:
operators that take no parameters will be transformed into user-defined terminals, while
the other ones will be turned into user-defined functions. For example, in Figure 2 we
show how the UCPOP drop operator can be transformed into an equivalent GP function.
Each precondition is tested by using the is-fact function, while the effects are
produced by using the add-fact and delete-fact functions. Finally, the parameter
typing is enforced based on the cast function, which will be analyzed in the next section.
STRONGLY-TYPED PLANNING OPERATORS

SYNERGY provides a domain-independent solution to a major problem related to the different natures of AI planning and GP. On one hand, planning operators are strongly-typed in the sense that each argument of an operator must be of a well-defined, pre-established type. On the other hand, GP functions are typeless and rely on the closure property [9], which ensures that any value returned by a function or terminal represents a valid argument for any function in the function set.

It is important to note that other authors who solved AI planning problems based on the GP paradigm [6, 9, 11] did not have to solve the strongly-typed problem (STP) because of the particular nature of the planning problems considered. For instance, Handley solves the robot navigation problem by defining a GP encoding that uses only terminals. Similarly, the versions of the blocks world problem solved in [9, 11] deal with a single domain concept (i.e., the BLOCK concept), and, consequently, there is no need to verify the type of the objects handled by the operators.

A naive solution to STP would consist of creating an additional terminal for each planning object and including the type-checkings in each operator’s preconditions. However, such an approach is not acceptable for practical reasons: if the problem instance defines a non-trivial number of objects, the vast majority of the operators will become no-ops because of the failing type-checkings. Consequently, the search for the GP solution will be dramatically slowed by the inefficiency of the problem encoding.

In order to solve STP, we created an efficient, general-purpose mechanism based on the analysis of each problem-instance to be solved. Let us suppose that SYNERGY has to solve an instance \( P_I \) of the planning problem \( P \). Let us also assume that \( P \) defines a set of \( n \) concepts \( C = \{C_1, C_2, ..., C_n\} \), and \( P_I \) creates for each concept \( C_i \) a number of \( N_i \) instances. For \( P_I \), SYNERGY will create the set \( T_A = \{T_1, T_2, ..., T_m\} \), where \( m \) is the least common multiple of \( N_1, N_2, ..., N_n \). Each additional terminal \( T_i \) is a Lisp function that simply returns the integer value \( i-1 \). Based on the values returned by the terminals \( T_i \), SYNERGY provides the function \( \text{CAST} \), which is used by the planning operators to convert an incoming argument to an object of any desired type. For instance, the function invocation \( \text{CAST} T_j C_i \) converts the value \( j-1 \) returned by \( T_j \) into an object of type \( C_i \) by returning the \( k \)-th instance of the concept \( C_i \), where \( k = (j-1) \mod N_i \).
We must emphasize that the function \texttt{CAST} provides a \textit{uniform} mapping of the additional terminals to objects: for any concept \( C_i \), each of its instances can be obtained via \texttt{CAST} from the \textit{same} number of distinct terminals. The uniformity of the mapping is a key factor in the creation of unbiased plans because a non-uniform \texttt{CAST} mapping would lead to an unbalanced use of the planning objects (i.e., SYNERGY would be more likely to use the objects that are pointed-at by a larger number of terminals from \( T_A \)).

\section*{EXPERIMENTAL RESULTS}

In order to test SYNERGY's capabilities, we ran it on several problems: the robot navigation (RNP), the blocks-world (BWP), and the briefcase problem (BP). In \textbf{TABLE I} we present two instances of each of the three problems: RMP-1, BWP-1, and BP-1 are the easiest instances that UCPOP fails to solve, while the other three are the hardest instances solved by SYNERGY. As we can see, for each domain, SYNERGY solves instances at least one order of magnitude harder than UCPOP.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Description</th>
<th>First solution found at generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNP-1</td>
<td>4x4 table, 6 obstacles (requires at least 4 \texttt{PUSH}-es)</td>
<td>1</td>
</tr>
<tr>
<td>RNP-2</td>
<td>100x100 table, 18 obstacles (requires at least 6 \texttt{PUSH}-es)</td>
<td>354</td>
</tr>
<tr>
<td>BWP-1</td>
<td>8 blocks, 1 table (1 stack to 3 stacks)</td>
<td>39</td>
</tr>
<tr>
<td>BWP-2</td>
<td>20 blocks, 1 table (1 stack to 2 stacks)</td>
<td>25</td>
</tr>
<tr>
<td>BP-1</td>
<td>4 objects, 5 locations, 1 briefcase</td>
<td>24</td>
</tr>
<tr>
<td>BP-2</td>
<td>10 objects, 10 locations, 10 briefcases</td>
<td>100</td>
</tr>
</tbody>
</table>

The variant of RNP solved by SYNERGY is defined as follows: given a rectangular table with \( k \) blocks located in its grid-cells, a robot \( R \) must navigate from its current position \( CP \) to the desired position \( DP \). In order to reach \( DP \), the robot can use two types of operators: \texttt{MOVE} and \texttt{PUSH}. Both \texttt{MOVE} and \texttt{PUSH} can be executed in four directions (north, south, east, west), and the robot can push exactly one block at a time.

Based on the experimental results, we can make several important observations. First, SYNERGY solved problem instances that are \textit{two orders of magnitudes} larger than the ones solved by UCPOP. Second, SYNERGY is capable of creating better plans once it finds a first solution. For example, SYNERGY found a first solution for RNP-2 at generation 354 (it had 293 operations, while the optimal plan required only 206 actions), but by the time it reached generation 867, SYNERGY was able to deliver a solution with only 216 operations. Finally, SYNERGY is especially well fit to solve hard problem instances: while on easy instances UCPOP is faster than SYNERGY, the GP-based approach is more appropriate for problem instances that have a higher level of difficulty.

As we tried to test SYNERGY on complex problems, we generalized the versions of BP and BWP that are used by UCPOP: instead of being restricted to a single briefcase, respectively a single table, we allow the use of several briefcases, respectively tables. As expected, the results are somehow similar to the ones obtained for the RNP domain: on easy instances UCPOP solves the problem faster than SYNERGY, but on complex problem instances SYNERGY is still capable of solving the problem, while UCPOP is unable to cope with the increased level of difficulty.
FUTURE WORK AND CONCLUSIONS

We plan to extend SYNERGY by adding two major features. First, by introducing *hierarchical planning operators*, SYNERGY would allow users to create several levels of abstraction that will reduce the search space. For instance, if we consider a combination of the BP and RNP, at a higher level of abstraction each robot could perform the three BP operations (*put-in*, *take-out* and *move-to*), and SYNERGY would not be concerned with any navigation details. Once the planner finds a solution at the BP-level, it translates each fully-instantiated *(move robot-1 location-1)* operator into a goal *(at robot-1 x1 y1)* that must be solved at the RNP-level. The use of hierarchical operators might be extremely beneficial for domains like the one presented above, in which achieving the goal *(at object-1 100 13)* typically involves a long sequence of *move* and *push* operators, followed by a single *take-out* action.

Second, we would like to allow users to define planning problems that involve dynamic environments. For instance, in the RNP, we could define mobile obstacles, which would be continuously changing their positions based on predefined trajectories. As candidate-plans are executed in simulation, after performing each planning operation SYNERGY could update the position of the mobile obstacles based on the predefined trajectory functions.

The major contribution of this paper consists of providing a domain-independent mapping of any AI planning problem-instance into an equivalent GP problem-instance. In final analysis, we can conclude that SYNERGY is a general-purpose AI planning system that is capable of solving large, complex problem instances. By supporting disjunctive preconditions, conditional effects, and existential and universal quantifiers, SYNERGY has an expressive power equivalent to UCPOP, and our initial results show that SYNERGY outperforms UCPOP on all the difficult examples it was tested on.

REFERENCES