Investigation of Different Seeding Strategies in a Genetic Planner

C Henrik Westerberg[1] and John Levine[1]

AI Applications Institut, University of Edinburgh,
80 South Bridge, Edinburgh, EH1 1HN
{carlw,johnl}@dai.ed.ac.uk
http://www.dai.ed.ac.uk/homes/carlw/

Abstract. Planning is a difficult and fundamental problem of AI. An alternative solution to traditional planning techniques is to apply Genetic Programming. As a program is similar to a plan a Genetic Planner can be constructed that evolves plans to the plan solution. One of the stages of the Genetic Programming algorithm is the initial population seeding stage. We present five alternatives to simple random selection based on simple search. We found that some of these strategies did improve the initial population, and the efficiency of the Genetic Planner over simple random selection of actions.

1 Introduction

This paper presents an investigation into different seeding strategies for initial population formation. This work is performed in the context of Genetic Programming (GP) applied to classical planning. An already implemented Genetic Planner [13, 14] was altered to use the different seeding strategies. We then experimented with the different strategies to determine their effect on the GP algorithm in terms of CPU time used to solve the problem, the quality of the initial population, and the number of generations required to solve the problem.

There are many areas in the GP algorithm which can be improved, such as population seeding, crossover, and the fitness function. We started at the beginning and wanted to improve the initial population seeding stage. The efficiency of the GP algorithm could be improved by increasing the quality of the initial population. This can be done by adopting simple rules or doing some limited search to decide which non-terminal should come next in the candidate. An easy example, from timetabling, would be add the easy time-course relationships into the initial population. We used simple search strategies like Depth First Search (DFS) and Best First Search (BFS)to improve the quality of the initial population.

This paper will first discuss the available work on Genetic Planning and seeding. The subsequent section gives a quick overview of planning and the types of planning problems the Genetic Planner caters for. This is followed by a description of how the GP algorithm was implemented a description of some specific algorithmic features. The next section describes the different seeding strategies which were implemented and results from experiments done using those strategies. Finally the last section discusses some conclusions and possibilities for further work.

2 Current Work

There are three papers that relate closest to the idea of domain independent planning by evolutionary means. Spector looked at using GP for simple planning problems in the blocks domain and also using GP to come up with general which work on any problem [10]. Muslea looked at the effectiveness of using GP for domain independent planning [9]. Westerberg summarises some work done studying the effectiveness of using GP on classical planning domains [13, 14].

There is also other related work to Genetic Planning. Handley looked at GP applied to robot control planning [6]. Aler has looked at generating heuristics for planning [2] and generating control rules for planning problems [1]. Finally, Koza looked at generating a general plan for a blocks stacking problem [7].

Work on population seeding for evolutionary methods is equally sparse. One example of a positive result from improving the seeding algorithm is Langon's work [8]. In Langdon's work the author starts by seeding the population with perfect individuals and then asks the GP algorithm to generalise them. The author compared this method with a random seeding strategy. The population based on perfect individuals was able to generalise better than the random method. This is an indication of the fact that there are benefits to improving the initial population seeding method rather than resorting to a simple random method.

3 Style of Planning

Planning is the task of deciding on which sequence of actions will best achieve some goal or set of goals. In general a planning system is given the world, represented by a set of facts, a set of descriptions of actions the planner can choose from, and goals that need to become true or remain true. Given this information the planner must then find a plan that when executed will achieve the goal or set of goals. There is interaction between actions in that some actions have to occur before others. For example if someone wants to buy some bread they would have to go to a shop before trying to purchase the bread. There are a large variety of planning problems depending on how the actions, goals, and world state are described.

The Genetic Planner concentrates on a style of planning called Classical Planning. One form of classical planning is called STRIPS planning [5]. This style of planning gives us the rules on how to set out a planning domain. The world state is made up of a list of facts that are true about the world. Using the Briefcase Domain as an example, also used by Muslea [9], facts look like: `at(papers,london)`, `at(mybriefcase,london)`. STRIPS planning also tells us how we should design our operators. For example, one of the planning operators from the Briefcase Domain is `putin`:

```
Operator: putin(Object,Briefcase)
Preconditions: at(Object,Location),
               at(Briefcase,Location)
Add List: in-briefcase(Object,Briefcase)
Delete List: at(Object,Location)
```

We give the operator's name and what types of objects the operator takes. The preconditions are those facts that need to be present in the world state before the action can execute. If the action executes the facts in the add list get added to the world state and the those facts in the delete list are deleted from the world state. Those facts which slowly get added and deleted from the knowledge base should hopefully push the world state towards the goal state.

Given the following initial state and goal state we can come up with a plan using our operators to transform the initial state to the goal state.

```
Initial State                  Goal State
at(mypapers,london)            at(mypaper,paris)
at(mybriefcase,london)         at(mybriefcase,paris)
```

The plan shown in Fig.1 solves the above problem:

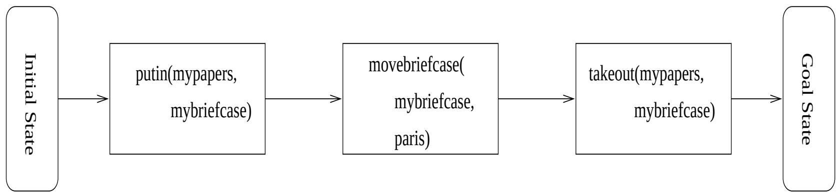

Fig. 1. An example plan/chromosome

The first step is to put the papers in the briefcase, the second step is to move the briefcase to paris. The last step is to take the papers out of the briefcase. The task of the Genetic Planner is to solve this style of planning problems. Finding solutions to these problems is quite difficult but we are more often interested in finding an optimal plan. For a domain independent system this is known to be NP-hard for most cases [4].

4 Genetic Planning

Genetic Planning is the application of GP to planning. An important difference between this and normal GP is that Genetic Planning works on plans as opposed to programs. Programs and plans are analogous as they can both be considered as an ordered set of instructions. The Genetic Planner starts by producing an initial population of candidates according to some strategy. The Genetic Planner then evolves the population of plans using survival of the fittest in order to produce a plan that will achieve some set of goals.

The Genetic Planner, reported here uses a generational algorithm with tournament selection, a domain independent fitness function, 1-point crossover, reproduction, and an addition mutation.

To highlight the other major differences between Genetic Planning and GP we will consider three specific algorithmic details.

4.1 Representing a Candidate/Chromosome

We decided to represent candidates using a linear genome rather than a tree genome [3]. This decision resulted in a considerable speed up and simplification of the system [13]. The linear genome is a linear list of planning operators and their arguments, much like Fig. 1. Each planning operator and associated arguments makes up an atomic action. Each candidate is made up of these atomic actions. In a GP context the planning operators from a particular problem domain are the non-terminals and the objects those actions take are the terminals. Each planning operator also takes arguments of a particular type. For example the `putin` operator from the Briefcase Domain takes an object and a briefcase. Those terminals that belong in each of these sets is already predefined so that no non-working actions are created due to having arguments of the wrong type in the operator. Also crossover in the system works at the action level so that actions are always being swapped with other actions. Crossover points can never occur in the middle of the arguments to an action. The atomic structure of the actions is thus preserved.

4.2 Creating a Candidate/Chromosome

The first strategy we implemented for creating new candidates for the initial population was the random method (Strategy 0). First the candidate is constrained to a specific length. For example, we may want to seed the population with candidates of length ten actions. For each action one of the available non-terminals is selected, and arguments for that non-terminal are selected. The non-terminal and the arguments are selected at random and the arguments are selected from sets of the correct type. The non-terminal and its arguments make up a single planning action. The strategy then adds the created action to the candidate and continues creating new actions until the maximum length of the candidate is reached.

4.3 Finding the Fitness

A difference between GP and Genetic Planning is that GP operates on a multitude of fitness cases and in Genetic Planning there is only one fitness case for each problem. Before the fitness function can assess a particular candidate the candidate must first be simulated. The simulation function starts with the initial state of the problem and sequentially starts applying the actions in the candidate to the initial state. For each action the preconditions of that action are checked against the current state. If the preconditions are satisfied that action is allowed to execute by updating the world state and the next action is considered. If the preconditions are not present in the current state then that action is ignored and the simulator moves to the next action. The current state is carried over to the next action untouched. The resulting state from applying all the actions in the candidate is then used as input for the fitness function. The fitness function works by comparing the input state with the goal state.

The fitness value in the Genetic Planner is made up of two parts. The first and main part considers how many facts from the goal state are present in the resulting state. This forms a proportion of number of achieved goals over number of actual goals. A small part of the fitness value comes from the number of working actions in the candidate. Currently 80% of the fitness value is given by the correctness of the plan, and the remaining 20% is given by the proportion of working actions in the plan.

5 Seeding Strategies

We now present the different initial population seeding strategies we considered. The intuition that prompted this work was to provide the Genetic Planner with a higher quality initial population to evolve from and therefore reducing the evolution time. As the order of actions is important in planning, we should have a seeding strategy that takes this ordering into account somehow, but the random strategy ignores this completely. Hence, we developed some seeding strategies based on different search methods. Each of the strategies works at the candidate level, and each candidate is set to a particular initial length.

5.1 Strategy 1: All Action Method

The first seeding strategy is a modification of the random one. In this case all possible actions for a particular domain-problem pairing are created and stored in a list. For a new candidate the strategy then selects actions randomly from the list and adds those actions to the new candidate. Each time an action is selected it is removed from the possible list of actions and random selection takes place on this smaller list. This continues until the list is empty, at this stage the list is reset. This method ensures that all the actions are present in the initial population and are present with almost equal diversity.

5.2 Strategy 2: DFS

The second new seeding strategy was one based on DFS. In this case the strategy starts with the initial state and the set of all actions. The strategy then searches for an action that can execute from the current state. Once an action is then added to the candidate. The action is applied to the current state to produce a new state. The strategy then proceeds to work from this new state. The strategy continues adding actions until the maximum initial size for plans is reached.

5.3 Strategy 3: DFS + Random Actions

A variation to the second seeding strategy is to force it to occasionally select a random action. In this instance at a particular state the strategy is either going to select an action at random from the set of all actions or the strategy is going to search for an action which executes. The probability of selecting either is set by a parameter. This strategy was added in case Strategy 2 produces initial populations which are not diverse enough.

5.4 Strategy 4: DFS from Random States + Random Actions

A second variation to the DFS strategy would be to alter the initial state the strategy did DFS from. This strategy works by creating a random initial state and then working like strategy number 3 from that initial state. Once a candidate has been produced a new initial state is created and the strategy proceeds. This would increase the variation of the initial population but would also give strings of applicable actions from all over the search space.

5.5 Strategy 5: BFS + Random Actions

The final strategy is based on Best First Search. This strategy starts with the initial state and the set of all possible actions. Using the initial state as input BFS strategy attempts to find the best action to apply from that state. Therefore it applies all actions to the state and records the fitness for each of the states. The action that does best is then added to the candidate. The state is updated by applying that action and the search continues. If more than one action return the same best fitness then an action is chosen randomly from the set of best actions. The final alteration to make is that like Strategy 2, this strategy occasionally picks random actions rather than the best action.

6 Experiments

Each experiment was conducted using the parameters described in Table 1, unless otherwise indicated.

One large experiment was done on all the Blocks World problems available to the planner for the Blocks World Domain. We reproduce results for the four hardest problems called Large A to Large D. The problem specifications were taken from the BlackBox distribution [15]. We used Spector's `puton` and `newtower` actions thereby shrinking the plan length by half when compared with plans

Table 1. Experimental Setup

Parameter	Setting
Termination	Maximum number of generations 100
Pop Size	2000
Initial Length	15-30 actions
Tournament Size	4
Max plan size	200 actions
Genetic op. prop	80% crossover and 20% reproduction
Addition Mutation	Applied 10% of the time and adds 8 random actions to random positions
Seeding Strategy	All, with 50% random actions for 3,4, and 5
Fitness	As described in Section 4.3.

created using the BlackBox distribution. This is due to the Blocks World Domain implemented for BlackBox explicitly modelling the robot arm, thus the their domain containing four planning operators.

Each problem is an extension of the predecessor by adding more blocks to the problem. Large A contains 9 blocks with a shortest plan of 6 actions, up to Large D, which contains 19 blocks with a shortest plan of 18 actions. For each of the problems, each strategy was used, and for each strategy there were ten runs.

6.1 Results

The following four tables report the results of the experimentation. Table 2 shows the average fitness of the initial populations and average value of the best individual in parenthesis. Table 3 shows the average number of generations used to solve each problem. The number in parenthesis for Table 3 indicates the number of failures. No parentheses indicates that there were no failures. This average also includes those runs that went to a hundred generations and failed to find a solution. A dash indicates that none of the ten runs were successful. Table 4 shows the average CPU time in seconds for that particular problem. This time includes population initialisation time and at the moment there is no accurate means to separate the two. This time was computed by dividing the total CPU time taken for the ten runs divided by the number of successful runs. Table 5 shows the results of varying the proportion of randomly selected actions for the Large B problem and using 10 runs for each proportion.

7 Discussion

The results from using Strategy 0 are the benchmark results. The first refinement, Strategy 1, reduced the amount of time and generations needed, but had little impact on the last two problems. An interesting point is that though initial

Table 2. Average and Best Fitness of Initial Populations

Strategy	Large A	Large B	Large C	Large D
0	32.1 (60.7)	26.1 (49.6)	10.0 (29.7)	9.8 (22.6)
1	26.4 (60.7)	21.9 (46.1)	9.4 (18.2)	8.0 (16.8)
2	26.3 (62.8)	23.8 (53.8)	13.3 (42.0)	11.6 (28.5)
3	27.4 (61.5)	24.8 (52.5)	12.1 (36.6)	11.0 (27.1)
4	26.9 (57.5)	22.1 (46.3)	9.4 (22.6)	8.2 (18.9)
5	57.2 (100.0)	51.3 (90.1)	30.5 (72.7)	29.3 (74.5)

Table 3. Generations used to Solve Problem

Strategy	Large A	Large B	Large C	Large D
0	7.9	16.3	80.8 (5)	-
1	6.7	15.4	76.7 (6)	-
2	4.6	9.4	44.7	89.5 (7)
3	5.0	9.8	51.5 (2)	-
4	7.2	16.0	75.0 (3)	-
5	0.0	2.9	8.1	5.5

Table 4. Time Taken to Solve Problem

Strategy	Large A	Large B	Large C	Large D
0	51.8	197.4	3862.4	-
1	41.9	168.8	4492.9	-
2	53.3	112.1	1064.7	8834.3
3	43.9	97.0	1484.5	-
4	64.8	194.8	2614.8	-
5	302.9	557.5	1459.8	3176.6

Table 5. Varying the Proportion of Random Actions for Strategy 5, for Large B

Proportion Random	Average Generations
0.1	20.6
0.2	13.4
0.3	5.2
0.4	0.0
0.5	2.0
0.6	3.9
0.7	3.8
0.8	6.1
0.9	11.8

population looks slightly worse, Strategy 1 does better than Strategy 0. Strategy 2, using DFS, allowed for significantly less time and generations when compared with Strategy 0. Also notable is that Large D was solvable using this strategy. Strategy 3 performs slightly worse than Strategy 2. Both 2 and 3 produced initial populations with good fitness values. Strategy 4 resulted in being equivalent to Strategy 0 in terms of number of generations, time taken, and quality of initial population.

Perhaps the most interesting result is Strategy 5, based on BFS. Using this seeding strategy, most of the work goes into seeding the population, about 80% of the CPU time. This strategy used the fewest generations to solve each of the problems. Although it took longer to solve the simpler problems, the rate at which the time taken to solve a particular problem is increasing at a slower rate than the other seeding strategy methods. This last strategy also produced by far the best individuals for the initial population.

Table 5 confirms that a 50/50 proportion of random actions and searched actions was a good idea. In fact, having nearly no random actions gave performance which was worse than the random strategy. But having initial candidates made of 40% random actions and 60% best actions gave the least number of generations for Large B.

8 Conclusions

We believed that the efficiency of the Genetic Planner can be improved by improving the quality of the initial population. We also had some intuitions about what features a good initial population should have. There should be many working actions in the initial populations. The order of the actions is important in planning so it's important that this is reflected in the initial population. Also the initial population should be diverse in that it includes all the actions. We feel that some of these intuitions have been weakened. The strategies that worked best, strategies 2 and 5, produced the least diverse populations but performed well. This is probably because the order of the actions is more useful than having a diverse population. Strategy 4 would satisfy all three requirements, but performed only as well as a random method. Perhaps this is because the ordering was completely lost. However the final table indicates that some randomness in the initial population is clearly beneficial. When this combined with an initialisation strategy that gives the best ordering of actions, the strategy produces the best result for Large B. Clearly, however more statistical methods and experimental work has to be carried out before any of these conclusions can be confirmed.

9 Acknowledgements

The first author would like to thank the EPSRC for support via a quota studentship.

References

1. Aler, R., Borrajo, D., Isasi, P.: Genetic Programming and deductive-inductive learning: A multistrategy approach. Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98, Madison Wisconsin, (1998) 10-18
2. Aler, R., Borrajo, D., Isasi, P.: GP Fitness Functions to Evolve Heuristics for Planning. Proceeding of the GECCO 2000 Conference, EvoPlan Workshop (2000)
3. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming An Introduction. Morgan Kaufmann Publishers, San Francisco CA (1998)
4. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence, 69(1-2):165-204, (1994)
5. Fikes, R.E., Nilsson, N.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2 (1971) 189-208
6. Handley, S.G.: The automatic generations of plans for a mobile robot via genetic programming with automatically defined functions. In Advances in Genetic Programming, MIT Press (1994) 391-407
7. Koza, J.R.: Genetic Programming. The MIT Press, Cambridge MA (1992)
8. Langdon, W.B., Nordin, J.P.: Seeding Genetic Programming Populations. Proceedings of the Third European Conference on Genetic Programming, EURO GP 2000, Edinburgh (2000)
9. Muslea, I.: SINERGY: A Linear Planner Based on Genetic Programming. Proceedings of the 4th European Conference on Planning, (1997)
10. Spector, L.: Genetic Programming and AI Planning Systems. Proceedings of Twelfth National Conference on Artificial Intelligence, Seattle Washington USA AAAI Press/MIT Press (1994)
11. Tate, A., Hendler, J., Drummond, M.: A Review of AI Planning Techniques. Readings in Planning. Morgan Kaufmann Publishers, San Mateo CA (1990)
12. Weld, S.: An Introduction to least commitment planning. AI Magazine 15(4), (1994)
13. Westerberg, C.H.: An investigation into the use of using Genetic Programming to solve Classical Planning Problems. 4th Year CS/AI Project, Division of Informatics, University of Edinburgh, (2000)
14. Westerberg, C.H., Levine, J.: "GenPlan": Combining Genetic Programming and Planning. Proceedings for the UK Planning and Scheduling Special Interest Group 2000, Milton Keynes UK (2000)
15. Blackbox: http://www.cs.washington.edu/homes/kautz/blackbox/index.html