Optimising Plans using Genetic Programming

C Henrik Westerberg
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
carlw@dai.ed.ac.uk

John Levine
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
johnl@aiai.ed.ac.uk

Abstract

Finding the shortest plan for a given planning problem is extremely hard. We present a domain independent approach for plan optimisation based on Genetic Programming. The algorithm is seeded with correct plans created by hand-encoded heuristic policy sets. The plans are very unlikely to be optimal but are created quickly. The sub-optimal plans are then evolved using a generational algorithm towards the optimal plan. We present initial results from Blocks World and the Briefcase Domain and found that GP method almost always improved sub-optimal plans, often drastically.

1 Introduction

Finding any plan for planning domains is often a difficult task, but we are often more interested in the even harder task of finding optimal or near optimal plans. The current fastest planning systems use heuristics and hill-climbing techniques. However, no heuristic is perfect and plans found in this way are often sub-optimal, in the sense they use more actions to achieve the goal state than are necessary.

We present a domain independent technique, based on Genetic Programming (GP) that attempts to optimise linear plans. The system accepts a seed of plans from which to optimise. This seed could be produced by a current planning system or plans made using heuristics. The amount of computational effort to devote to the optimisation stage can also be set by the user by setting various parameters of the system. The GP algorithm also has *anytime* behaviour, and could return the best current plan at any time during the run.

Using the Genetic Planning optimisation system, we experimented on two domains: Blocks World Domain, and the Briefcase Domain [6]. The Blocks World problems were kindly donated to us Jose Ambité. During the experimentation we were looking for how much the initial plans could be shrunk depending on the type of heuristics used, the behaviour of the system as it operated, and what changes we could make to the current system to improve its ability.

2 Plan Optimisation via Genetic Programming

We present here one possible implementation of using Genetic Programming as a linear plan optimiser. We used two different hand-encoded *policy sets* for the each domain in order to seed the initial population with correct but overly long plans. We then used a generational algorithm with standard genetic operators in order to optimise those plans [2]. We based our work on a previously implemented generational algorithm for linear planning [8].

The following implementational details have many alternatives, and are not fixed. One of the strengths of our approach is that the Fitness Function and Simulation stage can be altered to look for different cost aspects besides the simplistic plan length.

Plan Representation: Plans are represented as linear lists of sequential, instantiated, atomic actions. Each atomic action contains one operator and its arguments.

Simulation: The simulation stage takes an individual or plan and then attempts to apply all the actions. During the simulation stage various attributes of the plan can be recorded such as how many actions there are in the plan and what effect the plan had on the initial state. This information can then be used as input by the fitness function.

Fitness Function: The fitness function takes the output of the simulation stage and prescribes a fitness value to individual based on the information given to it. In the case of this system the fitness function has two parts. The first part says whether the plan achieves all the goals not. The second part is the number of actions in the plan and is used as a tie-breaker in tournament selection.

2.1 Genetic Operators

There is a large choice of genetic operators to be used during the optimisation stage. We have taken the position of keeping things simple and stochastic. One alternative is to implement domain specific operations, such as rewrite rules, for optimising particular domains.

Crossover: This system uses one-point crossover.

Reproduction: This is the simplest operator and it copies the selected parent into the next population.

Shrink Mutation: This type of mutation simply deletes a randomly selected action from the parent.

Move Mutation: This type of mutation moves a randomly selected action to a new randomly selected position.

Mutations occur on children created by either reproduction or crossover. The probabilities of the operators occuring are set by the user. The implementation presented here is based on an existing system and is by no means optimal for generating optimal plans. Improvements that can be made to it and some are suggested in Section 5.

3 Policy Set Planning

This section describes the hand-encoded policy sets used to seed the initial population for the two domains under consideration.

3.1 Blocks World

The Blocks World Domain is important because it is one of the benchmarking domains used to compare different planners. Blocks is also important historically as one of the original planning problem domains. In addition, finding optimal plans for Blocks World problems is known to be NP-hard [3]. We also chose the Blocks World Domain as a fast domain specific planning algorithm that produces optimal plans exists for it, called BWOPT [7].

Our system uses hand-encoded policy sets to produce entire populations filled with correct but suboptimal plans. A GP optimising system probably works better if the initial populations is diverse. To achieve this the policy sets were interpreted non-deterministically.

The policy sets generally function like this. The rules within each policy set are tested sequentially. For the current rule the current world state is examined and all actions that could operate on that state in accordance with the rule are discovered. At that point, one of the actions is selected randomly and added to the new plan. The current world state is updated and the formation of the plan continues until all goals in the goal state are achieved. If the rule allows for no actions, the next rule in the rule set is used and if one rule fires then the other rules are ignored.

There are several types of policy sets that can characterised by how easy it is to optimise the resulting plan. The three types we are interested in here are:

- **Optimal Policy Sets:** These policy sets always produce optimal plans, for any problem in the domain.
- **$\mathcal{DM}$-Optimal Policy Sets:** These policy sets always produce plans where an optimal plan can be discovered by only deleting and moving actions:
  - $\forall c \in C \rightarrow c^* \in \mathcal{DM}(c)$ where C is the set of all plans constructed by the policy set, c^* is an optimal plan and $\mathcal{DM}(c)$ is the set of all plans which can be created by only moving and deleting actions in c.
- **Satisficing Policy Sets:** These policy sets produce correct plans but may produce plans that are missing actions which the optimal plan would need.

Policy Set 1

1. Discover all actions achieving well placed blocks *or*
2. Find all actions moving movable non-well placed blocks to a new location

Policy Set 2

1. Discover all actions placing movable blocks onto the table *then*
2. Discover all actions achieving well placed blocks

A well placed block is one which no longer has to move, as it is in its target location and all blocks below it are well placed. A movable block is one which is not underneath a block or already on the table. Policy Set 1 does not scale very well for larger problem instances: when the first rule provides no actions, it "wanders" around at random until the first rule starts to succeed. The first policy set belongs in the class of *Satisficing Policy Sets*. The second policy set unstacks all the blocks and then stacks the blocks back up in the right order. This policy set belongs in the class of *$\mathcal{DM}$-Optimal Policy Sets*. This was the policy set Ambité used in PbR [1].

3.2 Briefcase Domain

The Briefcase Domain [6] is an alternative planning domain which can also be described as a classical planning domain. In this domain there are a number of briefcases, objects, and locations. Each object needs to be placed in the correct location. This is done by placing the object in a briefcase and then moving the briefcase to another location and removing the object from briefcase. The Briefcase Domain is the same problem as the air transportation part of the Logistics Domain used at the AIPS 98 and AIPS 2000 competitions.

```
putin(Object,Briefcase)
Pre: at(Object,Location),
     at(Briefcase,Location)
Add: in-briefcase(Object,Briefcase)
Del: at(Object,Location)

takeout(Object)
Pre: in-briefcase(Object,Briefcase)
Add: at(Object,Location)
Del: in-briefcase(Object,Briefcase)

movebriefcase(Briefcase,Location)
Pre: at(Briefcase,Location)
Add: at(Briefcase, Location)
Del: at(Briefcase,OldLocation)
```

The Briefcase Domain is used to here to present some alternative results to the Blocks World Domain. Finding an optimal plan is hard and increases with the inclusion of more objects and locations. Unfortunately, there is no existing program algorithm producing optimal plans for this domain, so we chose problems for which the optimal plan can found by inspection.

Here are the two policy sets we adopted for the Briefcase Domain:

Policy Set 1

1. Find all actions achieving well placed objects *or*
2. Find all valid actions from the current state

Policy Set 2

1. Move a non well placed object to the correct location

Policy Set 1 works in the same fashion as the policy sets for the Blocks Domain. Policy Set 2 is more like a HTN planning schema, in that prescribes all actions necessary to achieve a well placed object. For instance, at most 5 actions are needed, move the briefcase to a non-well placed object, place the object in the briefcase, move the briefcase to the object's target location, remove the object from the briefcase, and move the briefcase back to its original destination.

Like the Blocks World Domain, a well placed object is one which no longer has to move. Again the first policy is meant to be one in which a machine learning system could possibly learn and the second policy a stronger variant. Both produce sub-optimal but working plans. However, the first policy does not scale so well for larger problem instances.

4 Experimental Results

This section gives the experimental results for both the Blocks World and Briefcase Domain.

4.1 Blocks World

Each experiment was done using the parameters shown in Table 1. We performed 25 runs for each problem, and again for each policy set. We experimented using 50 Blocks World problems. During the run we recorded the average number of actions in the first best individual (from the seeding stage) and the average number of actions in the last best individual.[1] Each point on the x-axis represents a single problem. The order of the problems is first by blocks size, and then by average length of the first best individual.

Referring to Figure 1, Policy Set 1 shows significant but not complete improvement in plan length after 1000 generations. An additional termination criterion was implemented, called "no change" which stops a run if there is no change in fitness after X generations. We repeated the experiments setting X to 5000. Taking the 30 block problems as an example, these were shrunk down to the 50 action mark.

[1] CPU times are not considered as the system was implemented using Java, and running on Solaris. System times can be dramatically improved if written for C under Linux.

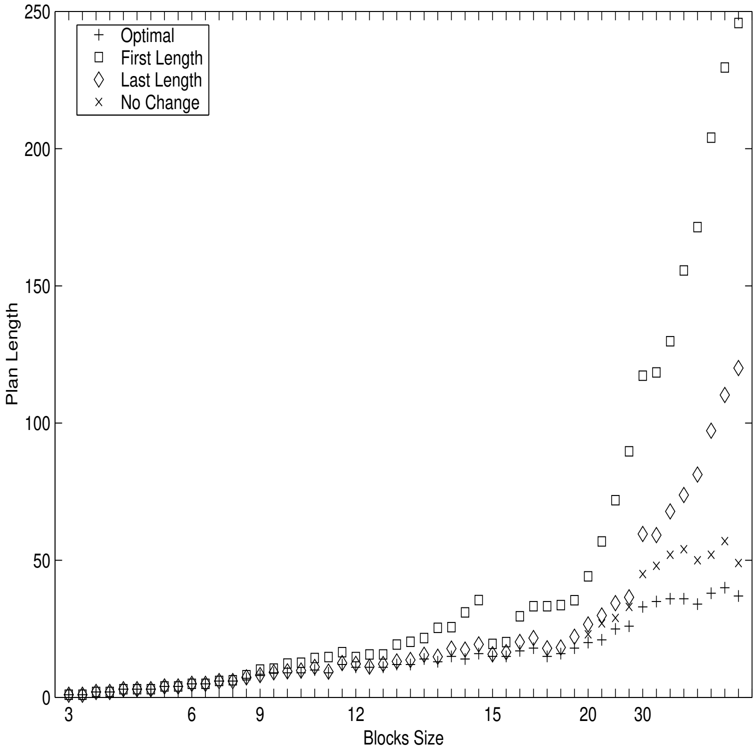

Figure 1: Policy Set 1 on the Blocks World Problems

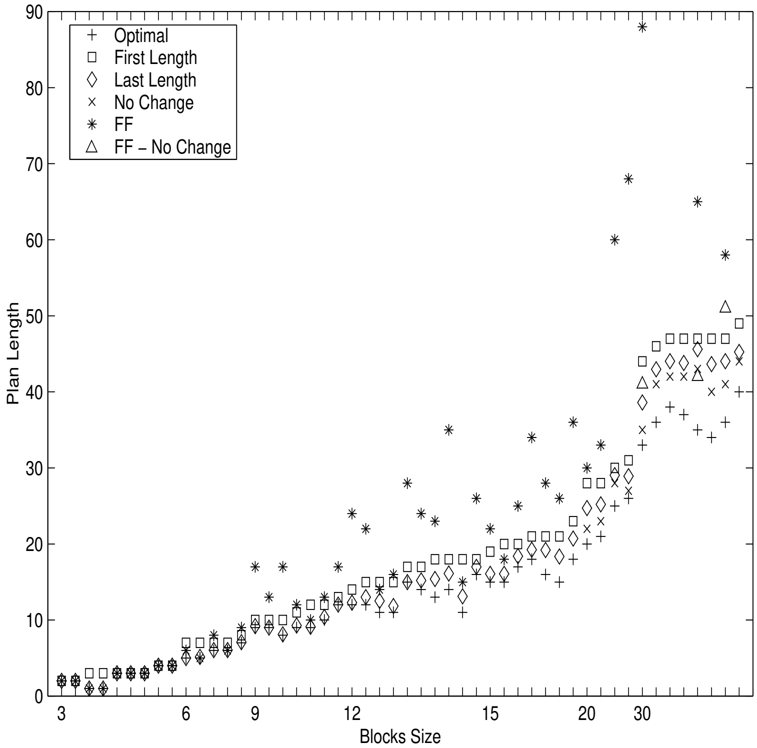

Figure 2: Policy Set 2 on the Blocks World Problems

Parameter	Setting
Termination	1000 generations
Population Size	20 plans
Initial Length	Up to 400 actions
Tournament Size	2
Maximum Plan Size	Up to 1000 actions
Genetic Operators	5% crossover, 95% reproduction
Shrink Mutation	Applied to 5% of children
Move Mutation	Applied to 5% of children

Table 1: GP Parameter Settings

Referring to Figure 2, some improvement could be made to the initial plan within 1000 generations even though the initial plans were reasonably close to optimal. The no change results managed to to shrink the plans a little more, and taking the 30 block problems again, these were shrunk down to around the 40 action mark. This difference between the two policy sets is returned to in the conclusions.

Also included in Figure 2 are results from FF [4]. We ran the 3 plans produced for the 30 block problems using the no change setup. The results are indicated with the triangles, and show significant shrinkage.

4.2 Briefcase Domain

For the Briefcase Domain we implemented five problems of two types. The first type, called simple, forced all the object to simple be transported from one location to the next. The second type, called circle, meant that the plan had to visit each location in a circular order, picking up objects and dropping the object off at the same time. The first number in the problem name indicates the number of locations, the second indicates the number of objects and the last number is number of briefcases:

1. 5-5-1 (simple): Briefcase at location A, objects 1-5 at location A, locations B-E are empty. The goal is to have all objects at location B.
2. 10-10-1 (simple): Briefcase at location A, objects 1-10 at location A, all other locations are empty. The goal is to have all objects at location B.
3. 5-5-1 (circle): Briefcase at location A, object 1 at location A, object 2 at B, object 3 at C, object 4 at D, and object 5 at E. The goal state is to have object 1 at B, object 2 at C, object 3 at D, object 4 at E, and object 5 at A.
4. 5-10-1 (circle): Similar to above but with two object at each location having to be transported to the next location.
5. 10-10-1 (circle): Similar to the third problem but with 10 objects and 10 locations.

Referring to Table 2, Policy Set 1, the random action selection allowed for quite inefficient plans. However, the system was able to shrink many actions out of the initial plans, with the no change criterion helping to further reduce the average plan length.

Referring to Table 3, Policy Set 2 on the simple type problems actually made the plans slightly harder to optimise, but for the circle type problems the plans were quite easy to optimise.

5 Conclusions and Future Work

The most successful current planners use heuristics and hill-climbing techniques. However, since no heuristic is perfect, such techniques often produce suboptimal plans, as in the case of FF. We have presented a linear plan optimisation technique, based on GP, which attempts to optimise plans. The system is domain independent, and can be used as addition to existing linear plan synthesisers. The system uses simple operations like mutation and crossover in order to accomplish this. The system could optimise plans to varying degrees of success depending on where the plans came from. A tentative conclusion is that plans made by $\mathcal{DM}$-Optimal policy sets can be optimised further towards the shortest plan than those made by satisficing policy sets.

We want to improve on the Generational framework suggested here for plan optimisation. There are a number of alternatives, such as a steady state algorithm, that we could adopt to decrease the length of the resulting plans. Also the system could be redesigned to optimise single plans.

We also want to broaden the definition of optimal to mean more than just plan length. More complicated domains with time, plan execution by an agent, resources, and so on, would make plan optimisation a multi-dimensional problem. It seems plausible that a genetic technique would be suitable for this kind of optimisation due to the way fitness functions and simulation are used.

Table 2: Policy Set 1 on the Briefcase Problems

Problem	Average First	Average Last (1000 gens)	Average Last (no change)	Optimal
5-5-1 (simple)	18.48	11.56	11.0	11
10-10-1 (simple)	116.12	27.16	21.2	21
5-5-1 (circle)	31.0	17.68	16.5	15
5-10-1 (circle)	57.16	32.64	27.8	25
10-10-1 (circle)	154.96	44.68	35.3	30

Table 3: Policy Set 2 on the Briefcase Problems

Problem	Average First	Average Last (1000 gens)	Average Last (no change)	Optimal
5-5-1 (simple)	20.0	12.84	11.0	11
10-10-1 (simple)	40.0	31.44	21.2	21
5-5-1 (circle)	23.0	15.0	15.0	15
5-10-1 (circle)	46.0	30.16	28.8	25
10-10-1 (circle)	48.0	30.08	30.0	30

6 Acknowledgements

The first author would like to thank the EPSRC for support via a quota studentship. The first author would also to thank the tireless support and contribution of his PhD supervisor John Levine. The first author would also like to thank Jose Ambité for his contribution of problems.

References

[1] Ambité, J.L., Knoblock, C.A.: Planning by Rewriting: Efficiently Generating High-Quality Plans. In Proceedings of the 14th National Conference on Artificial Intelligence, Providence RI USA, 1997

[2] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction. San Francisco CA, Morgan Kaufmann Publishers, 1998

[3] Bylander, T.: The computational complexity of propositional STRIPS planning. In Journal of Artificial Intelligence, 69(1-2):165-204, 1994

[4] Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. In Journal of Artificial Intelligence Research, Volume 14, Pages 253-302, 2001

[5] Koza, J. R.: Genetic Programming. The MIT Press, Cambridge MA USA, 1992

[6] Muslea, I.: SINERGY: A Linear Planner Based on Genetic Programming. In Proceedings of the 4th European Conference on Planning, pages 312-324, Toulouse France, Springer, Sep 1997

[7] Slaney, J., Thiébaux, S.: BLOCKS WORLD TAMED Ten thousand blocks in under a second. Technical Report TR-ARP-17-95, Automated Reasoning Project, Australian National University, Oct 1995

[8] Westerberg, C.H., Levine, J.: “GenPlan”: Combining Genetic Programming and Planning. In Proceedings for the UK Planning and Scheduling Special Interest Group, Milton Keynes UK, Dec 2000

[9] Westerberg, C.H., Levine, J.: Investigation of Different Seeding Strategies in a Genetic Planner. In Applications of Evolutionary Computing, Proceedings of EuroGP, pages 505-514, Lake Como Italy, Springer, Apr 2001