Applying Genetic Algorithms to Hierarchical Task Network Planning

Lea H. Ruscio, John Levine, and John K. Kingston

Artificial Intelligence Applications Institute
Division of Informatics, The University of Edinburgh,
80 South Bridge, Edinburgh, EH1 1HN, UK
{L.Ruscio, J.Levine, J.Kingston}@ed.ac.uk

Abstract

We describe a new planner that uses a genetic algorithm and domain-specific knowledge bases to solve hierarchical task network (HTN) planning problems. The knowledge base describes decompositions of tasks into subtasks using different resources. For any given problem, the GA evolves a solution that specifies a recursive decomposition of the tasks and a set of resources to be used. The planner has been tested on two simple domains (logistics and disaster relief), and is now being applied to a more complex military ground operations domain.

1 Introduction

1.1 HTN Planning and O-Plan

Hierarchical Task Networks (HTNs) can be used to represent large planning problems where high-level tasks can be recursively decomposed into sets of lower-level tasks, with basic actions at the lowest level. HTNs allow a great deal of flexibility for encoding interdependencies among subtasks, and alternate choices of task decompositions. There are a number of existing hierarchical task planners, including SHOP [1], SIPE [2], and O-Plan [3].

O-Plan uses schemas[1] to represent task decompositions. For example, a schema for evacuating a city has subtasks of moving evacuation teams and vehicles to the city, loading the population into the vehicles, and moving the loaded vehicles to the rendevous. The task of moving a vehicle can be decomposed into subtasks of checking and fueling the vehicle, briefing the driver, driving/flying to the location, etc. There may be multiple different schemas for a single task depending on the steps involved or assets required. For example, there might be different procedures and actions for an evacuation in response to a disaster, versus an evacuation before an expected disaster. Or there might be a choice of schemas for evacuating with buses versus evacuating with trucks.

The schemas use variables to indicate that an unspecified asset of a given type is required. For example, the schema to evacuate with trucks has a variable whose value indicates which specific trucks will be used. Some variable values are supplied in the initial problem specification, such as which city to evacuate. Most, however, are assigned during planning.

[1] "schema" is used throughout this paper to mean a script of actions in a knowledge representation sense. Note the contrast with the more common use of the term in a genetic algorithms context, where it refers to a subset of chromosomes/hyperplanes.

When given a specific set of tasks to plan, O-Plan recursively expands the high-level tasks, instantiating variables as it goes, until it has a set of basic actions which will accomplish those tasks. It schedules the basic actions, with deconfliction routines to ensure that no unit is assigned to two tasks at the same time. The expansion and instantiation occurs according to a backtracking algorithm similar to Prolog's, returning the first plan that satisfies the problem constraints. If replanning is requested, it backtracks through its choices trying other combinations of schemas and/or variable assignments.

Our main criticism of O-Plan is that the backtracking algorithm is not an effective way of selecting schemas and variables. O-Plan rarely returns an optimal plan without replanning. During replanning, O-Plan tries successive combinations of schemas and variables, without any indication of whether these combinations are likely to be better or worse than the ones it has tried. Replanning may take some time, depending on how many suboptimal combinations are encountered before an optimal combination. In some cases, as demonstrated in section 3.2 below, O-Plan may exhaust its resources before finding an optimal plan.

1.2 Military Ground Operations

Our research began in response to a request for an intelligent adversary capability to control the Red (enemy) forces in a military ground operations simulator. The adversary would have soldiers, tanks, trucks, ground artillery, and anti-aircraft artillery at its disposal. Scenarios would be played out over timeframes lasting from several hours to several days. The goal: to provide realistic opposition for experimental technologies controlling the Blue (friendly) forces.

This can be restated as a planning problem with three different levels. At the top level, general objectives are set, such as seizing an enemy airfield or defending headquarters. The next level determines the best methods for accomplishing the objectives, including assigning specific units to each objective. At the lowest level, individual units have been given a location or series of locations to move to and attack or defend, and must decide how to carry out those actions. The simulator already had the low-level functionality to move units and to configure them for attack or defence, with consideration of terrain and time constraints. It did not have any intelligent capabilities for the higher levels.

We focused our efforts on the middle level. A planner would take objectives as inputs and produce a list of unit assignments that could be translated into unit commands to be fed into the appropriate low-level routines. Objectives would need to be hand-coded for each scenario, but this would be an improvement on the previous requirement that each scenario have a hand-coded set of unit commands. An automated planner would also make it possible for the simulator to call for a replan if units were destroyed or objectives failed for other reasons. This was not an option with hand-coded movements.

The middle-level planning problem can take advantage of a great deal of domain knowledge. Most notable is knowledge about the types and numbers of units required to attack or defend different targets. Using this knowledge, we can treat it as an HTN planning problem. High-level objectives are accomplished through a sequence of lower-level actions. There are often several different sequences of low-level actions which could be used to accomplish a single high-level objective.

For the military ground operations domain, problems with 20 or more high-level objectives would not be uncommon. There will be many available assets, and many ways of assigning combinations of assets to objectives. The search space will be large, and a simple backtracking search may have difficulty producing good plans. A heuristic search would seem more appropriate.

We developed a planner that uses a genetic algorithm to perform a heuristic search. It uses a domain-specific knowledge base of schemas modeled after those used by O-Plan. It was designed with the military ground operations domain in mind, but can be applied to any HTN planning domain if a suitable knowledge base is provided.

2 Implementation

2.1 Overview

The planner has a knowledge base of schemas that specify task decompositions and resource requirements. Taken together, these schemas form the hierarchical task network. A genetic algorithm (GA) performs a heuristically-guided search through the possible plan expansions in the task network.

The GA maintains a population of chromosomes. Each chromosome represents a single plan, with each gene on the chromosome specifying one choice of how to expand a schema or which resource to assign to a task. In each generation of the population, pairs of chromosomes are selected for mating, a process which uses mutation and crossover operators to create two new chromosomes from the old ones. The new chromosomes form the population for the next generation, and the selection-mating-replacement procedure is repeated. A fitness heuristic provides a measure of how good a chromosome/plan is. Fitter chromosomes are more likely to be selected for mating, so the population should gradually evolve from the initial random solutions to optimal or near-optimal solutions. Refer to [4] for a more thorough introduction to genetic algorithms and evolutionary computation.

The planner exists as a standalone system and as a module within the military ground operations simulator. Both versions share the same core code, but the module has additional interface functionality to translate to and from the simulator's internal representations. Both versions are written in LISP, mainly because the simulator was written in LISP.

2.2 Knowledge Base

The knowledge base consists of a set of schemas. Each schema describes one way of completing one task. This may be a decomposition into subtasks, a choice among several alternate methods, or an indication that the task is a basic action and cannot be decomposed any further. The schemas also specify how many and what types of resources are needed for the task. Schemas may contain applicability conditions for themselves or for their required resources. They can also specify time durations and state effects.

The format and style of the schema specifications are similar to O-Plan's task formalism language, TF [5]. However, there are a few semantic differences between the schemas of the two planners and noticeable syntactic changes. In addition, there are many areas of TF which have not been implemented in the new planner since the schema functionality has been developed on an as-needed basis. See Figure 1 for some sample schemas.

2.3 Resources

Resources, also referred to as assets, are the tools which are available for use in carrying out the tasks. Different tasks may require different resources. Each resource has a type. Resources are stored in lists by type, for easier access during variable instantiation. See Figure 2 for sample resources for the schemas in Figure 1.

A distinction is made between exclusive and nonexclusive resources. Exclusive resources can only be used for one objective at a time. They include vehicles, evacuation teams, military units, etc. Nonexclusive resources can be used by multiple objectives at the same time. Cities and other locations are the most common example of nonexclusive resources.

2.4 Objectives

Objectives are the goals which must be achieved. They consist of a schema name and values for any required variables. Objectives may have suggested start and end times, or the timing may be determined entirely by asset availability. Objectives may also have interdependencies, where one objective cannot start until certain other objectives have been accomplished. All of this information is provided when defining the objectives. See Figure 3 for two sample objective sets for the schemas in Figure 1 and the resources in Figure 2.

2.5 Plan Representation

Each chromosome represents one possible plan for accomplishing the objectives. Chromosomes are lists of non-negative integers. Objectives are expanded one by one. At each choice of schema or variable, the next gene on the chromosome is used as an index into the list of possible choices. More specifically, the gene's value modulo the length of the list is used as an index. No distinction is made between genes for schema expansions and genes for variable instantiations. In fact, it is possible for the k^{th} gene to control an expansion for one chromosome and a variable instantiation for another.

Different plans require different numbers of choices, so in theory each chromosome would have a different number of genes. In practice, all chromosomes are created with the same length, and any excess genetic material is ignored during plan expansion. If a chromosome is too short to fully specify a plan, extra random genetic material is added to the end of the chromosome.

2.6 Genetic Operators

The genetic algorithm is based on the canonical GA as described in [6]. An initial population is created, members are selected for pairwise mating, the new generation replaces the old, and the process repeats.

Figure 1: Sample Schemas (Pseudocode)

```
move_shipment_to_docks
  type: choice
  required variables: s (shipment)
  expansions:
    move_shipment_to_docks_using_very_fast_vehicles
    move_shipment_to_docks_using_fast_vehicles
    move_shipment_to_docks_using_midspeed_vehicles
    move_shipment_to_docks_using_slow_vehicles
    move_shipment_to_docks_using_very_slow_vehicles

move_shipment_to_docks_using_very_fast_vehicles
  type: node
  required variables: s (shipment)
  new variables: v1 (very_fast_vehicle)
                 v2 (very_fast_vehicle)
  nodes:
    very_fast_transport_half_shipment(s v1 Docks)
    very_fast_transport_half_shipment(s v2 Docks)

move_shipment_to_docks_using_fast_vehicles
  type: node
  required variables: s (shipment)
  new variables: v1 (fast_vehicle)
                 v2 (fast_vehicle)
  nodes:
    fast_transport_half_shipment(s v1 Docks)
    fast_transport_half_shipment(s v2 Docks)

... similar for midspeed, slow, very slow ...

very_fast_transport_half_shipment
  type: basic
  required variables: s (shipment)
                      v (very_fast_vehicle)
                      to (location)
  duration: 2
  effects: at end, location of half of 's' is 'to'

fast_transport_half_shipment
  type: basic
  required variables: s (shipment)
                      v (fast_vehicle)
                      to (location)
  duration: 2.5
  effects: at end, location of half of 's' is 'to'

... similar for midspeed, slow, very slow ...
```

Figure 2: Sample Resources

```
Resources:
  locations: Docks, Warehouse
  shipments: S1, S2, S3, S4, S5
  very_fast_vehicles: GT0
  fast_vehicles: GT1
  midspeed_vehicles: GT2
  slow_vehicles: GT3
  very_slow_vehicles: GT4, GT5, GT6, GT7, GT8, GT9
Nonexclusive Resources: Docks, Warehouse, S1, S2, S3, S4, S5
```

Figure 3: Sample Objectives

```
Set 1:
  move_shipment_to_docks(S1)
  move_shipment_to_docks(S2)
Set 2:
  move_shipment_to_docks(S1)
  move_shipment_to_docks(S2)
  move_shipment_to_docks(S3)
  move_shipment_to_docks(S4)
  move_shipment_to_docks(S5)
```

Selection is done with a tournament of size three. Three chromosomes are picked at random and the one with the best fitness is selected for mating. This process, with replacement, is repeated as many times as there are members in the population. The result is a list of chromosomes which are mated in order. (First with second, third with fourth, etc.) The list may contain duplicates.

Mating takes pairs of chromosomes from the selection list and applies the crossover and mutation operators to obtain two new chromosomes.

The mutation operator is a standard pointwise random mutation. Each gene is mutated with a given probability, independently of any other genes. When mutated, a gene is changed to a random legal value.

The crossover operator is a one-point crossover that occurs with a given probability. The crossover point is randomly selected within the length of the shorter chromosome, if the lengths differ. All genetic material from that point on is exchanged.

2.7 Fitness

The fitness function measures how "good" a chromosome is, so that better chromosomes can be selected for mating. The planner's current fitness function calculates the total duration of the plan specified by the chromosome. Smaller values are better.

A simple schedule builder is used to calculate the duration. It expands the objectives in order, scheduling each one as early as possible given its required assets and the assets in use by previously-scheduled objectives. To do this, it maintains a list of intervals of scheduled time for each asset. When asked to schedule a slot for an objective, it takes the duration and the asset list and finds the first interval of that duration in which all of the assets are free.

We had considered several heuristics based on the maximum amount of time any single asset was required and the balance between assets. However, it was not clear that these would work consistently over many problems. Like the schedule builder, each of the heuristics required a full plan expansion. Most of the heuristics were only marginally quicker than the schedule builder; the best heuristic was slightly slower.

2.8 Example of Plan Expansion and Scheduling

The best way to understand the pattern of plan expansion and scheduling is to see an example. Consider the sample problem defined in Figures 1-3, using the first set of objectives. Suppose the chromosome is (9 87 16 53 42 14 35 39 ...).

There are no ordering constraints on the high-level objectives, so the schedule-builder starts with the S1 objective. *move_schema_to_docks* is a choice schema. The first gene determines which of the possible expansions is selected. There are 5 choices. 9 mod 5 = 4, so *move_shipment_to_docks_using_very_slow_vehicles* is selected and expanded next. It requires two *very_slow_vehicle*s. There are 6 choices. 87 mod 6 = 3, and 16 mod 6 = 4, so v1 = GT7 and v2 = GT8. Once the variables are instantiated, the task can be decomposed. For a node schema, each node must be expanded. The first node, *very_slow_transport_half_shipment (S1 GT7 Docks)*, is a basic action with a duration of 4. It requires the resources S1, GT7, and Docks. S1 and Docks are nonexclusive; only GT7's availability is considered. GT7 is not in use yet, so the objective is scheduled for the interval [0, 4) and the state is updated to reflect the new location of half of S1 at time 4. The second node is considered next. It is also a basic schema and can be scheduled in the interval [0,4) because it uses a different vehicle. Again the state is updated, with the other half of S1 also at the Docks at time 4.

The S1 objective has now been entirely scheduled. The S2 objective is next. Again, there is a choice of expansion at the top level. 53 mod 5 = 3, so the schema *move_shipment_to_docks_using_slow_vehicles* is used. That schema requires two new variables, each of type *slow_vehicle*. There is only 1 *slow_vehicle*. 42 mod 1 = 0 and 14 mod 1 = 0, so v1 = GT3 and v2 = GT3. Again each node in the task decomposition is a basic schema with no new variables. The first node can be scheduled in the interval [0, 3.5), with the state updated so that half of S2 is at the Docks at time 3.5. The second is scheduled for [3.5,7) since the vehicle GT3 has already been scheduled in the earlier interval. The state will be updated so that the other half of S2 is at the Docks as of time 7. The second objective is now fully expanded.

There are no more objectives, so scheduling is complete. The rest of the genes on the chromosome are ignored as excess. The fitness of this chromosome is 7.

3 Applications

3.1 The O-Plan Pacifica Disaster Relief Domain

O-Plan has several test domains available on its web page [7]. Initial testing used the Pacifica disaster relief domain to provide basic feedback on whether the planner was working properly. Six problems with varying difficulty levels were selected for testing. Each problem had between one and five objectives. There were no ordering constraints on the objectives.[2]

The planner was run 100 times on each problem. The same set of GA parameters was used for each run. These were: 100 generations, population size 30, initial

[2]None of the objectives were prerequisites for other objectives, though there were ordering constraints on the actions within each objective.

chromosome length 50, mutation rate 0.03, crossover rate 0.90.[3]

For the two simplest problems, an optimal plan was returned from all 100 runs. On three of the problems, the planner returned an optimal plan 92, 91, and 82 percent of the time, respectively. The other returned plans for those three problems were one time unit longer than the optimal length. For the final (and most difficult) problem, 38 percent of the returned plans were optimal, 52 percent were one time unit longer, and 10 percent were 3 time units longer.[4]

Though not exhaustive, these tests provided a measure of confidence that the planner was working as intended. It produces optimal or near-optimal plans for the test problems, in times ranging from 3 to 30 seconds. We also spot-checked additional problems without performing full trials. The resulting plans for those problems were also satisfactory.

3.2 A Logistics Problem: GA vs. O-Plan

The tests with the Pacifica domain showed that the GA-schema planner could produce good plans in reasonable amounts of time. We also constructed a test problem to demonstrate that the heuristic search of the GA could outperform the simple backtracking search in O-Plan.

Consider a simple logistics domain: Shipments need to be moved from a warehouse to the docks. There are five types of vehicles, rated by speed. They take 2, 2.5, 3, 3.5, and 4 hours respectively for a round trip from the warehouse to the docks, including loading, unloading, and other overhead time. Each vehicle can carry half of a shipment, so moving a single shipment requires two vehicles, or two round trips of a single vehicle. A shipment must use the same type of vehicle for both of its halves. See Figure 1 for the corresponding schema set.

For this specific problem, there are one each of the four fastest vehicles, and six of the slowest vehicles, initially located at the warehouse. Five shipments must be transported from the warehouse to the docks. See Figures 2 and 3 for resource and objective definitions. The optimal plan uses the very fast vehicle, the fast vehicle, and all 6 very slow vehicles. Its duration is 5 hours. The next best plan uses a midspeed vehicle and takes 6 hours.

O-Plan was asked to find successively shorter plans for this problem. It found plans of length 20, 16, and 12 in seconds. Finding 10-hour and 8-hour plans took about 50 minutes, with over 39,000 plans examined. Trying to find a 6-hour plan failed when memory was exhausted. This was on a test machine with 128Kb of RAM, after more than two hours of computation and 109,000 examined plans.

In comparison, the GA-schema planner takes approximately 4.5 seconds on the problem and returns optimal or near-optimal plans. In 100 trials, it returned a 5 hour plan 34 percent of the time and a 6 hour plan 66 percent of the time. Ideally we would like to see the optimal plan more frequently,[5] but the planner is finding good plans in relatively short runtimes.

It should be noted that O-Plan's schemas were intentionally ordered for general

[3] This is the set of parameters used for all of our testing. They were arbitrarily chosen, but have worked well enough so far.

[4] This problem had no legal plans which were two time units longer than the optimal plans.

[5] Experimentation with the GA parameters could potentially improve results.

problems on this domain, not this specific problem.[6] Using the very fast vehicles is generally preferable to using the fast vehicles, which is better than using the midspeed, etc. The Pacifica schemas use an equivalent ordering, so this should be a resonable choice of ordering on our part.

This problem was constructed specifically to show O-Plan's limitations, but it is not a particularly unrealistic problem. It is a simple problem but it requires significant backtracking, making it difficult for O-Plan. The GA-schema planner can outperfrom O-Plan on this type of problem because its search is guided by the length heuristic, rather than backtracking using a pre-programmed preference.

3.3 The Military Ground Operations Domain

Once we were satisfied that the planner performed reasonably well on simple domains, we returned to our initial problem - planning unit movements for a military ground operations simulator. The objectives are seizing and defending targets, which may be roads, bridges, cities, airfields, oilfields, harbours, etc. Targets are classified as large, medium, or small areas.

One interesting twist to the problem is the hierarchical nature of the assets. Each asset has a type and a numerical level which corresponds to its place in the division/battalion/regiment/etc hierarchy of the units. Level 1 units are comprised of several level 2 units, each comprised of several level 3 units, and so on. Choosing a unit for an objective affects the availability of the smaller units it is composed of and the larger units it is a part of. This had not been an issue in the simpler domains, but is easily addressed. An alternate version of the schedule builder looks for free intervals for the required assets plus all assets above or below them in the hierarchy.

Assets have specific types, but are also classified into the broad categories of air defence, ground forces, and artillery. In each category, there are options for each target size. In general, large areas use a level 2 unit or several level 3 units, medium areas use a level 3 unit or several level 4 units, and small areas use a level 4 unit or several level 5 units. An objective to seize or defend a location can use an air defence option and/or a ground forces option and/or an artillery option. Probability of success is higher if all three are used, but the option to omit allows for the possibility that there may not be enough assets for a full allocation to each objective.

Eventually the planning problems will consist of a number of friendly areas to defend (headquarters, roads, bridges, etc), plus several high-profile hostile targets to seize with a number of intermediate targets to seize and defend en route to each larger targets. For initial testing, a problem was designed with 4 friendly areas to defend, two high-profile targets (the airfield and the oilfields), and two sequences of 4 and 5 intermediate targets each (bridges: 1 friendly, 3 or 4 hostile). Each hostile target has separate objectives for seizing and defending, resulting in a total of 22 objectives. Each objective has start and end times assigned. Units should be at the target location by the start time, then seize or defend it until the specified end time.

[6] When planning and backtracking, O-Plan tries schemas in the order in which they appeared in the schema file. It is therefore common practice to order the schemas in the file based on preference.

The test results were generally good. Appropriate sets of units were moved towards the objectives for the appropriate times. However, the results were not ideal. One problem in particular was observed, namely that there is no preference for plans which use more of the available assets: air defence + ground troops + artillery versus ground troops only. In fact, if start times are not specified, there is often a pressure to use fewer assets because it takes less time to move them into place, resulting in a shorter plan.

The obvious solution to this problem would be to consider effectiveness in the fitness function. More-effective plans should be selected over less-effective plans of the same length. More-effective plans should possibly be selected over shorter less-effective plans, though the other extreme where objectives are delayed in order to stockpile more assets may not be desirable either. Some balance between the length and the effectiveness is required. It will also be necessary to combine effectiveness measures for different objectives to get an effectiveness measure for the entire plan. This perhaps should consider objective priorities so that plans with high effectiveness on important objectives and lower effectiveness on other objectives are preferred over plans with low effectiveness on important objectives and high effectiveness elsewhere. Introducing effectiveness estimates is the next step, but will take some thought.

Despite this issue, the initial results have been promising. The planner allocates assets to the objectives at the appropriate times. It is not making the best possible use of assets yet, but it is assigning them appropriately. We already have a capability to take a list of objectives and produce a list of unit taskings which can be used to generate the low-level commands. The next step is to improve on those taskings.

4 Ongoing and Future Work

Our work on the military ground operations domain is ongoing. The next step is an effectiveness measure in the fitness function, with attention to the balance between schedule length and schedule effectiveness. This enhancement will be a benefit to any domain where factors in addition to or instead of required time determine which plan is best.

We are considering tests on other domains, and hope to perform more rigorous comparisons against other planners. There are also a number of open questions concerning the chromosome representation and the effects of the structure of the schema set. This has been an exploratory project. Initial results are promising, but there are many more details to explore.

5 Conclusions

The GA-schema planner can be applied to any hierarchical task network planning problem which can be expressed in terms of schemas. Using a GA to control schema expansion and variable instantiation can produce good results, sometimes in less time than a simple backtracking system as in O-Plan. The planner has been applied to two simple domains, returning optimal or near-optimal plans on all tests. It is currently being applied to a more complex military ground operations domain. Initial results in this domain are good, though they also indicate that more work is needed. Several enhancements are currently being implemented to address the issues raised. In general, though, the work to date seems to indicate that genetic algorithms can be usefully applied to hierarchical task network planning problems.

6 Acknowledgements

The authors would like to acknowledge financial and domain expertise support for this research from BBN Technologies, San Diego, CA. This support was provided through BBN project 99-BBN-301, “Agile Control of Military Operations”, in response to DARPA BAA 99-18.

The Marshall Aid Commemoration Commission should also be acknowledged for funding the first author, via a Marshall Scholarship.

References

[1] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. *Proceedings of IJCAI-99*, 1999.

[2] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufman Publishers, Inc., 1988.

[3] K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52, 1991.

[4] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, third edition, 1996.

[5] O-Plan team. O-plan task formalism manual. Technical report, Artificial Intelligence Applications Institute, University of Edinburgh, July 1995.

[6] D. Whitley. A genetic algorithm tutorial. Technical report cs-93-103 (revised), Department of Computer Science, Colorado State University, November 1993.

[7] O-Plan Team. O-plan web demonstrations. *http://www.aiai.ed.ac.uk/˜oplan/web-demo/index.html*, 2000.

[8] Lea H. Ruscio. Planning with an enterprise model: Applying genetic algorithms to hierarchical task network planning. Msc thesis, Department of Artificial Intelligence, University of Edinburgh, September 2000.