Towards Automatic Model Generation by Optimization

Per NYBLOM [a] and Patrick DOHERTY [a]
[a] *Department of Computer Science, Linköping, {perny, patdo}@ida.liu.se*

Abstract.
The problem of automatically selecting simulation models for autonomous agents depending on their current intentions and beliefs is considered in this paper. The intended use of the models is for prediction, filtering, planning and other types of reasoning that can be performed with simulation models. The parameters and model fragments of the resulting model are selected by formulating and solving a hybrid constrained optimization problem that captures the intuition of the preferred model when relevance information about the elements of the world being modelled is taken into consideration. A specialized version of the original optimization problem is developed that makes it possible to solve the continuous subproblem analytically in linear time. A practical model selection problem is discussed where the aim is to select suitable parameters and models for tracking dynamic objects. Experiments with randomly generated problem instances indicate that a hillclimbing search approach might be both efficient and provides reasonably good solutions compared to simulated annealing and hillclimbing with random restarts.

Keywords. Automatic model generation

1. Introduction

Simulation models are commonly used in many disciplines for various purposes. In Artificial Intelligence, such models can e.g. be used to support prediction and decision making capabilities for autonomous agents. Agents that operate in complex environments can typically not use highly detailed models of all parts of their surroundings, especially when such models are used for task planning where the effects of many possible alternative action choices have to be predicted. Lack of computational resources often limit both the model and the planning horizon when a timely response is of importance. Traditionally, anytime algorithms [2] are often used to handle the need for a timely response where a reasonably good solution can be found quickly and if possible, the solution quality increases with time.

Another complementary approach to deal with the problem of limited computational resources and the need for a timely response is to try to vary the models and reason on several abstraction levels. In this paper we will consider an approach where the models are generated dynamically depending on the current beliefs and intentions at hand. This is very much related to the call for dynamic abstraction techniques within hierarchical reinforcement learning [1] and the automatic generation of simulation models to answer queries about physical dynamic systems [9].

The focus of this approach is to find a good tradeoff between the resulting model's accuracy and feasibility for the current situation in order to get as good expected performance of the agent as possible given the computational limitations. A part of that tradeoff is in this paper formulated as an optimization problem that captures the limitations and how good the resulting model will be given the agent's available model fragments and relevance information. A motivating example is also presented where the problem is to find the parameters of a set of model fragments that can be used to represent dynamic objects (such as vehicles) for tracking applications. These objects can be more or less relevant for different reasons and the relevance can change depending on the situation. The agent must therefore adapt its models to be able to get the most out of its computational resources. We are not aware of any previous work where this particular approach is taken to generate models.

We believe that automatic model generation in general has the possibility of making autonomous agents more robust when faced with unexpected situations because it can be used to focus the agents' attention by generating models of different complexity when e.g. their default assumptions turn out to be invalid [11].

2. Motivating Example

This section introduces a practical model generation/selection problem which motivates the dynamic change of models and other parameters depending on situation and relevance information. The context will be simulation model generation for particle filters with the purpose of tracking dynamic objects in e.g. traffic situations.

Particle filters [4] are often used in tracking applications where it is important to model nonlinear or multimodal phenomenons. When constructing a particle filter, it is often not clear what type of information to use in each particle and how many particles to use in total. A good choice must be a reasonable tradeoff that takes the available computational resources and the accuracy of the resulting filter into account.

When several objects or relations between objecs are important for the tracking agent, another complication arises. The agent must then be able to dynamically prioritize between the different parts of its environment and change its models accordingly. The relevance of these parts may change rapidly depending on the situation and this should be reflected in the way the agent uses its resources.

To make this method work, a metamodel that predicts the performance and computational cost of the different generated models must be available. This requires that a large empirical study must be performed where the performance of the models are measured for the different filter parameter settings. For particle filters in general, the performance increases with the number of particles and update frequency and we expect that this trend can be modelled by suitable mathematical functions.

When such a metamodel is available, it is possible to use optimization methods to maximize the performance of the filter sets given the relevance of the objects and relations between them. The optimization problem must also be constrained with the available computational resources.

This paper presents a method to select model types and settings by formulating such an optimization problem that can take relevance information and available computational resources into consideration.

3. The Model Selection Problem

This section describes how to use optimization for model selection where the input is the possibly relevant elements E (such as vehicles, other objects and features such as distances between objects), a set of possible representations REP (model fragments such as models of vehicles and pedestrians) for these elements together with representation value. Intuitively, the representation value describes how "good" it is to represent the elements E in the agent's current context with the different representations which is supposed to be extracted empirically as described in Section 2. This information is only considered in isolation in the metamodel and the task of the model selection is to find a good tradeoff between the representations when all these elements must be represented in some way *together in the same model* with limited computational resources. The output from the model selection is the representation to use for each element together with the update frequencies for the different model fragments.

Model fragments that include derivatives of state variables can be simulated with different update frequencies and the accuracy of the result often increases with the frequency. The quality of the result does not increase linearly with the frequency though. For example, if there is a large quality difference between 1 Hz and 10 Hz update frequency for a particle filter, there is typically not the same increase in quality for a difference between 1000 Hz and 1009 Hz (although there is some increase). It is important that this model quality increase is captured in the optimization problem formulation (see the frequency value function v_F below).

The optimization problem, which attempts to formulate the preferred simulation model parameters in terms of update frequencies and choice of model fragments, contains the objective function $g : \mathbb{R}^n \times REP^n \rightarrow \mathbb{R}$ which is assumed to have the following format:

$$g(f, r) = \sum_{e \in E} v_R(e, r) v_F(r_e, f_e) \tag{1}$$

where E is the set of elements $\{e_1, \cdots e_n\}$, r is the vector $[r_{e_1} \cdots r_{e_n}]$ of discrete variables that determine the representation r_e for each element $e \in E$ and f is the vector $[f_{e_1} \cdots f_{e_n}]$ of continuous variables that specifies the update frequencies. The *frequency value function* $v_F : REP \times \mathbb{R} \rightarrow \mathbb{R}$ captures the non-linear increase in quality of the resulting model with the frequency when a certain representation r_e with frequency f_e is used. The purpose of the *representation value function* $v_R : E \times REP^n \rightarrow \mathbb{R}$ is to represent both the relevance of an element and the suitability of representing it with a representation r_e in the context of all other elements' representations.

The resulting model takes a certain amount of time for each step depending on the set of model fragments used, and since the computational resources are limited, it is important to choose the model fragments wisely depending on how computationally intensive they are to simulate. The function $t_{step} : E \times REP^n \rightarrow \mathbb{R}$ specifies the amount of time it takes to make one step with a model fragment representation $r_e \in REP$ in the context of the other elements' representations. The constant T_{max} specifies the maximum amount of real world time that the simulation model can take per simulated time.

The update frequencies of the different representations then determine the following constraint:

(2) $$c(r, f) = \sum_{e \in E} t_{step}(e, r) f_e - T_{max} \leq 0$$

If the frequency value function $v_F(r_e, f_e)$ is strictly increasing for a fixed representation r_e (which seems to be a safe assumption), then the inequality in equation 2 can be replaced by an equality because it will always be active in an optimal solution.

The limit of the coarseness of the simulation updates can be modelled as follows:

(3) $$\forall e \in E[f_e \geq f_{min}(r_e)]$$

where the function $f_{min} : REP \rightarrow \mathbb{R}$ determines the minimum update frequency for each representation to increase the possiblity of a better resulting simulation model.

The *continuous subproblem* is defined as the constrained optimization problem in equations 1, 2 and 3 when r is fixed. Similarly, the *discrete subproblem* is defined as the optimization problem where f is fixed. Note that both of these subproblems have to be solved simultaneously, the only reason to divide them into two subproblems is that the continuous one can be solved separately by pure continuous methods.

4. Problem Specialization

The optimization problem defined by the Equations 1, 2 and 3 are supposed to be used in autonomous agents that may not have much computational resources. This means that the problem must be solved very quickly, which typically means that a reasonably good solution must be ready within few seconds or fractions of a second depending on the situation.

We have specialized the original optimization problem by selecting a particulary suitable frequency value function $v_F(r_e, f_e)$ that both approximately captures the constraints in equation 3 and makes the continuous subproblem very easy to solve. This makes it possible to use a search strategy where the discrete variables in r are first set to a value and then the global optimum of the continuous subproblem is found.

Several options for choosing the frequency value function $v_F(r_e, f_e)$ are possible but it is important that $v'_F(r_e, f_e)$ is a strictly decreasing function for a fixed r_e (due to the decrease in quality increase with frequency. See the discussion in Section 3). Our choice of the continuous function $v_{F,c}(f_e)$ (which denotes $v_F(r_e, f_e)$ when r is fixed) is a variant of the natural logarithmic function:

(4) $$v_{F,c}(f_e) = log(1 + f_e - f_{min,e})$$

This choice of frequency value function has the following consequences:

- If $f_e < f_{min,e}$, the element e provides a negative contribution to the objective function which can either be seen as a penalty for the quality decrease of e's simulation result or as a way to filter out elements that are irrelevant

- f_e can never be less than $f_{min,e} - 1$ ($v_{F,c}$ is not defined otherwise)

The derivative of $v_{F,c}(f_e)$ wrt f_e is a positive and strictly decreasing function which means that it fits the description of a frequency value function in Section 3. We are not certain that this particular choice of frequency function is suitable for selecting the update frequency of particle filters without any empirical investigation, but any concave, increasing function will make the approach described in this paper feasible.

5. The Continuous Subproblem

One important key to solving the reformulated problem is to take advantage of the fact that for every choice of representation r, the optimization problem transforms into a continuous optimization subproblem that can be solved in linear time in terms of the number of elements $|E|$.

The continuous variant of the objective function $g_c(f)$ and constraint $c_c(f)$ becomes the following:

$$g_c(f) = \sum_{e \in E} v_{R,e} log(1 + f_e - f_{e,min}) \tag{5}$$

$$c_c(f) = \sum_{e \in E} t_e f_e - T_{max} = 0 \tag{6}$$

where $v_{R,e} = v_R(e, r)$ and $t_e = t_{step}(e, r)$ for a fixed value r.

The objective function $g_c(f)$ is concave and the linear constraint is convex, which means that an optimal solution exist. The KKT conditions [8] for the continuous subproblem then shows that there exists a scalar λ in all local optimal solutions such that the following condition holds [1]:

$$-\lambda \nabla g_c(f) = \nabla c_c(f) \tag{7}$$

$\nabla g_c(f)$ and $\nabla c_c(f)$ are the gradients of $g_c(f)$ and $c_c(f)$ wrt f.

For our choice of $v_{F,c}(f_e)$ (see Equation 5), this condition is equivalent to:

$$\lambda \frac{-v_{R,e}}{1 + f_e - f_{e,min}} = t_e \tag{8}$$

for all elements $e \in E$, which defines a system of $|E|$ linear equations with $|E| + 1$ unknowns. The constraint in Equation 6 fills in the missing equation which in total gives the following system of equations:

[1] The negative sign on the left hand side in Equation 7 is due to the formulation of the KKT conditions for a minimization problem which in our case means that $-g_c(f)$ is minimized.

$$\begin{bmatrix} t_{e_1} & & & v_{R,e_1} \\ & \ddots & & \vdots \\ & & t_{e_n} & v_{R,e_n} \\ t_{e_1} & \cdots & t_{e_n} & 0 \end{bmatrix} \begin{bmatrix} f_{e_1} \\ \vdots \\ f_{e_n} \\ \lambda \end{bmatrix} = \begin{bmatrix} t_{e_1}(f_{e_1,min}-1) \\ \vdots \\ t_{e_n}(f_{e_n,min}-1) \\ T_{max} \end{bmatrix} \tag{9}$$

where $n = |E|$.

This system of equations can be solved in linear time ($\Theta(n)$) by first solving λ:

$$\lambda = \frac{T_{max} - \sum_e t_e(f_{e,min}-1)}{-\sum_e v_{R,e}} \tag{10}$$

and then finding the solution for f:

$$f_e = f_{e,min} - 1 - \frac{v_{R,e}}{t_e}\lambda \tag{11}$$

f_e will become less than $f_{e,min} - 1$ if $\lambda \geq 0$ ($v_{R,e}$ and t_e are always strictly positive constants), which would make $g_c(f)$ undefined for that solution. It is possible to avoid this problem by noting that λ can only become positive if $T_{max} - \sum_e t_e(f_{e,min}-1) \leq 0$, which in practice means that the agent is unable to simulate the model fragments even if all frequencies are set to the lowest possible.

Since there is a unique solution by directly applying the KKT conditions and at least one optimal solution exists, the solution found by solving Equation 9 is the global optimizer if $T_{max} - \sum_e t_e(f_{e,min}-1) > 0$. If $T_{max} - \sum_e t_e(f_{e,min}-1) \leq 0$, there is no solution.

If the choice of frequency value function turns out to be a bad one, it is still possible to solve the continuous subproblem by convex optimization methods due to the nature of frequency value function.

6. The Discrete Subproblem

We have shown that the optimal solution to the continuous subproblem can be calculated in linear time. In this section we show that the *0-1-knapsack optimization problem* can be transformed into the discrete subproblem in polynomial time, which means that the discrete subproblem is NP-hard [5].

The 0-1-knapsack optimization problem is defined as follows:

$$\text{maximize } \sum_{i=1}^{n} p_i x_i \tag{12}$$

$$\text{subject to } \sum_{i=1}^{n} w_i x_i \leq c \tag{13}$$

where $x_i = 0$ or 1.

If $REP = \{0, 1\}$, $v_F(r_e, f_e) = 1$ for all $r_e \in r$ when $f_e = 1$, $v_R(e_i, r) = p_i r_i$ and $t_{step}(e_i, r) = w_i$, then any 0-1-knapsack optimization problem can be transformed into a discrete subproblem in polynomial time. Due to the NP-hardness of the 0-1-knapsack optimization problem, the discrete subproblem is NP-hard as well.

7. Experiments

We have performed a set of experiments where the focus was to find an efficient method that delivers reasonably good solutions quickly, which is useful when a timely response is of importance. The strategy for solving the complete specialized hybrid optimization problem is to use a separate search for the discrete variables which are then evaluated by solving the continuous subproblem quickly. Our method of choice for the search in the discrete variables was hillclimbing (HC) local search which was compared to simulated annealing [6] and random restart hillclimbing (RRHC).

HC and RRHC used a neighbourhood function that returns $|E|(|REP|-1)$ neighbour states in which every single element $e \in E$ was set to all its possible representations REP. All states in the neighbourhood are valued by solving the continuous subproblem. The simulated annealing implementation used a random neighbourhood function N_{sa} where two elements' representations were changed randomly and the temperature was decreased as $t \leftarrow 0.98t$ where t is the current temperature. The temperature level was changed when 50 state changes had been performed or when the maximum number of steps (1000) had been taken. SA terminated when 5 temperature levels had passed in a row without any state change and then HC was performed on the best solution found so far (in order to at least get a local optimal solution).

Each test used randomly generated problem instances where the representation value function $v_R(e,r) = v_R(e,r_e)$ and $t_{step}(e,r) = t_{step}(e,r_e)$ which made it feasible to represent v_R and t_{step} with matrices. This simplification does not change the NP-hardness of the discrete subproblem. There were two possible starting states available: One was called the *greedy start state* r_g and it maximized $\frac{v_R(e,r_e)}{t_{step}(r_e)}$ (most value per simulation step time). The other one was called the *cheap start state* r_c which minimized $t_{step}(r_e)(f_{min}(r_e)-1)$ (cheapest representation for all elements). The cheap start state is always a valid state if any solution exists.

When an invalid state s was encountered (happens when $\lambda \geq 0$), the value for s was set to $-\gamma(\lambda+1)$ where γ is set to a large constant (10^9 in our implementation) to penalize invalid solutions hard. This solution seemed to work very well for this problem type and is related to the use of penalty functions in constrained optimization problems [10].

7.1. Problem Difficulty

We are not able to predict at this stage what characteristics the problem instances will have that will occur in practical situations when models are to be generated. We do believe that for the environment described in Section 2, the relevance function will take many possible values depending on the situation and that no particular pattern will exist that persist over all problem instances that we will consider. We initially performed an experiment with completely random problem instances and the result indicated that the simple hillclimbing approach were able to seriously compete with RRHC and SA. This result might be a good sign for practical applications due to the low cost of HC but we wanted to perform tests with more difficult problem instances. One possible measure of the difficulty of problems can be identified by varying T_{max}. A low T_{max} makes a fewer set of representations possible to use, which leads to a smaller set of valid solutions. When T_{max} increases, the set of solutions increase until all representations are possible for all elements. Somewhere between the smallest and very large T_{max}, we expect to find the most problem instances that have non-trivial solutions, which could possibly indicate a phase transition.

In the randomly generated problem instances, a constant $\alpha > 1$ is used to approximately vary the number of valid states and problem difficulty. T_{max} is set to $\alpha \sum_{e \in E} t_{step}(r_c)(f_{min}(r_c) - 1)$ where r_c is the cheap start state.

7.2. Cheap VS Greedy

When α is rather low, the set of possible states is reduced and it might seem natural to start the search in the cheap start state. If α is high, it might be better to start in the greedy start state instead. In one experiment, we compared the solution found by HC to SA and RRHC when HC started in the greedy, cheap and a random start state. α was varied between 1.1 and 10000 and 2000 randomly problems were generated for each setting with 30 elements and 20 representations. The result of this experiment is shown in Figure 1 where the upper subplot shows the fraction of the states when RRHC or SA finds a better solution than HC and the middle subplot shows the relative difference $\frac{|V_{best} - V_{hc}|}{|V_{best}|}$ between the best value found V_{best} and the result after HC V_{hc}. The lower plot shows the number of states visited by the hillclimbing search for the different start state types.

From the results it seems like the cheap state is the best singleton start state for this class of problem instances if one only considers the quality of the result. If, on the other hand, the time it takes to find a local optimum is more important than the solution quality, the greedy start state performs better than the cheap one for $\alpha \gtrsim 5$. The best result is received when both the greedy and the cheap start state are used. It is also interresting to note that the random start state provides as good solution quality as the greedy start state but it takes much longer time to find a local optimum.

For a much larger α, we expected that the greedy start state would outperform the cheap one due to the advantage of starting closer to a local optimum. The results in Figure 1 indicate on the other hand that the cheap start state is better to use even for a large α if the solution quality is very important. However, when HC starts at the greedy start state the time until a local optimum is found is much less than for the cheap and random states.

8. Conclusion and Future Work

The paper investigated a particular type of optimization problem that aims to capture the intuition of how the relavance of elements in an environment should be reflected in an automatically generated model that can be used for reasoning by autonomous agents. It was argued that a particular specialized version of the original optimization problem was suitable for the task due to the possibility of solving the continuous subproblem exactly in linear time. It remains to show that the logarithmic function used in the specialized version can model the performance increase wrt update frequency, but if this is not possi-

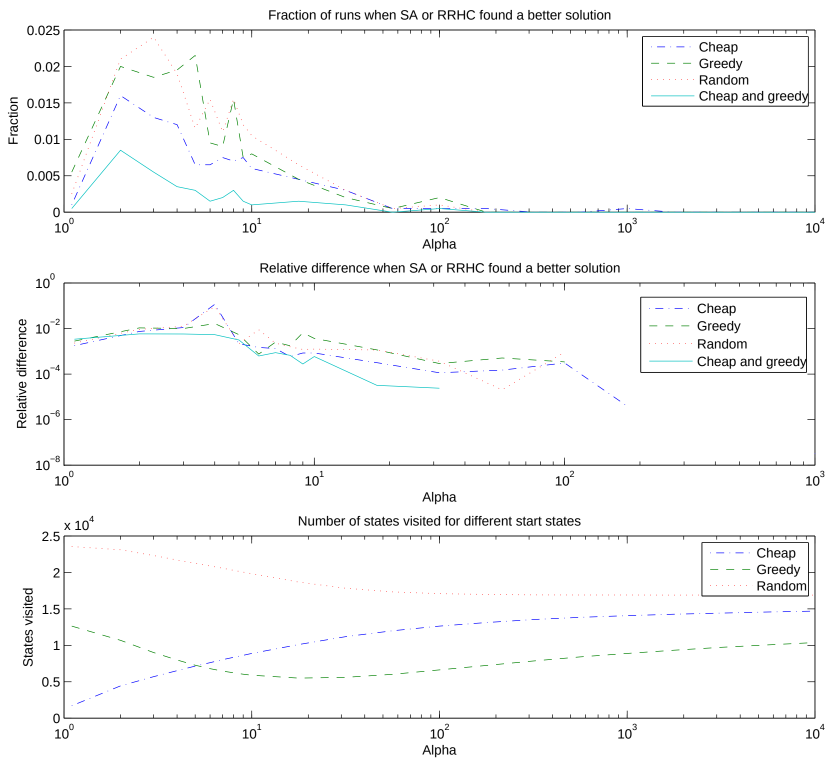

Figure 1. The experimental results when α was varied between 1.1 and 10000.

ble we might have to use convex optimization methods instead. The discrete subproblem was shown to be NP-hard.

The complete hybrid reformulated problem was solved with hillclimbing (HC) search with a simple neighbourhood function and compared to simulated annealing and random restart hillclimbing. Within the limitations of the experiment, the results indicated that informed starting states were better than randomly generated ones if the time to find a local optimum is of importance. The solution quality was best when HC was started in a so called “cheap” state which is always a valid state if a solution exists at all.

As future work, we will work towards actually using the model generation in a context where settings for particle filters are selected during vehicle tracking missions with our real autonomous helicopter system [3]. We also plan to use the method to select models for other tasks such as planning and extraction of the most likely sequence. Methods for such tasks that are more suitable for use with simulation models will be prioritized such as forward search, sequence estimation with particle filters [7] and reinforcement learning [13]. Some of these methods have already been used in context of task planning where simulation models are used to generated tractable discrete planning models [12] [11].

A future issue is also the representation of the function $v_R(e, r)$ which can be used to express the suitability of element representation combinations. We expect that constraints such as $r_{e_i} = a \Rightarrow r_{e_k} = b_1 \vee \cdots \vee r_{e_k} = b_m$ will be common in problem domains where a choice of representation for an element constrain other elements' representations. The main problem with these "extra constraints" is that we then have to deal with satisfiability of these constraints as well.

Acknowledgements

Many thanks to Martin Magnusson who has given a lot of useful comments to earlier versions of this paper.

References

[1] A. G. Barto and S. Mahadevan, 'Recent advances in hierarchical reinforcement learning.', *Discrete Event Dynamic Systems*, **13**(4), 341–379, (2003).

[2] T. L. Dean and M. Boddy, 'An analysis of time-dependent planning', in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54, (1988).

[3] P. Doherty, 'Advanced research with autonomous unmanned aerial vehicles', *Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning*, (2004).

[4] A. Doucet, N. Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice.*, Springer Verlag New York, 2001.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, 'Optimization by simulated annealing', *Science*, **220**(4598), 671–680, (1983).

[7] M. Klaas, D. Lang, and N. Freitas, 'Fast maximum a posteriori inference in monte carlo state spaces', in *Tenth International Workshop on Artificial Intelligence and Statistics*, (2005).

[8] H. W. Kuhn and A. W. Tucker, 'Nonlinear programming', in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492, (1951).

[9] A. Y. Levy, Y. Iwasaki, and R. Fikes, 'Automated model selection for simulation based on relevance reasoning', *Artificial Intelligence*, **96**(2), 351–394, (1997).

[10] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Verlag, 2006.

[11] P. Nyblom, 'Dynamic abstraction for hierarchical problem solving and execution in stochastic dynamic environments', in *Starting AI Researcher Symposium (STAIRS)*, (2006).

[12] P. Nyblom, 'Dynamic planning problem generation in a uav domain', *6th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, (2007).

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning An Introduction*, The MIT Press, 1998.