Route planning with ordered constraints

Brian Logan
School of Computer Science, University of Birmingham
Birmingham B15 2TT UK

b.s.logan@cs.bham.ac.uk

Abstract

Route planning in realistic terrains is a critical task for many autonomous agents such as mobile robots, autonomous vehicles or Computer Generated Forces. The route planning problem is often formulated as one of finding a *minimum-cost* route between two locations in a digitised map which represents a complex terrain of variable altitude, where the cost of a route is an indication of its quality. However route planners which attempt to optimise a single measure of plan quality often fail to provide real-time performance and the composite cost functions on which they are based are difficult to devise or justify. In this paper, we present a new approach to route planning in complex terrains based on a novel constraint-based search procedure, A^* with bounded costs (ABC), which generalises the single criterion optimisation problem solved by conventional route planners. Rather than attempting to find the lowest cost plan to achieve a goal, we plan to satisfy an ordered set constraints where the position of the constraint in the ordering reflects its importance. This approach provides a means of more clearly specifying agent tasks and more precisely evaluating the resulting plans.

Keywords: route planning, constraint-based planning, real-time planning.

1 Introduction

Route planning in realistic terrain is a critical task for many autonomous agents such as mobile robots, autonomous vehicles and Computer Generated Forces, as many of the agent's higher-level goals can only be accomplished if the agent is in the right place at the right time. In our work we are exploring architectures for agents which play a variant of the game of 'hide-and-seek' in complex terrains. Hide-and-seek provides a useful framework for investigating the interaction of motion and terrain, trading off current and future advantage and predicting the behaviour of other agents.

Our hide-and-seek agents operate in a dynamic simulated environment containing synthetic or real terrain data defining hills, valleys, impassable areas etc. and must act on the basis of incomplete or uncertain information. Each agent is initialised with one or more goals, for example to find the other agents or to remain concealed from them, and can acquire additional goals as a result of its interactions with other agents and its environment. A goal can be characterised as a relationship between the agent and the terrain, for example: being at the point A; being able to observe A; being hidden from an observer at A, and so on. Often these goals or the plans to achieve them are subject to additional constraints, for example that the agent should be at point A at or before a certain time, or that the route to point A should be concealed from one or

more opposing agents. Moreover, the amount of time an agent can afford to spend on planning depends on the current situation: uncertainty about the terrain, the positions of opponents etc. may mean that it is not worth developing a detailed plan. It is therefore desirable if the planner can quickly return a partial plan, or a crude plan only the first segment of which has been developed in detail, as a basis for immediate action.

In this paper we present a new approach to real-time route planning in complex terrains based on a novel constraint-based search procedure. In the next section we briefly outline some of the problems with conventional approaches to route planning based on A^*. In section 3 we present a new approach to route planning which generalises the single criterion optimisation problem solved by conventional route planners. In section 5 we describe our approach to real-time planning which uses meta-management rules to control the planner, allowing the agent to explicitly monitor the progress of the planner to determine when a satisfactory plan has been found, to relax or re-order the constraints when the planner is not making progress, or to interrupt the planner if the situation changes.

2 Route planning with A^*

The route planning problem is often formulated as one of finding a *minimum-cost* (or low-cost) route between two locations in a digitised map which represents a complex terrain of variable altitude, where the cost of a route is an indication of its quality. In this approach, planning is seen as a search problem in space of partial plans, allowing many of the classic search algorithms to be applied. A number of route planners in the literature are based on the A^* algorithm [3] or variants such as A^*_ϵ [7]. For example, A^* has been used in a number of CGF systems as the basis of their planning component, to plan road routes [1], avoid moving obstacles [4], avoid static obstacles [5] and to plan concealed routes [6].

However, while such planners are complete and optimal (or optimal to some bound ϵ), it can be difficult to formulate the planning task in terms of minimising a single criterion (cost function). It is rarely the case that we are searching for a plan that is optimal on a single criterion, and it is often more natural to express the problem requirements in terms of *constraints* on the plan. For example, our hide-and-seek agents must generate plans which satisfy a number of criteria, such as the length of the route, whether it is concealed from their opponents, the amount of time or 'energy' required to execute the plan and so on. In many cases an acceptable plan will be constrained to attain some level on one or more of the criteria; for example, if one agent is to intercept another agent, it must be in a given position at or before a particular time. Often there is a preference ordering over such constraints; we may wish to specify that constraints relating to the feasibility of the plan, such as the requirement that the plan should no include any 'no-go' cells (cells which exceed the maximum gradient negotiable by the agent), may be preferred to constraints which specify desirable attributes, such as the requirement that the plan should take no more than x timesteps to execute.

One approach to incorporating multiple criteria into the planning process is to define a cost function for each criterion and and use, e.g. a weighted sum of these functions as the function to be minimised. For example, we can define a 'visibility cost' for being exposed and combine this with cost functions for the length of the plan or the time required to execute the plan, to form a composite function which can be used to evaluate alternative plans. However if one or more of the individual cost functions is non-linear, using weights to determine the relative importance of different constraints

is not straightforward, because when the magnitudes of costs change, the effects of weights vary. The relationship between the weights and the behaviour of the planner is complex, and it is often not clear how the different cost functions should be combined to give the desired behaviour across all magnitude ranges for the costs. This makes it hard to specify what kinds of plans the planner should produce and hard to predict what the planner will do in any given situation. Small changes in the weight of one criterion can result in large changes in the plans generated by the planner. Similarly, changing the cost function for a particular criterion involves changing not only the weight for that cost, but the weights for all the other costs as well. In addition, if different criteria are more or less important in different situations, we need to find sets of weights for each possible set of criteria.

Even if were possible to specify a cost function which represents the constraints on the plan and their relative importance, current planners based on A^* are incapable of 'trading off' slack on one constraint to satisfy another, less important, constraint, since they retain only a single plan to a given point. For example, given constraints c_1, that the time taken to execute the plan p, $g_1(p)$ should be less than t, and c_2, that the percentage of the route which is visible to an opponent $g_2(p)$ is less than v, and plans p_a and p_b to some point n with estimated costs $f_1(p_a) = g_1(p_a)+h_1(n) < t$, $f_2(p_a) = g_2(p_a) + h_2(n) < v$, $f_1(p_b) = g_1(p_b) + h_1(n) < t$, $f_2(p_b) = g_2(p_b) + h_2(n) < v$, where $h_1(n)$ and $h_2(n)$ are heuristic estimates of the value of g_1 and g_2 for the best plan from n to the goal. Suppose $f_1(p_a) < f_1(p_b)$ and $f_2(p_a) > f_2(p_b)$, then if c_1 is more important than c_2, any composite cost function $\sigma = w_1 f_1 + w_2 f_2$ which respects this ordering will prefer p_a to p_b (see Figure 2).

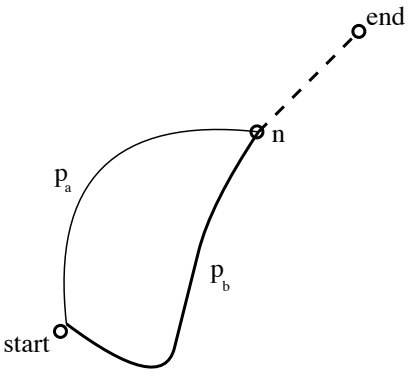


Figure 1: Plan subsumption with A^*

However if it subsequently turns out that $h_2(n)$ was an underestimate of the true cost $h'_2(n)$ of completing the plan from n such that $g_2(p_a) + h'_2(n) > v$ but $g_2(p_b) + h'_2(n) < v$ (i.e. no completion of p_a through n will satisfy the concealment constraint but there exists a completion of p_b which satisfies both constraints), we cannot backtrack to p_b since A^* retains only the (estimated) cheapest solution through n. A^* collapses both costs into a single value which is used to determine both the preference ordering and whether one plan dominates another. The resulting loss of completeness means we cannot use A^* to trade one constraint off against another.

3 Route planning with ordered constraints

In this section we describe a new approach which involves planning to satisfy an ordered set of constraints rather than attempting to find the lowest cost plan to achieve a goal. Instead of using a cost function of n arguments (one for each criterion) which computes e.g. a weighted sum of its inputs, we use a list of constraints where the position of the constraint in the list reflects its importance. In effect, we replace the optimisation problem solved by the planner with a satisficing or constraint satisfaction problem that allows optimisation as a special case.[1] For example, rather than finding the least cost path on the basis of both the time required to execute the plan and the visibility, we might specify a route that takes time less than x and is at least 50% concealed, or that takes time less than y and minimises visibility (subject to the time constraint).[2] This approach provides a means of more clearly specifying agent tasks and more precisely evaluating the resulting plans: a plan can be characterised as satisfying certain constraints and only partially satisfying or not satisfying others. For example, a particular plan might satisfy the requirement that the time taken be less than x, but violate the requirement that the plan be at least 50% concealed.

At present we envisage three main types of constraints:[3]

1. requirements that certain parts of the terrain should be visited or avoided e.g. that the route should not be visible from a given position, or to avoid no-go (i.e. impassable) areas (simple predicates with *true* or *false* values);

2. limits on some property of the plan such as the time required to execute, degree of visibility etc. (functions with values constrained to fall in some interval); and

3. optimisation constraints such as the requirement that the plan should be as short as possible (functions with values to be minimised or maximised, including, for example, a value being as close as possible to some constant).

We represent constraints as bounds on costs. A *cost* is a measure of plan quality relative to some criterion, and can be anything for which an ordering relation can be defined: numbers, booleans etc. A *cost function* is a function from a plan and a terrain model to a cost. A *constraint* is a relation between a cost and a set of acceptable values for the cost, for example the boolean value '*true*', a (possibly open) interval such as '< 10', '$= 100$', or '$\leq O_e + \epsilon$' (i.e. within ϵ of the estimated optimum value O_e, a minimisation constraint). Costs are used to determine if a plan satisfies a constraint, whereas constraints are used to control backtracking.

A (possibly partial) plan which satisfies all the constraints is termed *valid*. The concept of validity is complicated by the difficulty of evaluating a partial plan against the constraints. Constraints are typically properties of a complete plan and are not directly applicable to the partial plans produced by the planner. We therefore use a weaker criterion which allows us to evaluate partial plans: if some completion of a partial plan satisfies the constraints, then the partial plan is deemed to be acceptable. Where the constraint bounds a monotonically increasing function of the plan such as

[1] It is difficult to formulate this problem as a conventional constraint satisfaction problem as we don't know the number of variables (plan steps) in advance.

[2] The notion of 'constraint' developed below is closer to that of Fox [2] than that of e.g. O-Plan [12], though in both cases there are significant differences.

[3] There are also constraints on the planning process itself, e.g. that the planner should should take less than x timesteps to find a plan, but these are the concern of the meta-level planning rules which control the planning process, see section 5 below.

time or distance travelled, this is relatively straightforward. For example if the plan should take less than x timesteps to execute and a partial plan takes $x+1$ timesteps, then it is clear that no extension of the partial plan can ever satisfy the constraint. However in other cases we can't tell until the plan is complete whether the constraint is satisfied. Optimisation constraints introduce further difficulties in that the optimum is usually not known when planning begins; we can only estimate the optimum by attempting to produce a plan, and the current best estimate of the optimum is continually revised throughout the planning process.

In general, demonstrating that it is possible to complete a partial plan so as to satisfy the constraints is of course equivalent to the original planning problem. To avoid this problem, we use the following optimistic policy: if it is not possible to prove that a partial plan cannot satisfy the constraints, we make the assumption that the planner will be able to find a completion of the plan which does satisfy the constraints. Each constraint is associated with a *heuristic function* which returns an estimate of the cost of completing a partial plan. Together with the cost function, the heuristic function can be used to derive an estimated total cost for a plan. By comparing the total cost against the constraint, we can get an idea of whether some completion of the partial plan is *likely* to satisfy the constraint.

If the heuristic functions are admissible, e.g. if they always return an underestimate of the true cost for an upper bound or minimisation constraint, then we can guarantee that if a partial plan fails to satisfy a constraint, all extensions of that plan will also fail to satisfy the constraint, since the cost of the plan can only increase as the plan is extended. Conversely, we don't want the cost functions which greatly underestimate the true cost of the plan, as this will result in the planner being overly optimistic about a plan which will never satisfy the constraint. While admissible cost functions are desirable, they are not necessary. It is enough that the cost functions *generally* underestimate the true cost of a plan to limit the amount of effort wasted on plans which can never satisfy a constraint. If the cost functions are not admissible, we lose any guarantee of optimality, but given our emphasis on satisficing, this is not really a concern.

The planner uses an ordering over plans to direct the search and control backtracking. The search strategy used by the planner is similar to A^*: at each cycle, the planner expands the head of the OPEN list, i.e. the plan with the lowest estimated effort required for completion. Plans are ordered on the basis of the number of important constraints they satisfy. We compare the value of each constraint in the constraint list in turn until we find a constraint which is satisfied by only one of the plans, preferring the plan which satisfies the constraint. This is essentially lexicographic ordering on fixed length boolean strings in which *true* is preferred to *false*. In particular, a plan which satisfies only the first constraint will be preferred to one which satisfies constraints 2–n. For the purposes of comparison, we view the goal as the 0 *th* constraint, i.e. a complete plan which fails to satisfy some of the constraints is preferred to a valid partial plan.[4]

The total ordering on constraints is used to order partial plans into equivalence classes, with those which satisfy all the constraints in the first equivalence class, those that satisfy all but the last constraint in the second equivalence class and so on. By definition, all plans which satisfy a constraint are equally acceptable. However, if the heuristic functions are admissible, the estimated cost of a partial plan will typically increase as the plan gets longer. It is therefore a good idea to have some 'slack' between the cost of the plan and the constraint. We therefore associate each constraint with an *ordering relation* which defines a partial order over the estimated total costs for that

[4]It is clear that, in the general case, this ordering cannot be produced using a weighted sum cost function.

constraint, depending on how well the cost 'satisfies' the constraint. For example if v is a cost value and k_1, k_2 are constants, the following constraints could have the associated orderings:

Form of constraint on cost v	Cost ordering
$v = \mathit{predicate}(\mathit{plan})$	$\mathit{true} < \mathit{false}$
$v < O_e + \epsilon$	$<$
$v < k_1$	$<$
$v > k_1$	$>$
$v = k_1$	$\lvert k_1 - v \rvert$
$k_1 < v < k_2$	$\lvert((k_1 + k_2)/2) - v\rvert$

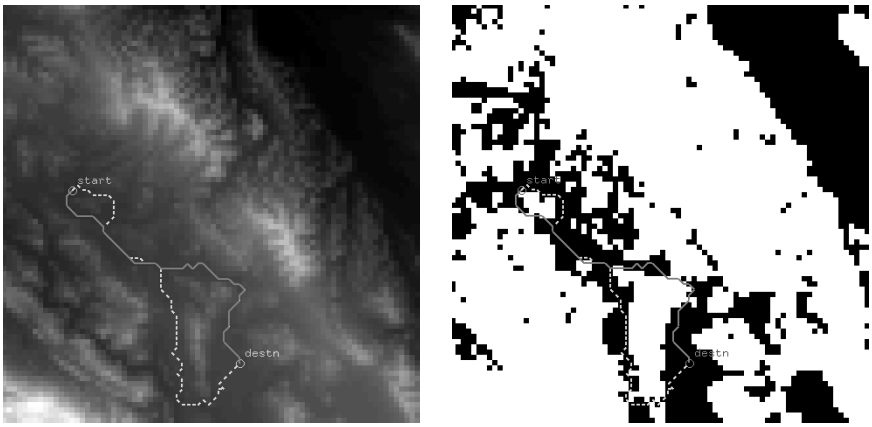
This allows us to sub-order plans within an equivalence class, i.e. how well the plan satisfies the constraint or how close it is to satisfying the constraint. Favouring plans which over-satisfy the constraint reduces the likelihood that the plan will violate the constraint as the length of the plan increases, reducing the amount of backtracking. Conversely, for violated constraints, the sub-ordering favours plans which are closer to satisfying the constraint. This can be useful in the case of 'soft' constraints, where minor violations are acceptable. Moreover, plans which have more slack are often more robust in the face of unexpected problems during execution. Several ordering strategies are possible. For example we could order the equivalence classes using the costs for the most important constraint, or the cost for the most important violated constraint, or all the costs in an additional lexicographic ordering.

4 A simple example

In this section, we illustrate the use of ordered constraints with some example plans produced by the current implementation. The planner currently supports four constraint types: path constraints, no-go constraints, time constraints, and concealed route constraints. A path constraint is a bound on a non-linear 'path cost function' which returns a value expressing the ease with which the plan could be executed. The cost function is based on the 3D distance travelled with an additional non-linear penalty for going uphill. No-go constraints establish an upper bound on the maximum gradient of any cell traversed by the plan. Time constraints establish an upper bound on the time required to execute the plan (or equivalently on the length of the plan), assuming the agent is moving at a constant speed of one cell per timestep. Concealed route constraints enforce a requirement that none of the steps in the model be visible from one or more observation positions.

We consider the problem of planning from coordinates (14, 45) to (46, 12) in an 80×80 grid of spot heights representing a 10km $\times$ 10km region of Southern California.[5] In this example we use only two constraints, a path cost constraint and a no-go constraint. Figure 2(a) shows the terrain model (lighter shades of grey represent higher elevations), and Figure 2(b) shows a no-go map for the same terrain (black cells have a maximum gradient ≤ 0.1, white cells have a gradient > 0.1). In the first case we require that the path cost should be $< 20,000$ and the maximum gradient should not exceed 0.1 (about 6 degrees). These two constraints cannot be jointly satisfied—there is no plan with a path cost $< 20,000$ which has a maximum gradient ≤ 0.1. The

[5] We are grateful to Jeremy Baxter at DERA Malvern for providing the terrain model.



(a) (b)

Figure 2: Planning with two constraints.

planner therefore returns a plan (plan A) which satisfies the first constraint, while minimising the number of violations of the second constraint (plan A has a path cost of 17,997 and violates the no-go constraint five times). This plan is shown as the solid line in Figures 2(a) and 2(b). Note how the route runs through five no-go (white) regions in Figure 2(b). In the second case, we tighten the path cost constraint, requiring that the path cost should be $< 10,000$. Again the two constraints cannot be jointly satisfied, but in this case there is no plan which satisfies the path cost constraint. The planner therefore returns a plan (plan B) which satisfies the second constraint, while coming as close as possible to satisfying the first constraint (plan B has a path cost of 30,242). This plan is shown as the dashed line in Figures 2(a) and 2(b). Note that in this case, the route remains entirely in the 'go' (black) regions in Figure 2(b), passing to the south of the obstruction to do so.

While an A^* planner using a weighted sum cost function could find plan A, it could not find plan B, since the minimum cost plans to the cells around point n (the last common point on plans A and B before the destination) satisfy the first constraint and violate the second constraint, whereas plan B violates the first constraint and satisfies the second constraint and would therefore be discarded. With better heuristics it might be possible for A^* to determine that no completion of plan A through point n would satisfy the path cost constraint, however this is cannot work in general. In this case, the path cost function is non-linear and the only admissible heuristic is a linear estimate.

As might be expected, this additional flexibility involves a certain overhead compared with A^*. The lexicographic ordering of plans requires the comparison of k constraint values for each pair of plans. If we sort within equivalence classes, we must also perform an additional $\log m$ comparisons, where m is the number of plans in the equivalence class. In addition, we must update the constraint values of the plans in the OPEN list when we obtain a better estimate of the optimum value for an optimisation constraint. If the heuristic functions are admissible, any improvement in the estimate of the optimum can only increase it, and any plan that satisfied a constraint will still do so. Similarly, any improvement in the estimate of the optimum can only increase

the amount of slack, as a plan will be closer to the optimum than before. However, plans which didn't satisfy the constraint may come to do so, thereby moving from one equivalence class to another.

There is also a storage overhead associated with this approach. For each plan we must now hold k constraint values in addition to the k costs from which the constraint values are derived. More importantly, we must remember all the non-dominated plans from the start point to each point visited by the planner rather than just the minimum cost plan as with A^* since: (a) it may be necessary to 'trade off' slack on a more important constraint to satisfy another, less important constraint; and (b) it may not be possible to satisfy all the constraints, in which case we must backtrack to a plan in a lower equivalence class. (One plan p_a *dominates* another plan p_b if both plans terminate in the same point, and there is at least one cost f_i such that $f_i(p_a) < f_i(p_b)$ and there is no cost f_j such that $f_j(p_a) > f_j(p_b)$.) Nor can we discard plans after they have been expanded as otherwise we can't check for loops. In some cases remembering all the non-dominated plans can be a significant overhead. However, there are a number of ways round this problem, including more intelligent initial processing of the constraints and discretising the Pareto surface. For example we can require that the planner retain no more than p plans to any given point, by discarding any plan which is similar to an existing plan to that point. (In the limit, this reduces to A^* where we only remember one plan to each point.)

5 Controlling the planner

The architecture of the hide-and-seek agents is based on the general agent architecture described in [10], and consists of three layers: a reactive layer, a deliberative layer and a meta-management layer. The reactive layer contains automatic or pre-attentive processes such as reflexes and the generation of goals in response to changes in the agent or its environment. The deliberative layer contains knowledge-based processes in which options are explicitly considered and evaluated before selection, such as planning, scheduling and decision making. These processes are resource limited; for example, there are only a finite number of goals the agent can attend to at any one time. In the hide-and-seek agents, the deliberative layer consists of three main components: visibility reasoning, belief revision and route planning. The meta-management or reflective layer controls the activities of the deliberative layer, providing global monitoring and 'self-evaluation' functions. The agents are implemented using the SIM_AGENT toolkit [9].

The route planning capabilities of the hide-and-seek agents is distributed across the deliberative and meta-management layers.[6] At the deliberative layer, the route planning component is implemented as a time-sliced constraint-based planner that plans to achieve a single goal at a given level of abstraction and an abstract model generator that can produce a (more) abstract version of a given terrain model. The basic components of the planner are controlled by a collection of meta-management rules, which decide when to plan, what sorts of plans are required and how much effort the agent can afford to spend on planning.

Typically, the agent will have to act before the planner has found a valid or even a complete plan. This can happen when, for example, the time required to produce and execute a plan exceeds the time available to achieve the goal and the agent must plan

[6] At present, the role of the reactive layer in route planning is limited to goal generation.

and act concurrently, or when there is an immediate threat to the agent at its current location. We therefore arrange for the planner to return the best (possibly partial) plan it can find within a given time-slice (typically 200 milliseconds though this will be problem dependent). It is then up to the planning rules at the meta-management level to decide whether the plan is acceptable in the circumstances, in which case the agent can begin execution of the plan, or whether the planner should be allowed to continue searching for a better plan.[7] Thus a decision may be taken to accept a partial plan which violates some constraints, if the agent is pressed for time. Conversely, if the situation allows, the planner can be restarted and run for another time-slice. This approach moves the complex constraint evaluation problem (e.g. how close a constraint is to be being satisfied) which is both constraint specific and context sensitive out of the planner and into the meta-management layer.

6 Conclusions and further work

We have presented a new approach to real-time route planning in continuous terrains based on a general constraint-based search procedure. Our approach has a number of advantages over much of the work found in the literature. By using an ordered set of constraints to represent the requirements on the plan we avoid the difficulties of formulating an appropriate set of weights for a composite cost function. Constraints provide a means of more clearly specifying agent tasks and more precisely evaluating the resulting plans: a plan can be characterised as satisfying some constraints and only partially satisfying or not satisfying others. In addition, the total ordering over constraints blurs the conventional distinction between absolute (hard) constraints and preference (soft) constraints. In our approach, all constraints are preferences that the planner will try to satisfy, but it is up to the agent to decide how important these are in the current context, for example if planning should be terminated if one of the constraints is violated, or if the agent should accept an invalid or incomplete plan when under time pressure.

We currently have an initial implementation of a time-sliced constraint-based planner, based on ABC, which will plan a route from an initial point to a destination point satisfying a number of boolean and interval constraints. However the current implementation does not include all of the features outlined above. In particular optimisation constraints are not supported, and further work is required to complete the implementation and improve its performance. More work is also necessary to characterise the performance implications of ABC relative to A^*.

Another area which we hope to explore is the extension and refinement of the meta-management planning rules which control the basic planner. For example, it would be interesting to investigate utilising information about violated constraints to redefine the problem when an acceptable (e.g. valid) plan cannot be found in a reasonable amount of time. By monitoring the progress of the planner, e.g. the number of constraints satisfied by the current best plan returned at the end of each time-slice, the agent could get some idea of the difficulty of the planning problem. If the planner does not appear to be making progress, e.g. all the plans found so far violate one or more important constraints, the agent could elect to change the order of the constraints, relax one or

[7] Obviously, if we can show that there is no plan which satisfies all or enough of the constraints, there is no point in giving the planner more time to search for a better plan; the only option is to relax one or more of the constraints. However this is difficult to determine without exhaustive search, unless the cost functions are admissible. If the most optimistic estimate of the cost, for all the plans on the OPEN list fail to satisfy the constraint, then the constraint can never be satisfied.

more constraints or even to redefine the goal, before making another attempt to solve the problem. We believe that the separation of the agent's overall planning capabilities into a series of basic components controlled by a collection of planning rules will facilitate the incremental development of additional capabilities and the exploration of more complex real-time planning strategies.

Acknowledgements

We wish to thank Aaron Sloman and the members of the Cognition and Affect and EEBIC (Evolutionary and Emergent Behaviour Intelligence and Computation) groups at the School of Computer Science, University of Birmingham for useful discussions and comments. This research is partially supported by a grant from the Defence Evaluation and Research Agency (DERA Malvern).

References

[1] C. Campbell, R. Hull, E. Root and L. Jackson. Route planning in CCTT, in *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioural Representation*, Technical Report, Institute for Simulation and Training, pp. 233–244, 1995.

[2] M. S. Fox. *Constraint-directed search: a case study of job-shop scheduling*, PhD thesis, Carnegie Mellon University, 1983.

[3] P. E. Hart, N. J. Nilsson and B. Raphael. A Formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.* SSC-4(2), pp. 100–107, 1968.

[4] C. Karr, M. Craft and J. Cisneros. Dynamic obstacle avoidance for Computer Generated Forces, in *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioural Representation*, Technical Report, Institute for Simulation and Training, pp. 245–254, 1995.

[5] C. Karr and S. Rajput. Unit route planning, in *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioural Representation*, Technical Report, Institute for Simulation and Training, pp. 295–304, 1995.

[6] M. Longtin and D. Megherbi. Concealed routes in ModSAF, in *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioural Representation*, Technical Report, Institute for Simulation and Training, pp. 305–314, 1995.

[7] J. Pearl. A^*_ϵ – An algorithm using search effort estimates, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 4, No. 4, pp. 392–399, 1982.

[8] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*, Addison-Wesley, 1984.

[9] A. Sloman, and R. Poli. SIM_AGENT: A toolkit for exploring agent designs, in *Intelligent Agents II: Agent Theories Architectures and Languages*, M. Wooldridge et al (Eds.), Springer-Verlag, pp. 392–407, 1996.

[10] A. Sloman. What sort of control system is able to have a personality?, in *Proceedings Workshop on Designing personalities for synthetic actors*, R. Trappl (Ed.), Vienna, June 1995, (To appear).

[11] A. Stenz. Optimal and efficient path planning for partially known environments, in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1994.

[12] A. Tate, B. Drabble and J. Dalton. Reasoning with Constraints within O-Plan2, in *Proceedings of ARPI Workshop*, Tucson Arizona, Morgan Kaufmann, 1994.