Agent route planning in complex terrains

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

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

Abstract

For many autonomous agents, such as mobile robots, autonomous vehicles and Computer Generated Forces, route planning in complex terrain is a critical task, 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. 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 are difficult to integrate into the architecture of an agent, 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 and describe how a planner based on this approach has been integrated into the architecture of a simple agent. This approach provides a means of more clearly specifying agent tasks and more precisely evaluating the resulting plans as a basis for action.

1 Introduction

Autonomous agents must make decisions in complex, dynamic and uncertain environments in pursuit of multiple, possibly conflicting, goals. For many autonomous agents, such as mobile robots, autonomous vehicles and Computer Generated Forces, route planning in complex terrain is a critical task, 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. For these agents, the route planning task provides a useful framework in which to investigate many of the issues which affect the design of the agent's architecture, including when to plan, what sorts of plans are required and what to do when plans go wrong. The resulting problems of trading off current and future advantage are common to many forms of deliberation, including inference, belief revision and predicting the behaviour of other agents, and solutions to these problems are an essential element in any 'broad' agent architecture. [1]

In our work we are exploring architectures for agents which play variants of the game of 'hide-and-seek' in complex terrains. 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 and describe how a planner based on this approach has been integrated into the architecture of a simple agent. In the next section we briefly outline some of the problems with conventional approaches to route planning based on A^*. In subsequent sections we present a new approach to route planning which generalises the single criterion optimisation problem solved by conventional route planners and 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. We conclude by arguing that constraints form an appropriate interface between the higher-level components of the agent architecture and its basic planning abilities, by providing a means of more clearly specifying the agent's tasks and more precisely evaluating the resulting plans as a basis for action.

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 [5] or variants such as A^*_ϵ [11]. For example, A^* has been used in a number of Computer Generated Forces systems as the basis of their planning component, to plan road routes [2], avoid moving obstacles [6], avoid static obstacles [7] and to plan concealed routes [10].

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 not 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 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 situation.

Even if it 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. A^* retains only the (estimated) cheapest solution through a given point. A^* collapses all costs into a single value which is used to determine both the preference ordering and whether one plan dominates another. The resulting loss of information 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 [9]. 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

[1] It is difficult to formulate this problem as a constraint satisfaction problem [8] given the number of states ($> 100,000$), the length of the plans (> 500 steps), and the softness of the constraints (see below).

[2] The notion of 'constraint' developed below is closer to that of Fox [4] than that of e.g. O-Plan [16] or UMCP [3], though in both cases there are significant differences.

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.

In the case of the hide-and-seek agents, there are three main types of *first-order* constraints:

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).

and also some *second-order* constraints, for example 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 below.

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 model to a cost. Different cost functions use different abstractions of the basic topographic model. For example, the no-go cost of a plan may be computed using a thresholded maximum gradient map, a visibility cost may be computed using a visibility map which represents the degree to which each cell in the model can be seen by opposing agents and so on. 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.

3.1 Valid Plans

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 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 constraints are *admissible*, e.g. if the associated cost function always returns 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.

3.2 Plan ordering

The planner uses an ordering over plans to direct the search and control backtracking. 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*. 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. (It is clear that, in the general case, this ordering cannot be produced using a weighted sum cost function.)

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. We therefore prefer plans which over satisfy the constraints, i.e. where there is some 'slack' between the cost of the plan and the constraint. We associate each constraint with an *ordering relation* which defines a partial order over the estimated total costs for that 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
$v = \textit{predicate}(\textit{plan})$	$true < false$
$v < O_e + \epsilon$	$<$
$v < k_1$	$<$
$v > k_1$	$>$
$v = k_1$	$\lvert k_1 - v \rvert$
$k_1 < v < k_2$	$\lvert((k_1 + k_2)/2) - v\rvert$

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. In our work to date, we have used a lexicographic ordering over costs to sub-order the equivalence classes.

3.3 A^* with Bounded Costs

The search strategy used by the planner is similar to A^*.[3] We use two lists, an OPEN list of unexpanded partial plans, and a CLOSED list which records all non-dominated plans to each point visited by the planner. At each cycle, we expand the plan with the greatest slack in the first non-empty equivalence class. If this is a valid solution and all the constraints are admissible we return the plan and stop. Otherwise we generate all the successors of this plan, and for each successor we cost it and determine its equivalence class. We remove from OPEN and CLOSED all paths dominated by any of the successors of the plan and discard any successor which is dominated by any plan on OPEN or CLOSED. 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 $f_j(p_a) > f_j(p_b)$. We add any remaining successors to OPEN, in order, and recurse (see Figure 1).

In addition, the planner retains a pointer to the best plan found to date, which is returned if the planner is interrupted by a timer or after some pre-determined number of expansions have been performed before a complete, valid plan has been found. If the constraints are not admissible, we can never be sure we have found the best plan without an exhaustive search: even if we have a plan which satisfies all the constraints, there may be another plan with greater slack. In this case it is up the agent to determine if the best plan found so far constitutes an acceptable solution in the current context (see below).

As might be expected, the additional flexibility of ABC 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

[3] ABC is a strict generalisation of A^*; with a single admissible optimisation constraint its behaviour is identical to A^*.

```
OPEN ← [start]
CLOSED ← []

repeat
    if OPEN is empty return false

    remove n, the least member of the first non-empty
    equivalence class, from OPEN and place it on CLOSED

    if n is a solution then return n

    otherwise for every successor, n', of n

        cost n' and determine its equivalence class

        remove from OPEN and CLOSED all paths dominated by n'

        if n' is dominated by any path on OPEN or CLOSED,
        discard n'

        otherwise add n' to OPEN, in order
```

Figure 1: The ABC algorithm

in the equivalence class. In total, we use three orderings: a preference ordering on constraints, a preference ordering on costs and subsumption ordering on costs which is used to compute the set of non-dominated paths to each state. 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. (Unlike A^*, the CLOSED list contains all non-dominated paths to a state, rather than the least cost path to each expanded state.) 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 sufficiently similar to an existing plan to that point. (In the limit, this reduces to A^* where we only remember one plan to each point.)

4 Controlling the planner

The architecture of the hide-and-seek agents is based on the general agent architecture described in [14], and consists of three layers: a reactive layer, a deliberative layer and a reflective or 'meta-management' layer (see Figure 2). 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. For example collision avoidance and simple perceptual processing, including object identification and tracking, are implemented at the reactive layer. 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 reflective layer controls the activities of the deliberative layer, providing global monitoring and 'self-evaluation' functions. For example, the reflective layer is responsible for scheduling competing goals within the agent. The agents are implemented using the SIM_AGENT toolkit [13].

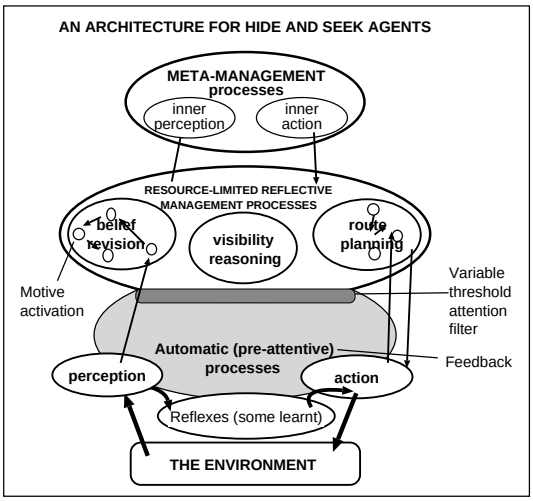


Figure 2: The Agent Architecture

The route planning capabilities of the hide-and-seek agents is distributed across the deliberative and meta-management layers.[4] 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. These basic components are controlled by a collection of planning rules. At the reflective layer, the planning capabilities of the agent 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.

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

4.1 The Deliberative Layer

At the deliberative level, agent tasks are represented as goals. A *goal* is a description of a state s in which certain propositions are true and in which certain actions have been performed. Goals are represented as state descriptions consisting of two parts: a conjunction of propositions which should be true in state s, for example that the agent should be at some location or that the time should not be later than some time t, and a list of actions which should have been performed in some state prior to s, for example that the agent should have observed some location l. In simple cases such goals can be achieved without planning. For example, if the goal is to be at the current or an adjacent position and any actions to be performed can be performed in the current state, then the deliberative layer can simply invoke behaviours defined at the reactive layer, to e.g. move to an adjacent location.

However, in more complex cases, such goals give rise to sub-goals to have a plan, the execution of which will result in the agent being in a state in which the propositions are true and which will allow the actions to be performed during its execution. For example, the task of observing a location to discover if it is occupied becomes a constraint that the plan should pass through at least one cell from which the target location is visible. In some cases, tasks are decomposed into more than one constraint on the plan. For example, 'observing a target location from concealment' is broken down into two constraints: that the plan should pass through at least one (concealed) observation position, and that no step in the plan should be visible from the target position. (We assume that visibility is asymmetric and that certain configurations of the terrain allow an agent to observe a location without itself being seen by an agent at that location. For example, an agent can 'peek' over a ridge to observe another agent in the next valley without itself being seen.)

Route planning goals are generated either in response to changes in the environment which are outside the scope of the simple hard-wired behaviours at the reactive layer or from higher level goals communicated to the agent by other agents. To achieve these goals, the hide-and-seek agents often have to produce plans of several hundred steps at the resolution of the base model. The resulting search problems are intractable, and it is necessary to simplify the problem in order to limit the search. One way to do this is to first generate an abstract plan which can then be refined to give a detailed plan in the base model. If the size of the terrain model exceeds a (context dependent) threshold, the planning rules generate a goal to produce a plan in an abstract model at a larger scale, together with goals to produce scaled versions of the abstract models required by the cost functions and constraints (e.g. no-go and visibility models). If the resulting scaled models are still too large for practical planning, a further abstract plan goal is produced. This process is repeated until the abstract model is small enough to plan in effectively.

When an (abstract) plan is produced at some level of abstraction, this can be used to guide the planning process at the level below. The abstract plan is used to define a 'corridor' within which the planner will search for a refinement of the abstract plan at the next lower level of abstraction. The corridor is itself represented as a constraint, an 'abstract plan constraint', which is simply added to the existing list of constraints at the next lower level of abstraction to give a new planning goal. The position at which the abstract plan constraint is inserted into the original list of constraints determines how important it is to stay within the corridor defined by the abstract plan. For example, if we put the abstract plan constraint first in the list of constraints, the planner will abandon all the other constraints before it leaves the corridor. If we put it last, the

abstract plan constraint is simply advice to the planner, which it may ignore in an attempt to satisfy the other constraints.

The resulting, more detailed, plan is used to construct a new corridor to constrain further refinement at the next lower level of abstraction. Successive refinements may result in repeated displacement of the centreline of the corridor at lower levels of abstraction and helps to eliminate artifacts introduced by the abstraction process.[5]

4.2 The Reflective Layer

At the reflective layer, a collection of meta-management rules monitor the progress of the deliberative layer, and determine the order in which goals, including route planning goals (abstract plan goals, plan refinement goals or plan execution), are processed.

The default strategy is to find a complete plan at one level of abstraction before starting to refine it at a lower level of abstraction. However the real-time demands on the agent and/or uncertainties about the the terrain and the positions and goals of other agents mean that this is often not an appropriate approach. However meta-management rules allow context dependent plan abstraction and refinement, allowing the agent to decide when and how far to abstract, when to accept an abstract plan as the basis of future action, when to start refining the abstract plan and how much of it should be refined to the level of basic actions.

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 then execute a plan exceeds the time available to achieve the goal and the agent must plan 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 is problem and processor 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.[6] If the agent is pressed for time, a decision may be taken to accept a partial plan or a complete plan which violates some constraints as the basis of further plan refinement or action in the environment. Conversely, if the situation allows, the planner can be restarted and run for another time-slice. The architecture allows the agent to plan on several different abstraction levels in parallel while simultaneously executing some initial fragment of the base-level plan, and ensures that any further refinement of an abstract plan is consistent with the already executed portion of the base-level plan.

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. The meta-management rules allow 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 sufficiently to invalidate the current plan.

[5] Other problems caused by abstraction, or averaging, may require task specific abstraction procedures.

[6] 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.

5 A simple example

In this section, we illustrate the use of ordered constraints with two example plans produced by the current implementation. The planner currently supports seven constraint types:

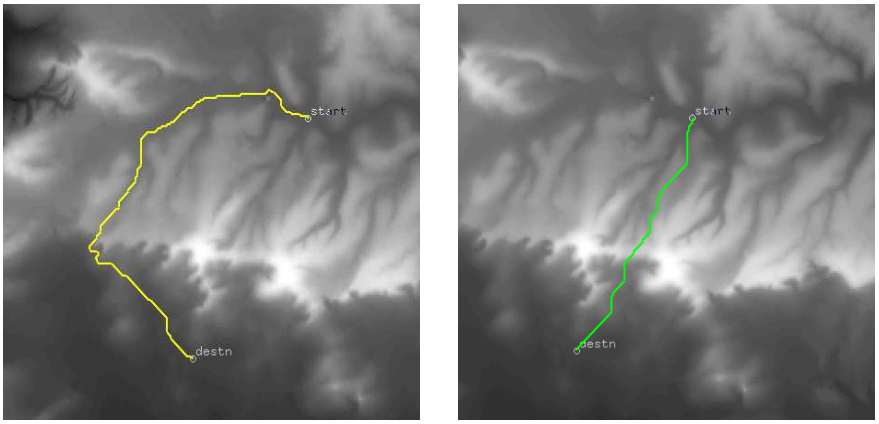
- energy constraints bound a non-linear 'effort' function which returns a value expressing the ease with which the plan could be executed—the cost function is based on the 3D distance travelled with an additional non-linear penalty for going uphill;
- time constraints establish an upper bound on the 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;
- no-go constraints establish an upper bound on the maximum gradient of any cell traversed by the plan;
- concealed route constraints enforce a requirement that none of the steps in the model be visible from one or more observation positions;
- region constraints enforce a requirement that the plan should pass through one or more points in a given circular region;
- observation constraints enforce a requirement that the plan should pass through one or more points from which an agent can observe a target position; and
- concealed observation constraints require that the plan should pass through one or more points from which an agent can observe another agent while remaining concealed from it.

We consider the problem of planning from coordinates (223, 162) to (160, 43) in an 400×400 grid of spot heights representing a 20km $\times$ 20km region of a synthetic terrain model. In this example we use only two constraints, a time constraint and an energy constraint. Figure 3(a) shows an (enlarged) region of the terrain model (lighter shades of grey represent higher elevations).

In the first case we require that the time taken to execute the plan should be less than 500 timesteps $(t < 500)$, i.e. it should not exceed 25km at a constant speed of one cell (50m) per timestep, and the energy cost should be less than 25,000 $(e < 25,000)$. The resulting plan (plan A), shown in Figure 3(a), is 263 steps long (13.15km) and has an energy cost of 24,968, i.e. it just satisfies the energy constraint. A straight line path would have given maximum slack on the first constraint, but the planner has traded slack on the more important constraint to satisfy the second, less important constraint.

Figure 3(b) shows what happens if we relax the energy constraint such that $e < 50,000$. The plan (plan B) now goes over the ridge rather than following a more circuitous route along the river valley. The energy cost has increased to 34,815 but the time taken to execute the plan has reduced. The length is now 7.25km, which is the shortest plan which satisfies the new, relaxed, energy constraint.

Plan B is the is the sort of plan A^* with a weighted sum cost function would produce if the weights were chosen in such a way as to ensure that the time constraint were never violated. In contrast, if it were impossible to satisfy both constraints, e.g. if $t < 250$ and $e < 25,000$, the ABC planner would satisfy the time constraint while coming as close as possible to satisfying the energy constraint.



(a) (b)

Figure 3: Planning with two constraints.

6 Conclusions and further work

We have presented a new approach to real-time route planning in continuous terrains based on a novel constraint-based search procedure and illustrated how this approach can be embedded within an agent architecture. The feasibility of the theoretical ideas is demonstrated by an actual implementation in the SIM_AGENT toolkit.

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. There is a straightforward correspondence between the 'real problem' and the constraints passed to the planner. As a result, it is not necessary to establish that the solution with least cost actually satisfies the constraints on the plan. Changing the relative importance of the criteria or introducing new cost functions or constraints does not require re-computation of weights. 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, trading off slack on a more important constraint to satisfy another, less important, constraint, and 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.

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 (to a greater or lesser degree) and only partially satisfying or not satisfying others. Annotating plans with the constraints they satisfy facilitates the integration of the planner into the architecture of an agent by providing a convenient interface between the condition-action rules that coordinate the agent's behaviours and the functions of the planner. We do not have to choose a cost threshold below which it is safe to start executing a plan; the implications of executing the current best plan are immediately apparent. 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 reflective layer.[7]

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 [9]. However the current implementation does not support optimisation constraints and further work is required to complete the implementation and improve its performance. More work is also necessary to establish the optimality and/or completeness of ABC and to characterise its performance implications 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 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 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. Natasha Alechina read an earlier version of this paper and made many useful comments. This research is partially supported by a grant from the Defence Evaluation and Research Agency (DERA Malvern).

References

[1] J. Bates and A. B. Loyall and W. S. Reilly. Broad agents, *Proceedings AAAI spring symposium on integrated intelligent architectures*, 1991, (reprinted in *Sigart Bulletin*, 2(4), Aug. 1991, pp. 38–40)

[2] 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.

[7] Earlier versions of the hide-and-seek agents incorporated a simple A^*-based route planner as a primitive action, on the assumption that encapsulating the planner would simplify the integration of planning with other behaviours. In practice, this turned out to be too inflexible.

[3] K. Erol, J. Hendler, D. Nau and R. Tsuneto. A Critical Look at Critics in HTN Planning, in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, IJCAI-95*, pp. 1592–1598, 1995.

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

[5] 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.

[6] 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.

[7] 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.

[8] H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI-96*, AAAI Press/MIT Press, pp. 1194–1201, 1995.

[9] B. Logan. Route planning with ordered constraints, in *Proceedings of the 16th Workshop of the UK Planning and Scheduling Special Interest Group, 17-18 December 1997*, University of Durham, UK, 1997, (to appear).

[10] 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.

[11] 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.

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

[13] 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.

[14] A. Sloman. What sort of control system is able to have a personality?, in *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents*, R. Trappl and P. Petta (Eds.), Springer-Verlag (Lecture notes in AI), pp. 166–208, 1997.

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

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