Qualitative Decision Support using Prioritised Soft Constraints

Brian Logan and Aaron Sloman

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

{b.s.logan,a.sloman}@cs.bham.ac.uk

Abstract

A key assumption of all problem-solving approaches based on utility theory is that we can assign a utility or cost to each state. This in turn requires that all criteria of interest can be reduced to a common ratio scale. However, many real-world problems are difficult or impossible to formulate in terms of minimising a single criterion, and it is often more natural to express problem requirements in terms of a set of constraints which a solution should satisfy. In this paper, we present a decision support system for route planning in complex terrains based on a novel constraint-based search procedure, A^* with bounded costs (ABC), which searches for a solution which best satisfies a set of prioritised soft constraints, and illustrate the operation of the system in a simple route planning problem. Our approach provides a means of more clearly specifying problem-solving tasks and more precisely evaluating the resulting solutions as a basis for action.

1 Introduction

A key assumption of all problem-solving approaches based on utility theory is that we can assign a *utility* or *cost* to each state. This in turn requires that all criteria of interest can be reduced to a common ratio scale. For example, in a game of chess it is assumed that all the pieces and their positions on the board can be given a value on a common scale. Similarly, in decision theory, it is assumed that, for example, the inconvenience of carrying an umbrella and the discomfort of getting wet can be expressed as commensurable (dis)utilities. However, many real-world problems are difficult or impossible to formulate in terms of minimising a single criterion. These difficulties are particularly apparent when utility theory is used as the basis of a decision support system and information about relative utilities must be supplied by the user.

In this paper, we consider decision support systems for multi-criteria problems, where the relative importance of criteria change both between problems, and during the problem-solving process as the implications of the user's preferences become apparent. We define a class of generalised constraint-based search problems which admits both prioritised constraints (i.e., constraints which are more or less important) and soft constraints (i.e., constraints which can be satisfied to a greater or lesser degree), and present a generalisation of the well-known A^* search algorithm, A^* with bounded costs (ABC), which searches for a solution which best satisfies a set of prioritised soft constraints. We then briefly describe an implemented route planning system based on

ABC and illustrate the advantages of our approach in a simple route planning problem. We conclude by arguing that constraints form an appropriate interface between the user and the planner, by providing a means of more clearly specifying problem-solving tasks and more precisely evaluating the resulting plans as a basis for action.

2 An example: route planning

As a motivating example, consider the problem of planning a route in a complex terrain consisting of hills, valleys, impassable areas etc. Such problems arise in connection with path planning for a mobile robot, planning routes for forestry operations or planning a route for a new road or railway line among others. In each case, a number of factors will be important in evaluating the quality of a plan: the length of the route; the maximum negotiable gradient; the degree of visual intrusion (in the case of an environmental impact assessment of a new railway line); the degree to which the surrounding terrain is visible from the route (when surveying an area); and so on. In any particular problem, some of these criteria will affect the feasibility of the route, while others are simply preferences. Route planning is a good example of a wide class of decision support tasks, where a range of different criteria must be traded-off against one another to obtain an acceptable solution.

The problem of finding a route in a complex environment can be 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 [1]. In this approach, planning is seen as a search problem in the space of partial plans, allowing many of the classic search algorithms such as A^* [5] or variants such as A^*_ϵ [11] to be applied. 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).

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. In general, the relationship between the weights and the solutions produced is complex, and it 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 a planner should produce and hard to predict what it will do in any given situation; small changes in the weight of one criterion can result in large changes in the resulting plans. 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. Moreover, if different criteria are more or less important in different situations, we need to find sets of weights for each situation.

At best the amount of, e.g., time or energy, we are prepared to sacrifice to remain hidden is context dependent. In general, the criteria which determine the quality of a solution are incommensurable. For example, the alternatives may only be ordered on an ordinal scale, with some of the criteria referring to the feasibility of the plan while others refer to properties that are merely desirable. It is difficult to see how to convert such problems into a multi-criterion optimisation problem without making ad-hoc assumptions.

3 Search with prioritised soft constraints

In many cases an acceptable solution will be constrained to attain some acceptable range of values on one or more of the criteria, and it is often more natural to express the problem requirements in terms of a set of constraints which the solution must satisfy. In the remainder of this paper 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 [8]. 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 order of the constraints reflects their 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. 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 t and is at least 50% concealed, or that takes time less than u and minimises visibility (subject to the time constraint).

This approach provides a means of more clearly specifying problem-solving 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.

We consider three main types of constraint:

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 should 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).

Constraints are represented as bounds on costs.[1] A *cost* is a measure of path quality relative to some criterion, and can be anything for which an ordering relation can be defined: e.g., numbers, booleans, or more generally a label from an ordered set of labels (e.g., 'tiny', 'small', 'medium', 'large', 'huge') etc. A *cost function* is a function from a plan and a terrain 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 from a particular location 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*', '$= 100$', an open interval such as '< 10', '> 20', or '$\leq O + \epsilon$' (i.e. within ϵ of the optimum value *O*). Costs are used to determine if a plan satisfies a constraint, whereas constraints are used to control backtracking.

[1]The notion of 'constraint' developed below is closer to that of Fox [4] than that of e.g. O-Plan [12] or UMCP [3], though in both cases there are significant differences.

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: unless we can 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. If the constraints are *admissible*, e.g. if the associated cost function always returns an underestimate of the true cost of the cheapest completion of a partial plan 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. 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.

3.2 Plan ordering

If the problem is over-constrained, there will be no solution which satisfies all the constraints. In such situations, it is often possible to distinguish among the invalid solutions, as the violation of some (sets of) constraints will be preferable to others. We order plans on the basis of the number of important constraints they satisfy, comparing 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, and preferring the plan which satisfies the constraint. This is essentially lexicographic ordering on fixed length boolean strings in which *true* is preferred to *false*.

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. 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.)

If the problem is under-constrained, there may be many valid solutions. In such cases, we prefer plans which over-satisfy the constraints, i.e., where there is some ‘slack’ between the cost of a path and the bound on the cost defined by a constraint. This is important if we are looking for the optimal solution but also for other reasons. For example, solutions which over-satisfy time or energy constraints are often more robust in the face of unexpected problems during the execution of the plan. 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 a constraint requires that a solution satisfy a certain property and the associated cost function is boolean valued, then the value ‘true’ is preferred to ‘false’. Similarly, if a constraint is to find a route with ‘medium’ or better concealment, then ‘small’ would be preferred to ‘large’. In general, if v_1 and v_2 are values and k_1 is a constant, then the following constraints could have the associated orderings: v_1 is preferred to v_2 if

Form of constraint on cost v	Cost ordering
$v < O_e + \epsilon$	$v_1 < v_2$
$v < k_1$	$v_1 < v_2$
$v > k_1$	$v_1 > v_2$
$v = k_1$	$\lvert k_1 - v_1 \rvert < \lvert k_1 - v_2 \rvert$

These *slack orderings* allow us to sub-order plans within an equivalence class, with those plans which have the greatest slack being the most preferred. Conversely, for violated constraints, the sub-ordering may favour plans which are closer to satisfying the constraint. This can be useful in the case of 'soft' constraints, where minor violations are acceptable.[2] 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^*. We use two lists, an OPEN list of unexpanded nodes (partial plans), and a CLOSED list which records all non-dominated plans to each point visited by the planner. At each cycle, we take the node with the greatest slack in the first non-empty equivalence class from the OPEN list and put it on CLOSED. Call this node n. If n is a valid solution and all the constraints are admissible we return the plan and stop. Otherwise we generate all the successors of n, 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 n and discard any successor which is dominated by any path 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.

If the constraints are admissible, the first complete plan found will satisfy the greatest number of most important constraints; if slack ordering is used, this plan also has greatest slack. 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. The planner retains a pointer to the best plan found to date, which is returned if the planner is interrupted by the user or after some pre-determined number of expansions have been performed before a complete, valid plan has been found. The plans returned by the planner are *annotated* with the constraints they satisfy and the amount of slack for each constraint and the user can use these annotations to to determine if the best plan found so far constitutes an acceptable solution in the current context (see below section 4 below).

Given reasonable assumptions regarding the constraints, ABC is both complete and optimal [9]. However, 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

[2]Favouring plans which over-satisfy the constraints has the additional advantage of reducing the likelihood that the plan will violate the constraint as the length of the plan increases, reducing the amount of backtracking. (If the cost functions are admissible, the estimated cost of a plan will typically increase as the plan is expanded.)

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.

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. 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 Route planning with ordered constraints

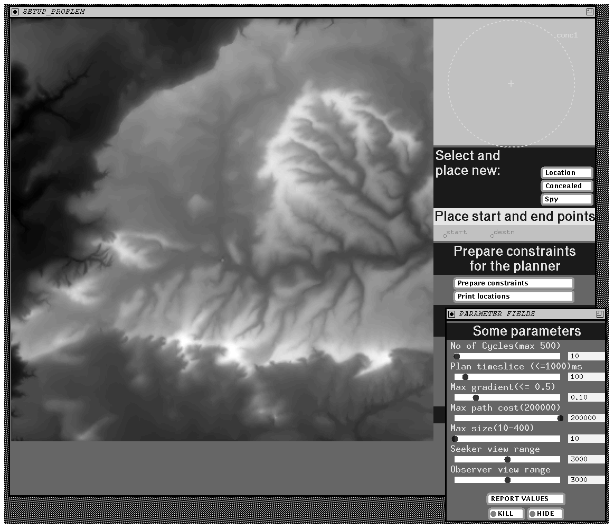
In this section, we present an example application of the ABC algorithm and illustrate the flexibility of our approach compared to conventional approaches based on weighted sum cost functions. We briefly describe a simple decision support system for route planning based on ABC and illustrate its operation in a simple route planning problem.[3] The decision support system 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 planner currently supports six constraint types:

- energy constraints bound a non-linear 'effort' function which returns a value expressing the ease with which the plan could be executed;
- time constraints bound the time required to execute the plan, assuming the agent is moving at a constant speed of one cell per timestep;
- no-go constraints bound the maximum gradient of any cell traversed by the plan;
- region constraints require that the plan should pass through one or more points in a given circular region;
- concealed route constraints require that none of the steps in the plan be visible from one or more observation positions; and
- observation constraints require that the plan should pass through one or more points from which an agent can observe a target position.

The interface to the system consists of two main windows or control panels as shown in Figure 1. The larger window shows the currently selected terrain model, the start and end positions of the plan, graphical representations of positional constraints,

[3]The application described in this section is for illustrative purposes only, and ignores many criteria that a real route planning system would have to address, e.g., minimum radius of curvature.

and the output of the planner in the form of one or more plans and trace information about progress on the current problem. The second, smaller, window (inset at bottom right) is used to set constraint parameters and to control the behaviour of the planner, e.g., to decide whether hierarchical planning should be used. (Additional windows showing models and plans at different levels of abstraction can be displayed if the user chooses to use hierarchical planning.) A third window (not shown in Figure 1) is used to select the terrain model and control the output of trace information.



```
No of Cycles(max 500)          10
Plan timeslice (<=1000)ms      100
Max gradient(<= 0.5)           0.10
Max path cost(200000)          200000
Max size(10-400)               10
Seeker view range              3000
Observer view range            3000
```

Figure 1: Planner control panels.

Constraints which take a location in the terrain as a parameter (i.e., region, concealed route and observation constraints) are specified graphically, by creating an instance of the constraint type and placing its graphical representation at the appropriate location on the model. Other parameters, e.g., the visual range of an observer, and the parameters for non-positional constraints (i.e., energy, time and no-go constraints) are specified using the parameter control panel. There is a default ordering over the constraints, with feasibility (no-go) constraints being the most important, followed by positional constraints, and finally preference constraints (energy and time), however the default ordering can be over-ridden by the user. If there are several positional constraints, their ordering is also specified graphically.

The start and end position, together with any constraints specified by the user, define the goal which is passed to the planner. To achieve these goals, the planner often has 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 is then refined to give a detailed plan in the base model. If the size of the terrain model exceeds a user-selectable threshold, the planner will attempt to find a plan in an abstract model at a larger scale. If the resulting scaled model is still too large for practical planning, the planner will attempt to find a plan in an abstraction of the scaled model, and so on. This process is repeated until the abstract model is small enough to plan in effectively.

The resulting abstract plans are used to guide the planning process at the level below. Each 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.[4]

4.1 An example plan

As an example of the operation of the system, 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 (shown in Figure 1). Figure 2 shows an (enlarged) region of the terrain model (lighter shades of grey represent higher elevations). In this example we use only two constraints: that the time taken to execute the plan should be less than 500 timesteps $(t < 500)$, and that the energy cost should be less than 25,000 $(e < 25,000)$. There is a conflict between the two constraints, in that shorter plans involve traversing steeper gradients and so require more energy to execute.

Figure 2 shows the plan returned by the ABC planner. It requires 263 timesteps and 24,968 units of energy to execute, i.e. it just satisfies the energy constraint. A straight line path would have given maximum slack on the first (time) constraint, but the planner has traded slack on the more important constraint by following a more circuitous route along the river valley to satisfy the second, less important constraint (energy).

While trading off slack on the first constraint to satisfy the second constraint, the planner will *never* prefer a path which satisfies only the second constraint to a path which satisfies the first constraint. To obtain the same behaviour with a weighted sum cost function of the form $w_1 t + w_2 e$ we must ensure that the ratio of w_1 to w_2 is greater than the maximal value of $|e(p_a) - e(p_b)|/|t(p_a) - t(p_b)|$ for any two plans p_a and p_b. But then a planner minimising $w_1 t + w_2 e$ will never trade off slack on the first constraint to satisfy the second one. In contrast, if it were impossible to satisfy both constraints, e.g. for the constraints $t < 250$ and $e < 25,000$, the ABC planner would satisfy the time constraint while coming as close as possible to satisfying the energy constraint.

5 Related work

Our work has similarities with work in both optimisation (e.g., heuristic search for path finding problems and decision theoretic approaches to planning) and constraint satisfaction (e.g., planning as satisfiability).

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

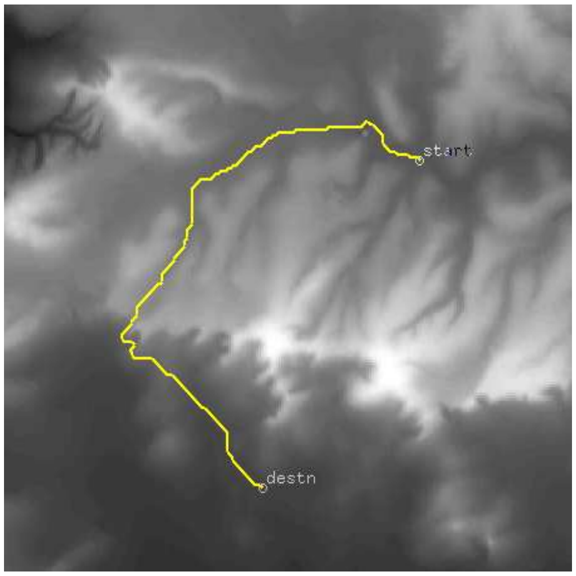


Figure 2: Planning with two constraints.

ABC is a strict generalisation of A^*: with a single optimisation constraint its behaviour is identical to A^*. However unlike heuristic search and decision theoretic approaches, we do not require that all the criteria be commensurable. The emphasis on non-dominated solutions has some similarities with Pareto optimisation which also avoids the problem of devising an appropriate set of weights for a composite cost function. However the motivation is different: the aim of Pareto optimisation is to return some or all of the non-dominated solutions for further consideration by a human decision maker. In contrast, when slack ordering is used, ABC will return the most preferred solution from the region of the Pareto surface bounded by the the constraints which are satisfied in the highest equivalence class. If an optimal solution is not required (i.e., a slack ordering is not used), the algorithm will return any solution which satisfies the constraints; such a solution will not necessarily be Pareto optimal.

ABC also has a number of features in common with constraint satisfaction techniques. However, algorithms for boolean CSPs assume that: (a) all constraints are boolean (i.e., they are either true or false), (b) all constraints are equally important (i.e., the solution to an over-constrained CSP is not defined), and (c) the number of variables is known in advance. Dubois et al. [2] introduce Fuzzy Constraint Satisfaction Problems (FCSP), a generalisation of boolean CSPs, which support prioritisation of constraints and preference among feasible solutions. In addition, FCSPs allow uncertainty in parameter values and ill-defined CSPs where the set of constraints which define the problem is not precisely known. However, in common with more conventional techniques, their approach assumes that the number of variables is known in advance. For many problems, this assumption is invalid, for example, in route planning the number of steps in the plan is not normally known in advance. Several authors, for example [6, 7], have described iterative techniques which can be applied when the number of variables is unknown. However the problems to which these techniques have been applied are considerably smaller than the route planning problems to which ABC has been applied which typically involve more than 100,000 states and plans of more than 500 steps. Moreover these approaches are incapable of handling prioritised or soft constraints.

ABC has many of the advantages of FCSPs and iterative techniques: it can handle prioritised and soft constraints (though not uncertain values or cases in which the set of constraints which define the problem is not precisely known) and problems where the number of variables is not known in advance.

6 Conclusions and further work

In this paper, we have argued that classic search algorithms make very strong assumptions, such as the assumption of commensurability, which do not hold for many real-world problems. We have presented a new approach to formulating and solving multi-criterion search problems which relaxes some of these assumptions.

By using an ordered set of prioritised soft constraints to represent the requirements on the solution we avoid the difficulties of formulating an appropriate set of weights for a composite cost function. Changing the relative importance of the criteria or introducing new cost functions or constraints does not require re-computation of weights. The ordering over constraints blurs the conventional distinction between absolute (hard) constraints and preference (soft) constraints. In our approach, all constraints are preferences that the problem-solver will try to satisfy, trading off slack on a more important constraint to satisfy another, less important, constraint, and it is up to the user to decide how important these are in the current context, for example if planning should be terminated if one of the constraints is violated.

Constraints provide a means of more clearly specifying problem-solving tasks and more precisely evaluating the resulting solutions. There is a straightforward correspondence between the ‘real problem’ and the constraints passed to the problem-solver. A solution 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 means that the implications of adopting or executing the current best plan are immediately apparent, and facilitates the integration of the planner into a decision support system (or the architecture of an agent, see for example [10]) by providing a convenient interface between the user’s problem and the functions of the planner. If a satisfactory solution cannot be found, the degree to which the various constraints are satisfied or violated by the best plan found so far can be used to decide whether 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. Our approach moves the complex constraint evaluation problem which is both constraint specific and context sensitive out of the planner and into the decision support system.

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 [8]. 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 characterise the performance implications of ABC relative to A^*. Another area which we wish to explore is that of mixed initiative planning. For example, it would be straightforward to allow the user to ‘sketch’ an initial plan, either directly on the base model, or in one of the abstract models produced by the planner, and use this to generate an abstract plan constraint to guide subsequent search by the planner.

The present work is the first step in the development of a hybrid approach to search with prioritised soft constraints. It raises many new issues related to preference orderings over solutions ('slack ordering') and the relevance of different constraint orderings for different kinds of problems. However, we believe that the increase in flexibility of our approach outweighs the increase in computational cost associated with ABC.

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] 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*, pages 233–244. Institute for Simulation and Training, 1995.

[2] D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[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*, pages 1592–1598, 1995.

[4] M. 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):100–107, 1968.

[6] 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*, pages 1194–1201. AAAI Press/MIT Press, 1996.

[7] V. Liatsos and B. Richards. Least commitment—an optimal planning strategy. In *Proceedings of the 16th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 119–133. University of Durham, Dec 1997.

[8] B. Logan. Route planning with ordered constraints. In *Proceedings of the 16th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 133–144. University of Durham, Dec 1997.

[9] B. Logan and N. Alechina. A^* with bounded costs. Technical Report CSRP-98-3, School of Computer Science, University of Birmingham, 1998.

[10] B. Logan and A. Sloman. Agent route planning in complex terrains. Technical Report CSRP-97-30, School of Computer Science, University of Birmingham, 1997.

[11] J. Pearl. A^*_ϵ – an algorithm using search effort estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(4):392–399, 1982.

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