First Year Report: Model-based Planning

Allan M. Bruce

Department of Computing Science, University of Aberdeen
July 19th 2004

Abstract

This report introduces a novel approach to the execution of qualitative reasoning, by determining the stages that can be executed in parallel. The development of such a reasoner is underway and is intended to be incorporated into a distributed computing network known as the GRID. Once this reasoner has been developed, it will provide the core reasoning engine for a model-based planner. Model-based planners are particularly suited to changing worlds as they do not require explicit knowledge of the current state. A model-based planner can instead determine how to achieve a goal from a state which had not been considered previously.

Table of Contents

1 Introduction

The field of qualitative reasoning is very interesting. It was developed using ideas from Naïve physics and common-sense reasoning. QR has been the area of a lot of research since, and a lot of interesting developments have come about. Some reasoners incorporate fuzzy reasoning which allows a semi-quantitative approach and makes temporal calculations easier. Until recently these reasoners have been developed in languages such as LISP or Prolog, which limits their appeal to a limited number of people. This has also restricted their modularity, particularly interfacing with other programs and technologies. Developing a reasoner in Java allows this interfacing, enhances portability and does not limit the target audience. One technology which is still young in development, but very interesting lies in the field of e-science, known as grid computing. These distributed computing networks allow programs to run in parallel on different systems within the network, allowing the performance of supercomputers to be met and available to a wider range of clients.

One of the main inspirations for developing a novel architecture for qualitative reasoning is to develop a new model-based planner. Model-based planning has been successful. Using a constraint-based qualitative reasoner is thought to make the planner more applicable to a larger set of problems. The grid computing also allows this planner to execute in a parallel manner which should increase the performance greatly.

This report introduces the fields of QR and planning, and details the work completed so far in developing a new architecture for Morven, a fuzzy qualitative reasoner.

2 Qualitative Reasoning

Qualitative Reasoning is one area of many in Artificial Intelligence. The inspiration behind QR is to reason in a manner similar to a human's common sense. This allows systems to be analysed with only incomplete knowledge or information that is not very accurate. The basic idea of QR is to perform analysis with symbols, or ranges of numbers rather than precise information. In its simplest form, QR labels 3 distinct symbols, negative, zero and positive (-, 0, +). These can be used to determine a variables state, but extra information is useful. Cellier[1] describes an example using this method, but also giving information about the derivatives to aid in the analysis. Morgan[2] found that extending the derivative information to include further derivatives was also advantageous. These extra derivatives, known as Vector Envisionment, were studied by Coghill[3], and it was found that more than three levels (that is the 0^{th} derivative, the 1^{st} and 2^{nd} derivatives) did not aid in describing curves, as the naked eye cannot distinguish the extra derivative information.

2.1 Existing Engines

2.1.1 QSIM

QSIM is one of the most well used and highly developed QR engines[4]. QSIM was originally developed by Kuipers[5] as a constraint based qualitative simulation engine. Models in QSIM are described by qualitative differential equations. These equations are an abstraction of differential equations where the variables are qualitative. This requires knowledge of the behaviour of the system being modelled, if this is not known then basic constraints can also be used, for example monotonic functions, and algebraic functions. Variable *x* is said to be monotonically increasing with *y* if an increase in *y* results in an increase in *x* throughout the solution space.

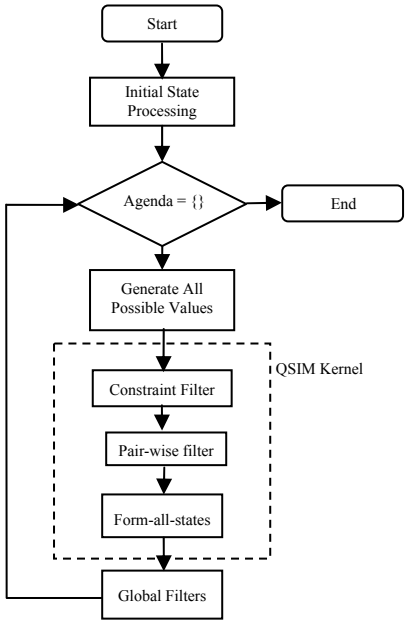

Figure 1: QSIM Algorithm

During execution, QSIM generates all possible values of variables, and uses a constraint filter, and pair wise filtering[6] to discard inconsistent values, thus QSIM utilizes a non-constructive algorithm. The core QSIM algorithm is shown in figure 1.

2.1.2 Fuzzy Reasoners

2.1.3 FuSim

Both Fuzzy Reasoning and Qualitative Reasoning allow problems to be analysed with uncertainty. Shen & Leitch[7] decided to combine these two influences and created FuSim. FuSim uses Fuzzy Numbers (instead of the quantity space mentioned above: -, 0, +), which allows a system to have more information in the quantity space if available resulting in a more accurate analysis. Fuzzy numbers are represented as 4-tuples: *a*, *b*, α, β. The region between *a* and *b* is where the fuzzy number is 100% true. α specifies the size of the region less than *a* that the fuzzy set is neither true nor false, but has a degree of truth. This degree is linear from (*a*, 100%) to (*a*- α, 0%) - Similarly for the region from *b* to *b*+β. Figure 2 shows a diagram of a fuzzy-tuple.

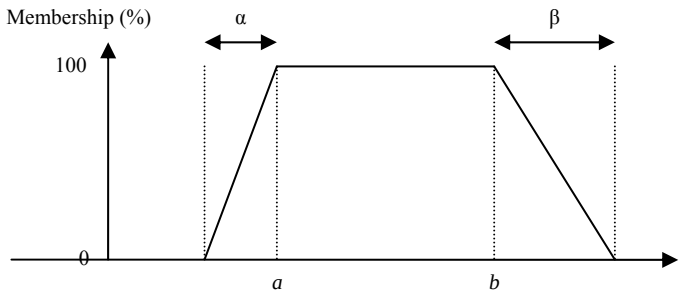

Figure 2: Fuzzy 4-tuple

It is possible to also set up 3 tuples to behave like a symbolic reasoner i.e.:

1. have one tuple that covers the complete negative range, with (α,β) = 0
2. a tuple with (a,b,α,β) = 0
3. have one tuple that covers the complete positive range, with (α,β) = 0

FuSim had a few disadvantages, however. The temporal filters do not provide additional filtering over the constraint and pair-wise filters. FuSim only uses the 0^{th} and 1^{st} derivatives as in QSIM which is restrictive.

2.1.4 Morven

Coghill[4] developed a qualitative reasoning framework using fuzzy numbers called Morven. Morven combined the advantages of several existing technologies including FuSim and Vector Envisionment. Being a framework, Morven could be used to create envisionments or simulations in both synchronous and asynchronous modes. Morven was found to be successful, but lacked portability being developed in LISP. Morven is the main inspiration for the work detailed in this report.

3 Morven Revisited

Following on from the LISP version of Morven, the new version is intended to be a qualitative reasoning engine which uses fuzzy numbers fully to describe the quantity space(s) and also in calculation of state generation & state transitions. Quantity spaces are fully defined by the user, therefore as many fuzzy tuples can be specified as required. Quantity spaces are also set at a per-derivative level, enabling the user to specify more information if available. Morven uses differential planes which are specified in the model, therefore allowing the user to specify how many derivatives to calculate, similar to Vector Envisionment. Morven is a constraint-based qualitative reasoner, which requires constraints to be specified as qualitative differential equations, similar to QSIM. These are specified per differential plane.

3.1 Novel Features

3.1.1 Portability

Morven is being completely developed in Java, allowing it to be run across many platforms easily. Object-oriented programming also allows Morven to be easily updated in the future, and maintainable.

3.1.2 Parallelization

Platzner & Rinner first considered parallelization in QSIM[6]. They found that the tuple filters in QSIM could be run independently of each other and therefore could be run in parallel. They noticed almost an order of magnitude performance increase when using 7 processors running in parallel.

This inspired an idea to increase the amount of parallel calculations within a qualitative reasoner. Morven also executes the transition analysis and state generation phases in parallel, and is scaleable to run on very large systems.

Platzner and Rinner used a dedicated hardware architecture which constricts the implementation. Morven, instead, uses an abstract architecture that can run on almost any system. Figure 3 shows a diagram of the basic architecture used in Morven.

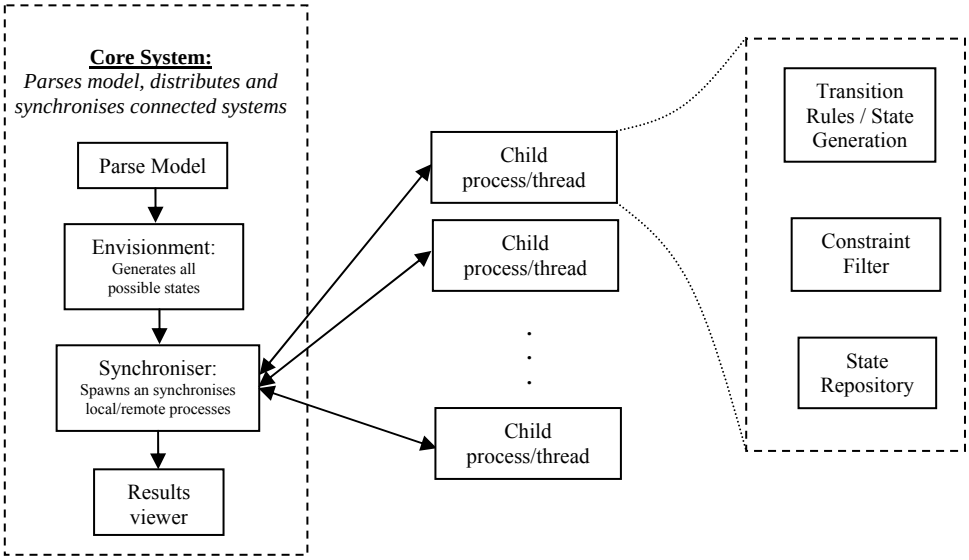

Figure 3: Morven Architecture
Morven spawns processes on remote machines if connected to the GRID, otherwise threads are created on the local machine. Each child process/thread contains its own transition rules, constraint filter and quantity space repository depending on the required type of process.

3.1.3 Constraint Filter

QSIM and many other qualitative reasoners adopt an approach similar to Davis[8] for the constraint filter, which utilises a nested loop. The outer loop continues until no changes in the constraints are seen. The inner loop iterates through all the constraints and checks for inconsistencies.

Morven just requires one pass for the constraint filter, and this loop can be run in a parallel manner. First, all the possible states are generated using transition rules, then each state is considered in turn and checked for inconsistencies. If a state is found to be inconsistent, it is immediately discarded. See figure 4 for a comparison of the two approaches. In addition to the

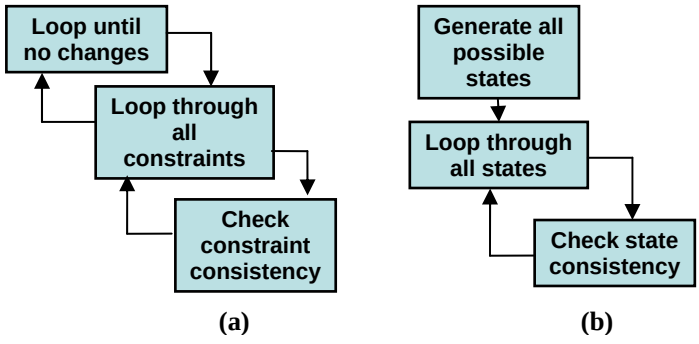

Figure 4: Constraint Filters
(a) Davis approach uses a nested loop whereas (b) Morven uses a single loop to filter the constraints.

constraint filter, the user can also specify the range a variable may take, thus allowing extra states to be trimmed (e.g. a negative Volume is impossible, and therefore any states that may have this may be discarded immediately).

3.2 Future Work

To take advantage of the parallelization of Morven, it is the intention to utilise a distributed computing network called the GRID. The Globus Toolkit allows easy use of the network which allows programs to run in parallel on different machines worldwide. This should allow Morven to run far faster than any other qualitative reasoner around today. It will be interesting to see the speed comparisons between running Morven on a single processor machine, a multi-processor machine and on several machines world-wide using the GRID.

Morven is also intended to be the core reasoner for a model-based planner. The next section discusses the field of planning and the inspirations for developing a model-based planner.

4 Planning

Using Morven as the core qualitative reasoner, a model-based planner is intended to be developed. Below is a brief description of planning, and in particular model-based planning.

Planning is another field of Artificial Intelligence, however unlike qualitative reasoning, planning is a mature area of study and research still continues into better planning theory. The aim of a planner is to construct a plan to solve a given problem from a set of actions and their consequences. For example, if a planner was given the problem to buy milk and walk the dog, and it had (amongst many others) the actions GoToShop, PurchaseMilk, then it should be able to construct the sub-plan to GoToShop and then PurchaseMilk. Plans are rarely this simple, they often contain many solutions for a single sub-problem, and some sub-problems may be executable independently of one another, allowing parallelization.

Planners have similarities with problem solvers, but differ in the representation of actions, states and goals[13]. (Russell and Norvig[13] continues to give a detailed example of how these differ - the reader is directed to §11.2 of the reference).

In the buying milk example, a planner knows that the goal state includes *Have(Milk)* and the planner should have include in the knowledge, that *Buy(x)* results in *Have(x)*, therefore the planner would reason that it requires to *Buy(Milk)* without considering any other unnecessary actions.

The planner may also add these actions anywhere in the plan where needed unlike a problem solver which adds them as a sequence of actions from the initial state. This means that a planner would know that to be able to *Buy(x)* it would have to be *At(Supermarket)*. Any state that included *At (Supermarket)* could be used regardless of whether the planner already has other sub-goals achieved. A problem solver would have to consider the state containing *At(Supermarket)* with all other sub-goals to be able to know to *Buy(Milk)*. This reduces the branching factor considerably of the planner's search space considerably.

A planner regards most parts of the world independent, and can therefore split the large problem into sub-problems (known as divide-and-conquer). The buying milk example would be split into two sub-goals:

1. Buy some milk
2. Walk the dog

Problem solvers would have to consider a course of action that solved both of these conditions in one problem space hence increasing the search space.
Most planners also include some mechanism of execution monitoring, which allows the planner to know when a stage of the plan has failed or the world has changed and it requires re-planning.
Planners are free to come to their solutions in any way as long as they produce a solution to the planning problem. One way of arriving at the solution is to use regression from the goal state. From the goal state, the planner should only add steps that achieve a precondition which is not yet met. If one of these steps may be breached then the planner protects them to safeguard against another step interfering and deleting the precondition. All causal links are automatically protected as they satisfy the requirement for protection. If any of the protected links are threatened by other steps then an ordering constrained is used to ensure that the preconditions are not breached. This ordering may be a promotion or demotion[14] depending if the threatening step is placed after or before the protected link respectively. If the planner is unable to achieve promotion or demotion of such steps then it should try a different choice at some earlier point.

4.1 Graphplan

Amongst the planners available, Graphplan[12] is possibly the most successful planner, and has been the basis of many research activities[9-11]. The main idea for Graphplan developed from graph theory. Graphplan constructs a compact planning graph (a directed levelled graph) based on domain information, goals and initial conditions of a given problem. There are three types of edges and two types of nodes in the planning graph. These are:

Nodes:

- Proposition nodes

- Action nodes

Edges

- Precondition-edges
- Add-edges
- Delete-edges

Each level of the planning graph is split into two stages, the pre-condition stage and the action stage, each with the corresponding type of node. These nodes are connected by either of the edges. Each action node is connected to its pre-condition with a precondition-edge. Each action may have add-effects, therefore will be connected to the next level with an add-edge and similarly for delete-effects and delete-edges. Figure 5 shows a simple example of a planning graph for the rocket domain[12].

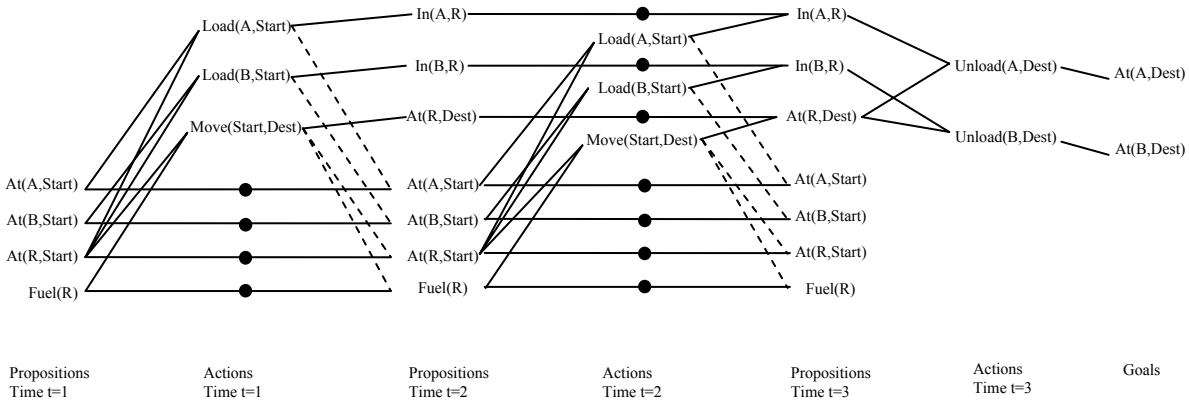

Figure 5: Planning Graph of the rocket domain

A domain that has 4 objects: a rocket (R), some fuel and two pieces of cargo (A and B). Actions in the domain include Load, Unload, Fuel, and Move. Delete-edges are represented by a dashed line and no-operations by a line with a dot.

Graphplan uses the planning graph to guide its search to obtain a partial-order plan. Graphplan can also determine sections of the planning graph that are mutually exclusive of each other, allowing the search to be further reduced, hence increasing efficiency.

4.2 *M*odel-based Planning

Traditional planners use execution monitoring to determine if the world has changed or any stage of the plan has failed. Re-planning is then required to achieve the goal. Traditional planners also rely that the problem space is static[15], i.e. worlds that do not change without the planner's knowledge. This results in most planners being unsuitable for real worlds, and especially in the world of robotics. Model-based planners do not rely on static worlds and can instead reason about their actions and the consequences. This is particularly useful in changing environments. Model-based planners do not require explicit information of actions, as they use the internal model to evaluate the results of such actions, e.g. an action may be to boil water. A traditional planner would set water on the heat for a set pre-calculated time. Instead, a model-based planner reasons about the process of heating water, and can determine how long to boil for, and also more importantly determine if the heat is enough to make the water boil.

Another example: Suggest that a cooling plant requires a valve to be adjusted to allow a certain flow of water through a cooling pipe. A traditional planner would have to have pre-planned all possible contingencies in case they occur, whereas a model-based planner can reason that if the cooling isn't sufficient then opening the valve further is required, or if there is too much cooling then closing the valve would solve the problem. In fact, with a qualitative reasoning engine within the model-based planner it could determine that if a little more cooling is required, then the valve requires to be opened a little more. This is the motivation of the main work for the authors PhD.

4.2.1 Excalibur

Model-based planners already exist, the most notable being Drabble's Excalibur[15]. Excalibur uses Forbus' Qualitative Process Theory[16] to model the internal representation of the world. This system reasons about processes in a manner similar to the boiling water example above.

Excalibur has the same stages as a normal planner but also includes a plan reasoner and simulation of the world which predicts the state of the real world after the action of the planner has been executed. These are linked by a plan co-ordinator which

passes action queries from the planner to the plan reasoner and replanning requests from the plan reasoner to the planner if the action is unsuitable.

Excalibur has been used mainly in robotics and autonomous spacecraft control, but has also been used as an online tutor and assistant in the classroom. Excalibur uses a model based on the process ontology. Using the constraint ontology allows a wider set of problems to be modelled.

5 Future Work

As mentioned in §2.2, the fuzzy qualitative reasoner Morven has been successfully developed to include parallelization in several stages. It is proposed to incorporate these ideas into a distributed computing network known as the GRID. With this complete, this will introduce a novel approach to qualitative reasoners.

One use of this new qualitative reasoner is to develop a model-based planner using Morven as the core reasoner.

Existing model-based planners are based on old planners and Excalibur used a reasoner based on Qualitative Process Theory. It is believed that a planner based on Graphplan and using the fuzzy qualitative reasoner Morven would introduce a novel and exciting model-based planner to the planning community.

6 References

[1] Cellier F., *Continuous System Modelling*, Springer-Verlag 1991.

[2] Morgan A., *The Qualitative Behaviour of Dynamical Physical Systems*, PhD Thesis, University of Cambridge, November 1988.

[3] Coghill G., *Vector Envisionment of Compartmental Systems*, M.Sc. Thesis, University of Glasgow, April 1992.

[4] Coghill G., *Mycroft: A Framework for Constraint Based Fuzzy Qualitative Reasoning*, PhD Thesis, Heriot-Watt University, September 1996.

[5] Kuipers, B., "Qualitative Simulation", *Artificial Intelligence*, **29**(3), 289-338, September 1986.

[6] Platzner M. & Rinner B., "Toward Embedded Qualitative Simulation: A Specialized Computer Architecture for QSIM", *IEEE Intelligent Systems*, **15**(2), 62-68, March/April 2000.

[7] Shen Q. & Leitch R., "Fuzzy Qualitative Simulation", *IEEE Transactions on Systems, Man and Cybernetics*, **23**(4), 1038-1061, July-August 1993.

[8] Davis E., "Constraint Propagation with Interval Labels", *Artificial Intelligence*, **32**(3), 281-331, July 1987.

[9] Smith D. & Weld D., "Temporal Planning with Mutual Exclusion Reasoning", *Proceedings of the 16th International Conference on AI*, 1999.

[10] Krogt R. et al, "Exploiting Opportunities Using Planning Graphs", *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, December 2003.

[11] D. Weld, et al., "Extending Graphplan to Handle Uncertainty and Sensing Actions", *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, July 1998.

[12] Blum A. & Merrick F., "Fast Planning Through Planning Graph Analysis", *Artificial Intelligence*, **90**, 281-300, 1997.

[13] Russell S. & Norvig P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1st Edition – 1995.

[14] Kleer J. de & Williams B., "Diagnosing Multiple Faults", *Artificial Intelligence*, **32**(1), 91-130, April 1987.

[15] Drabble B., "Excalibur: A Program for Planning and Reasoning with Processes", *Artificial Intelligence*, **62**(1), 1-40, July 1993.

[16] Forbus K., "Qualitative Process Theory", *Artificial Intelligence*, **24**, 85-168, December 1984.