ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-915

Qualitative Simulation with Answer Set Programming

Timothy Wiley[1], **Claude Sammut**[1] and **Ivan Bratko**[2]

Abstract. Qualitative Simulation (QSIM) reasons about the behaviour of dynamic physical systems as they evolve over time. The system is represented by a coarse qualitative model rather than precise numerical models. However, for large complex domains, such as robotics for Urban Search and Rescue, existing QSIM implementations are inefficient. ASPQSIM is a novel formulation of the QSIM algorithm in Answer Set Programming that takes advantage of the similarities between qualitative simulation and constraint satisfaction problems. ASPQSIM is compared against an existing QSIM implementation on a variety of domains that demonstrate ASPQSIM provides a significant improvement in efficiency especially on complex domains, and producing simulations in domains that are not solvable by the procedural implementation.

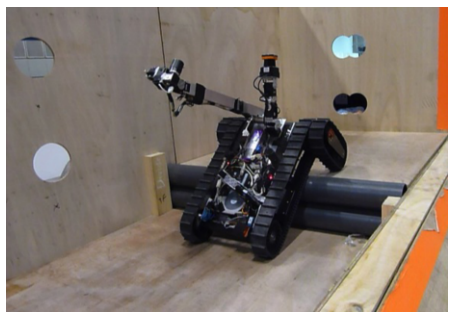

Figure 1. iRobot Negotiator platform for Urban Search and Rescue. The robot is shown climbing a step.

1 Introduction

Qualitative Reasoning is a field of research that models the behaviour of physical systems in continuous state spaces. Variables of a system and relationships between them are coarsely represented by qualitative descriptions, rather than by precise quantitative values or numerical models. For example, consider the iRobot Negotiator (Figure 1), a track-based robotic platform used for Urban Search and Rescue. In a qualitative model of the Negotiator, if the robot's velocity is positive, the x-coordinate of the robot's position increases. However the precise rate at which the x-coordinate increases is unknown. Further, in a qualitative solution to solving a given task, the robot might have to drive forward, then turn right, and finally continue driving forward. Again, the precise quantitative turning angle or length of time to perform each step is unknown. Qualitative reasoning algorithms only deduce qualitative relationships between the variables of a system, or predict the qualitative evolution in the state of a dynamic system [9, 12]. These techniques have been used for a broad range applications including Spatial Reasoning [8], developing controllers for mechanical machines such as a shipping-create crane [6], or live monitoring tools for controlling home appliances [10].

Qualitative Simulation (QSIM) [17] is a technique that applies qualitative reasoning to dynamic systems and predicts how the systems change over time, by simulating the sequences of states that the system will transition through as time progresses. During simulation, the system may be influenced by external forces which affect further changes in the state of the system. The domain of a system is defined by qualitative variables and the system's dynamics are defined by a qualitative model. In the Negotiator system, for example, variables may include the robot's velocity and x-coordinate, while the model specifies that the velocity is the derivative of the x-coordinate. QSIM shares many features of Constraint Satisfaction Problems. Potential states that the system may take are generated by transition rules. The qualitative model places constraints on the values of the system's variables to test if a generated state is valid. Therefore, a valid simulation of the system is a sequence of states that conform to the transition rules and constraints of the qualitative model.

Answer Set Programming (ASP) [13] is a logic reasoning tool suited to solving constraint satisfaction problems. Programs are specified in ASP using first-order logical formulations, from which an ASP solver generates potential solutions. The potential solutions are verified against integrity constraints that specify invalid logical facts that may not appear in a solution. Solutions are found by first grounding the formulation into a collection of facts where all variables from the original formulation are enumerated by atomic values. A solver then finds solutions satisfying the grounded problem. Designing an efficient ASP program requires balancing the workload of both the grounder and the solver.

This paper details *ASPQSIM*, a novel formulation of QSIM in ASP which has improved run-time performance over an existing QSIM implementation. The efficiency of ASPQSIM is compared on common domains in the qualitative reasoning literature, and on the Negotiator robotic domain.

1.1 Motivation

In previous work [25] we incorporated and extended Qualitative Simulation into a planner for a robot. The robotic system is described by a qualitative variables and a model, which is given to the planner that defines actions in relation to special qualitative *control variables*. Qualitative simulation is used to produce a sequence of states that lead from an initial state to a desired goal state, from which the planner calculates the actions necessary to solve the given task. However, each action in the plan is parameterised. The precise quantitative val-

[1] School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia email: {timothyw, claude}@cse.unsw.edu.au
[2] Faculty of Computer and Information Science, University of Ljubljana, Trzaska 25, 1000 Ljubljana, Slovenia email: bratko@fri.uni-lj.si

ues needed to physically execute the plan on the robotic system are discovered by a trial-and-error learner. This architecture (Figure 2) was applied to the iRobot Negotiator platform (Figure 1) used for Urban Search and Rescue. Specifically, the task of climbing onto a step (Figure 3) was chosen, as this is a common research problem in the field [16]. The Negotiator contains a set of main tracks to drive the robot and sub-tracks, or flippers, that can re-configure the geometry of the robot to climb over obstacles. The planner must choose the best sequence of actions to overcome terrain obstacles without becoming stuck. The step climbing task is solved using one of two approaches (Figure 3), either driving forward over the step, or if the step is too high, turning the robot around and driving backwards over the step. Qualitative planning provides a domain independent method to learning robotic behaviours that does not require extensive domain knowledge that is typically needed to build numerical computer simulations [24], or required for domain specific reasoning during planning [23]. However, for real-world problems, such as the Negotiator climbing a step, the planner's search space becomes very large.

Traditional implementations of QSIM cause the qualitative planner to hit server time and space complexity problems [25]. Improving the efficiency of QSIM by encoding the algorithm in ASP is the purpose of this work. As within the planner QSIM is only used to produce sequences of states, ASPQSIM is compared with existing QSIM implementations for this purpose. That is, experiments compare the efficiency of each algorithm to find a simulation from an initial state to a goal state.

1.2 Related Work

ASPQSIM uses the constraint satisfaction nature of QSIM for building the ASP formulation. ASP has also been previously applied to constraint satisfaction problems, typically using a *generate-and-test* methodology [2]. In this technique, ASP *cardinality rules* generate potential solutions to the constraint problem, and *integrity constraints* test (or validate) a potential solution by representing information about a valid solution that must not be true. The generate-and-test technique is used in ASPQSIM.

Applying tools designed for solving Constraint Satisfaction or Logic Programs (CSPs/CLPs) to QSIM has been previously investigated. The ECLiPSe CSP solver was combined with a parallel processing architecture [21] but the resulting system did not perform substantially better than existing procedural implementations. An implementation of QSIM that used the Prolog CLP(FD) library was found to be more efficient than standard procedural implementations [3]. However, this required customised numerical representations of the value of variables, and was only applied to a cascading water tank domain. Applying this numerical technique to other domains has not been experimentally investigated. Temporal logic was combined with the ECLiPSe CSP solver and applied to spatial reasoning domains [1], however no evaluation of the efficiency of the implementation was performed. In contrast to these system, ASPQSIM provides an efficient solver without having to convert the qualitative representation to a custom numerical encoding.

Planning is another frequently studied application of ASP. Numerous *Action Languages* within ASP have been proposed [15, 19]. These languages provide generic methods of representing planning problems. However, planning with qualitative simulation does not require explicit consideration of actions during simulation. Therefore, action languages are not required for ASPQSIM.

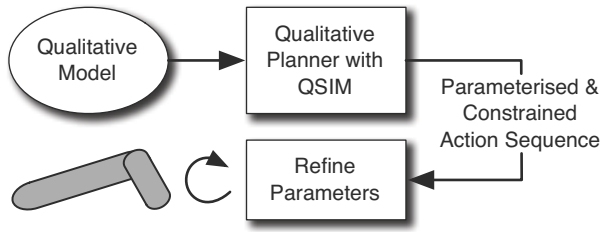

Figure 2. Three stage architecture for learning robotic behaviours using a qualitative planner based on the QSIM algorithm, and a quantitative trial-and-error learner.

2 The QSIM Algorithm

We first describe the QSIM algorithm and then explain its ASP formulation in Section 3. In qualitative simulation, each possible value of a variable, v, of the system being modelled, is described with respect to landmark values, L_i of v, within the domain of the variable:

$$v := [-\infty, L_0, L_1, L_2, \ldots, \infty]$$

Landmarks are distinguished symbolic values within the domain of v. However, their exact quantitative values are unknown, and the variable's domain may optionally contain landmarks for negative and positive infinity. The qualitative value of a variable is defined by a magnitude (which is either a landmark or the interval between two landmarks) and a direction of change (steady, increasing or decreasing) that indicates how the variable's magnitude changes over time. For example, v may hold the values:

$$v = L_0..L_1/dec \qquad v = L_1/std \qquad v = L_1..L_2/inc$$

A *qualitative state* of the system is the combination of a value for each variable in the system. Time is also explicitly represented in QSIM as the algorithm calculates the change in a system over time. Time is describe relative to discrete landmarks,

$$T := [t_0, t_1, t_2, \ldots, t_n]$$

but unlike qualitative variables, time always increases and is finitely bound by the maximal landmark t_n. Each qualitative state either occurs at a time point t_i or during a time interval $t_i..t_{i+1}$, and during simulation, time alternates between points and intervals. Time points and intervals are referred to as *time steps*. Similar to variables, the ordering of time landmarks is known but the quantitative value of each time landmark is unknown.

A *qualitative model* defines valid qualitative states of the system. A model is described using qualitative constraints in the form of *Qualitative Differential Equations (QDEs)* which place restrictions on the magnitude and the direction of change of variables. For example, the monotonicity constraint $\mathtt{M}^+(x, y)$ requires that the directions of change for x and y are always equal. If the value of x is increasing, y must also be increasing, and likewise for decreasing x and y. Table 1 lists common types of qualitative constraints. Potential qualitative states are validated against the model to determine whether the system may evolve into the given state.

We previously extended the definition of the qualitative model by introducing *qualitative rules* [25] of the form:

$$Name : \{Preconditions\} \rightarrow Constraint$$

A qualitative state is only validated against the constraint of a given rule if the preconditions for the rule are met. Qualitative rules

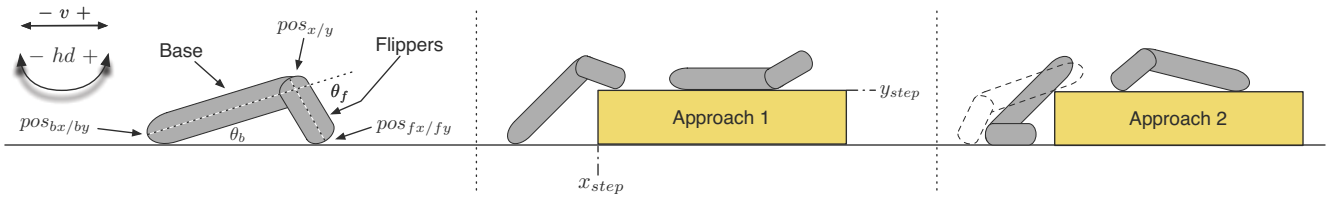

Figure 3. Representation of Negotiator and the step climbing task with the two broad approaches to climbing the step, driving forward (Approach 1) and reversing (Approach 2).

Table 1. Common types of qualitative constraints.

QDE	Description
$\texttt{M}^+(x, y)$	Monotonicity between x and y
$\texttt{M}^-(x, y)$	Inverse monotonicity between x and y
$\texttt{sum}(x, y, z)$	$z = x + y$
$\texttt{deriv}(x, y)$	y is the time derivative of x
$\texttt{const}(x, k)$	$x = k/std$

allow the model to change over time as the dynamic system changes, rather than each constraint applying globally in all states.

Algorithm 1 lists the main parts of the QSIM algorithm using Bratko's Prolog implementation [5], which has been extended to use qualitative rules. Given a state s_i occurring at time step T_i, the `state_transition` predicate defines the valid form of the successor state s_{i+1} which the system may evolve into at the next time step. There may be multiple potential successor states. To calculate the successor state, the next time step is determined by the `time_transition` predicate where $p(i)$ represents a time point and $i(i, i+1)$ an interval. The qualitative values of the variables $[v_1, \ldots, v_n]_i$ of state s_i are extracted, and a potential successor state is generated using the QSIM transition table [17] that defines all possible values for each variable in s_{i+1}. Finally, the potential state is validated against the model, by applying each rule of the model in sequence. The `test_precond` predicate asserts that the preconditions of the rule are met. However specific implementations of the predicate are not relevant and are not provided. The `validate_qde` predicate delegates the validation of each rule to the appropriate predicate for the QDE. For example, the `mplus` predicate implements the $\texttt{M}^+(x, y)$ QDE. Predicates for other QDEs are found in Bratko's original QSIM implementation [5].

A system may be simulated over multiple time steps through repeated applications of `state_transition`. This produces a directed graph of connected states that a system could evolve through. Therefore, it is possible to find a sequence of states between a specific initial state s_0 and specific goal state s_g, using a cost-based search.

2.1 Cost-based Search

Cost-based heuristic search algorithms typically require a combination of the cost for each state transition $s_i \rightarrow s_{i+1}$, and an estimate of the distance to the goal state. We have previously proposed definitions of these costs in [25]. For simplicity, the cost of each state transition is defined as a constant value of one. The heuristics for estimating the distance to the goal state use the Qualitative Magnitude Distance (QMD). The QMD is the length of the shortest state sequence that is required for a single variable to transition between two of its values (the value of the variable in a given state, and its value in the goal state). The MaxQMD heuristic takes the maximum QMD over all variables of the system while the TotalQMD heuristic sums the QMDs. As shown in our experiments with ASPQSIM (Section 4) and our previous work [25, 26], the choice of heuristic greatly impacts the performance of QSIM.

Algorithm 1: State transition rules and the validation of the $\texttt{M}^+(x, y)$ QDE in "pseudo Prolog", reflecting the Bratko Prolog QSIM implementation.

```
% Definition of a single valid transition s_i → s_{i+1}
% M is the qualitative model
% The state s_i is a list of the value of the variables [v_1, ..., v_n]_i
state_transition(s_i, T_i, M, s_{i+1}, T_{i+1}) :-
    time_transition(T_i, T_{i+1}),
    maplist(var_transition(T_i), s_i, s_{i+1}),
    validate_model(M, s_{i+1}).

% Time transition between points and intervals
time_transition(p(i), i(i, i+1)).
time_transition(i(i, i+1), p(i+1)).

% State transition rules (examples)
var_transition(p(i), L_1/std, L_1/std).
var_transition(p(i), L_1/std, L_1..L_2/inc).
var_transition(p(i), L_1/std, L_0..L_1/dec).
var_transition(i(i, i+1), L_1/std, L_1/std).

% Validate a state against all rules in the model
% Model M is a list of rules rule(P, Q)
validate_model(M, s_i) :- maplist(validate_rule(s_i), M).

% Validate the state against a single rule
% If preconditions P are true, then constraint Q must hold
validate_rule(s_i, rule(P, Q)) :- ¬test_precond(P, s_i).
validate_rule(s_i, rule(P, Q)) :-
    test_precond(P, s_i),
    validate_qde(Q, s_i).

% Validation for M+(x, y) constraint
% The list [corr] contains corresponding values.
% relative_qmag gives the sign <, > or ==
% that relates mag_x with x or mag_y with y
mplus(x : mag_x/dir_x, y : mag_y/dir_y, [corr]) :-
    dir_x == dir_y,
    maplist(mplus_correspond(mag_x, mag_y), [corr])

mplus_correspond(mag_x, mag_y, (c_x, c_y)) :-
    relative_qmag(mag_x, c_x, sign_x)
    relative_qmag(mag_y, c_y, sign_y)
    sign_x == sign_y
```

3 ASPQSIM

ASPQSIM, our formulation of Qualitative Simulation in ASP, is described using the ASP-Core-2 syntax.

Each qualitative variable of the system is specified by facts of the form:

$$\text{qvar}(x).$$

The legal values of variable are defined by facts of the form:

$$\text{qmag}\,(x, \text{land}\,(L_0))\,. \quad \text{qmag}\,(x, \text{interval}\,(L_0, L_1))\,.$$

which state that x is either at the landmark L_0 or is in the interval $L_0..L_1$. Qualitative directions of change are listed in the facts:

$$\text{qdir(std)}. \quad \text{qdir(inc)}. \quad \text{qdir(dec)}.$$

Time landmarks are explicitly represented as:

$$\text{timevalue}\,(0..t_n)\,.$$

Each time step (for point $p(..)$ and interval $i(..)$) is represented as:

$$\text{time}\,(\text{p}\,(T)) \text{ :- timevalue}\,(T)\,.$$
$$\text{time}\,(\text{i}\,(T, T+1)) \text{ :- timevalue}\,(T)\,, \text{timevalue}\,(T+1)\,.$$

A qualitative state is a collection of facts of the form:

$$\text{holds}(Time, Var, Mag, Dir).$$

which represent that in time step $Time$, variable Var has the value $Var = Mag/Dir$. The complete state at a given time step requires one and only one `holds` fact for each variable.

Using the above facts the main elements of ASPQSIM are detailed in Algorithm 2. The state transition rules are given as cardinality rules. These generate potential `holds` facts and use the cardinality to enforce that one and only one fact for each variable at each time step is generated. The predicates `time`, `qmag` and `qdir` lookup facts and ensure correct instantiations of the arguments of `holds`. The transition rules not listed can be represented in a similar manner.

The qualitative model is represented using integrity constraints to test whether a generated set of `holds` facts are valid. Each qualitative rule in the model is defined by a fact

$$\text{rule}(Id, PreConds, Constraint).$$

containing an unique identifier, the number of preconditions and qualitative constraint for the rule. Each precondition for a rule is specified by the fact

$$\text{precondElem}(Time, Id, i).$$

which denotes that at time step $Time$, the i'th precondition for the rule corresponding to the identifier Id is met. If all of the preconditions for a rule are met at a given time step, the qualitative constraint for the rule is activated for that time step by the fact

$$\text{qde}(Time, Id, Qde).$$

where Qde is the qualitative constraint for the rule.

As an example, the $\texttt{M}^{+}(x, y)$ constraint is enforced by two integrity constraints. The first integrity constraint ensures the directions of change for the two variables are equal. The second constraint ensures the values of variables are correct relative to known corresponding values. Similar to Algorithm 1, `correspond` lists the corresponding values for the $\texttt{M}^{+}$ constraint, and `relative_qmag` is the relation $<$, $>$ or $==$ between two qualitative values. Additionally, the unique identifier for the rule ensures the correct corresponding values are used.

Algorithm 2: State transition rules, integrity constraints for the $\texttt{M}^{+}(x, y)$ QDE, and specification of the initial and goal states in ASPQSIM.

```
% State Transition Rules (example)
1 { holds(i(T, T1), V, land(L1), std);
    holds(i(T, T1), V, interval(L1, L2), inc) :
        qmag(V, interval(L1, L2));
    holds(i(T, T1), V, interval(L0, L1), dec) :
        qmag(V, interval(L0, L1)) } 1
:- time(i(T,T1)), holds(p(T), V, land(L1), std).

% A QDE holds at time T if all preconditions hold at T
qde(T, Name, QDE) :- preconds(T, Name), rule(Name, _, QDE).

% preconds holds at T if every element holds at T
preconds(T, Name) :- rule(Name, Count, _), time(T),
    Count { precondElem(T, Name, N) : N = 1..Count } Count.

% Example precondElem
precondElem(T, example, 1) :- time(T), holds(T, x, L0, std)

% M+(x,y) integrity constraints
:- qde(T, _, mplus(V1, V2)),
    holds(T, V1, _, Dir1), holds(T, V2, _, Dir2),
    Dir1 != Dir2.
:- qde(T, Name, mplus(V1, V2)),
    holds(T, V1, Mag1, Dir1), holds(T, V2, Mag2, Dir2),
    Dir1 == Dir2,
    correspond(Name, Corr1, Corr2),
    relative_qmag(Mag1, Corr1, Sign1),
    relative_qmag(Mag2, Corr2, Sign2),
    Sign1 != Sign2.

% Constrain initial state
holds(p(0), V, Mag, Dir) :- initial(V, Mag, Dir).
1 { holds(p(0), V, Mag, Dir) :
    qmag(V, Mag), qdir(Dir) } 1 :- qvar(V).

% Constrain goal state
:- goal(V, Mag, Dir), time(p(t_n)), not holds(p(t_n), V, Mag, Dir).
```

To find a sequence of states between an initial state s_i and goal state s_g, two sets of facts define s_i and s_g. The initial state and goal state are represented by a set of the two kinds of facts:

$$\text{initial}(Var, Mag, Dir). \qquad \text{goal}(Var, Mag, Dir).$$

The `initial` facts constrain the value of the `holds` facts for each variable at the first time step $p(0)$. However, not every variable of the system may be specified in the initial state. Therefore, a cardinality rule ensures that there exists one `holds` fact for each variable at $p(0)$. The goal state is enforced by an integrity constraint for each `goal` fact, such that each `holds` must conform to the goal at the terminal time step $p(t_n)$.

3.1 Incremental Solving by Iterating over Time

ASP programs cannot have unbounded values. Therefore in ASPQSIM, the number of time steps for a simulation, that is the precise value of the maximal time landmark t_n, must be manually specified before solving begins. Furthermore, ASPQSIM will always find a state sequence that takes all available time steps. However, it

Table 2. Complexity of the domain used in comparing performance.

Domain	Variables	Rules	Potential States
Bouncing Ball	3	3	180
Bathtub	6	5	1728
5-Tanks	16	25	5.2×10^{12}
10-Tanks	31	50	3.0×10^{24}
Negotiator	18	105	7.5×10^{14}

is desirable to find the shortest sequence of states, and in practice the length of the shortest sequence is unknown. This problem is resolved by using incremental ASP solving as implemented *iClingo4* [14]. During incremental solving, the representation starts with only one time landmark, and the number of landmarks is increased iteratively until a state sequence is found. This also gives the shortest sequence of states that is required to reach the goal.

The incremental version of ASPQSIM, called Inc-ASPQSIM, modifies Algorithm 2 to instruct the ASP Solver how to update the grounded facts for each iteration. The value of the maximal time landmark t_n is incremented by one at the start of each iteration. All facts that do not contain an argument for time, and hence do not change, are grounded once before solving begins. For the remaining statements, additional grounded facts are added on each iteration. Grounded facts are not reprocessed and only new facts are grounded that correspond to the time step for the current iteration. Finally, grounded facts for the goal integrity constraint are removed from the solver's database and reasserted on each iteration. Old goal integrity constraints must be removed, otherwise the constraints would require that the goal is reached at every time step.

4 Performance

Experiments were conducted to compare the efficiency of ASPQSIM to Bratko's Prolog QSIM implementation that has been extended to use qualitative rules. The experiments also compared the efficiency of ASPQSIM to the incremental version Inc-ASPQSIM. The experiments were conducted on Negotiator step climbing task, and commonly studied domains of varying complexities within the qualitative reasoning literature [18]. The Bouncing Ball, Bath and N-Tanks (N cascading water tanks which sequentially fill each other) domains were chosen. The common domains were used in the experiments to ensure that ASPQSIM runs just as efficiently within these domains. The experiments were conducted on a 64-bit MacBook Pro 8,1 (2GHz Intel Core i7), that used SWI-Prolog (v. 6.6.1) for the Prolog QSIM, and Clingo (v. 4.2.2) for the ASP Solver.

Table 2 lists the complexities of each domain in terms of the number of variables and rules in the domain, and the upper bound on the number of potential states in the search space. The Bouncing Ball, Bathtub and 5-Tanks are simple domains with few variable and rules. The 10-Tanks and Negotiator domains are significantly more complex, where the 10-Tanks domain has a large number of variables, and the Negotiator domain has a large number of rules with complex preconditions. For each domain a set of experiments is conducted where each QSIM implementation must find a sequence of states that solves a given task in that domain. For the bouncing ball, the task was to simulate the trajectory of the ball over one bounce, for the bathtub the task is to fill the bath, and for the N-Tanks the task is fill all N tanks. In the Negotiator domain multiple different tasks were used. The step climbing task may be accomplished using two approaches (Figure 3). However, as noted in Future Work (Section 5), with only qualitative information the planner cannot deduce which approach is appropriate as this depends on the quantitative height of the step. Thus, the qualitative model was modified such that only one approach could be discovered for the relevant experiments.

Table 3. Profile of the time spent in each phase of ASP for ASPQSIM. The percentage of the total time required for grounding is calculated.

Negotiator Domain	Grounder (sec)	Solver (sec)	Time Grounding (percentage)
Approach 1 (2 vars)	4.92	0.06	98.8%
Approach 1 (5 vars)	4.94	0.07	98.6%
Approach 2 (2 vars)	15.84	3.65	81.3%
Approach 2 (5 vars)	15.03	3.17	82.6%

Previously in [25] we found that the efficiency of the Prolog QSIM greatly depended on the choice of heuristic for the cost-based search. Therefore, the experiments compared ASPQSIM to the performance of Prolog QSIM with both the MaxQMD and TotalQMD heuristic. We also noted in [25] that the goal for a task may not include all variables of the system, as depending on the domain, it may not be possible to determine in advance an appropriate value in the goal state for every variable. For the Negotiator step climbing task, typically only the final velocity and x-coordinate are known in the goal state. The values of up to 3 other variables may additionally be known, which describe the bounding-box of the robot in the goal state. Furthermore, we noted in [25] that the number of variables specified in the goal greatly impacted the efficiency of the Prolog QSIM's. Thus, the experiments also analysed the impact of the number of variables in the goal state for the Negotiator domain.

Table 4 summarises the results of the experiments. The simple domains are efficiently solved by all QSIM implementations. This demonstrates that ASPQSIM is viable for simple domains. However, on the significantly more complex 10-Tanks and Negotiator domains, ASPQSIM significantly out-performs the Prolog QSIM. In some cases, ASPQSIM is able to find a solution where the Prolog QSIM failed to find a solution in a reasonable period of time (greater than 3 hours) or Prolog ran out of memory.

ASPQSIM significantly out-performs the Prolog QSIM because of the constraint satisfaction nature of QSIM. Table 3 shows the breakdown of the time spent in each phase of the ASP solver. The transition rules and qualitative model highly constrain possible solutions, which the ASP solver takes advantage of whereas, the Prolog QSIM cannot. Hence ASPQSIM finds solutions faster. The majority of the ASP execution time is due to the grounding, but the grounder is still able to execute quickly despite having to generate integrity constraints for the qualitative model at each time step.

The experiments also show that Inc-ASPQSIM is, at worse, only marginally slower than ASPQSIM on non-trivial domains. This is largely because over 80% of the work of the ASP solver is in the grounder (Table 3). Both ASP versions require the same amount of work for grounding. The results show that there is little overhead from the solver failing to find solutions while incrementing t_n.

5 Future Work - Quantitative Constraints

Qualitative Simulation has a number of known deficiencies that stem from both the non-determinism in the state transition rules and the use of purely qualitative landmarks [22]. On the Negotiator we have found that using QSIM with only qualitative landmarks may produce a sequence of states that cannot be physically executed on the robot [25]. For example, the planner will think it possible to climb a step that is one kilometre high! This problem was resolved in [26] by introducing quantitative values for some landmarks and propagating quantitative constraints in the manner of [4] during simulation

Table 4. Comparison of the execution time (in seconds) of the modified Bratko Prolog QSIM (using both heuristics) and ASPQSIM. The percentage speed increase for the ASPQSIM compared to the Prolog QSIM, and the percentage difference in speed between ASPQSIM and Inc-ASPQSIM is given. For some domains, Prolog QSIM does not find a solution as it took too long (∗) or ran out of memory (†).

Domain	Prolog QSIM		ASPQSIM	Inc-ASPQSIM	ASPQSIM increase		Slowdown of
	MaxQMD	TotalQMD			MaxQMD	TotalQMD	Inc-ASPQSIM
Bouncing Ball	0.28	0.28	0.07	0.07	431%	404%	6.2%
Bathtub	0.29	0.28	0.04	0.07	725%	400%	75%
5-Tanks	3.25	3.44	0.18	0.26	1,858%	1,317%	49.1%
10-Tanks	*	*	0.78	0.82	-	-	5.8%
Negotiator Approach 1 (2 vars)	42.54	188.63	4.98	4.42	853%	4,269%	-11.3%
Negotiator Approach 1 (5 vars)	634.51	103.41	5.01	5.27	12,672%	1,962%	5.2%
Negotiator Approach 2 (2 vars)	†	†	19.49	20.39	-	-	4.7%
Negotiator Approach 2 (5 vars)	†	†	18.20	18.82	-	-	3.4%

in order to rule out physically invalid states. The Prolog QSIM implementation used the CLP(FD) library [7] to easily implement the constraints. However, the use of quantitative constraints had significant impacts on the performance of QSIM.

To ensure the correctness of simulations, quantitative constraints should be added to ASPQSIM. However, it has been shown that for numerical reasoning problems ASP performs significantly worse compared to CLP(FD) due to an explosion in the work of the grounder [11]. Efficiently implementing quantitative constraints in ASPQSIM using *hybrid-reasoning* [20] is currently being investigated, but preliminary results show poor performance. Hybrid-reasoning in ASP shifts the quantitative constraints out of the grounder and into the solver.

6 Conclusion

ASPQSIM relies on the solving power of Answer Set Programming, to simulate the evolution in the state of a dynamic system over time. ASPQSIM was experimentally compared with a Prolog implementation of QSIM. It should be noted that Prolog QSIM is an extension of a Prolog program in [5] which was aimed at clarity and conciseness, with limited considerations of efficiency. Nevertheless, the experimental results indicate that ASP is probably a better framework for implementing QSIM, in which efficiency is easier to achieve. Additionally, using incremental ASP solving (Inc-ASPQSIM) avoids manually specifying the length of the simulation and does not introduce large overheads to performance.

Acknowledgements

We thank Dr. Torsten Schaub (University of Potsdam, Germany) and Dr. Michael Thielscher (University of New South Wales, Australia) for their assistance in developing an efficient encoding of QSIM in the ASP language and working with the Clingo solver.

REFERENCES

[1] K. R. Apt and S. Brand, 'Infinite Qualitative Simulations by Means of Constraint Programming', volume 4204 of *Lecture Notes in Computer Science*, 29–43, Springer Berlin Heidelberg, (2006).
[2] M. Balduccini, 'Representing constraint satisfaction problems in answer set programming', in *Workshop on Answer Set Programming, in Logic Programming, 25th International Conference on*, (2009).
[3] A. Bandelj, I. Bratko, and D. Šuc, 'Qualitative Simulation with CLP', in *Qualitative Reasoning (QR), 16th International Workshop on*, (2002).
[4] D. Berleant and B. J. Kuipers, 'Qualitative and Quantitative Simulation: Bridging the Gap', *Artificial Intelligence*, **95**(2), 215–255, (1997).
[5] I. Bratko, *Prolog Programming for Artificial Intelligence*, Addison-Wesley, 2011.
[6] I. Bratko and D. Šuc, 'Learning Qualitative Models', *AI Magazine*, **24**(4), 107–119, (2003).
[7] P. Codognet and D. Diaz, 'Compiling constraints in clp(FD)', *The Journal of Logic Programming*, **27**(3), 185–226, (1996).
[8] A. G. Cohn and S. M. Hazarika, 'Qualitative spatial representation and reasoning: An overview', *Fundamenta Informaticae*, **46**(1-2), 1–29, (2001).
[9] J. De Kleer and J. S. Brown, 'A qualitative physics based on confluences', *Artificial Intelligence*, **24**(1-3), 7–83, (1984).
[10] G. F. DeJong, 'Learning to Plan in Continuous Domains', *Artificial Intelligence*, **65**(1), 71–141, (1994).
[11] A. Dovier, A. Formisano, and E. Pontelli, 'A Comparison of CLP(FD) and ASP Solutions to NP-Complete Problems', volume 3668 of *Lecture Notes in Compter Science*, 67–82, Springer Berlin Heidelberg, (2005).
[12] K. D. Forbus, 'Qualitative Process Theory', *Artificial Intelligence*, **24**(1-3), 85–168, (1984).
[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, *Answer Set Solving in Practice*, Morgan & Claypool Publishers, 2013.
[14] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, 'Potassco: The Potsdam Answer Set Solving Collection', *AI Communications*, **24**(2), 107–124, (2011).
[15] M. Gelfond and V. Lifschitz, 'Action Languages', *Electronic Transactions on AI*, **3**(6), 193–210, (1998).
[16] A. Jacoff, E. Messina, B. A. Weiss, S. Tadokoro, and Y. Nakagawa, 'Test arenas and performance metrics for urban search and rescue robots', in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pp. 3396–3403, (2003).
[17] B. J. Kuipers, 'Qualitative Simulation', *Artificial Intelligence*, **29**(3), 289–338, (1986).
[18] B. J. Kuipers, *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press, 1994.
[19] J. Lee, V. Lifschitz, and F. Yang, 'Action language BC: preliminary report', in *Artificial Intelligence (IJCAI), 23rd International Joint Conference on*, pp. 983–989, (2013).
[20] M. Ostrowski and T. Schaub, 'ASP modulo CSP: The clingcon system', *Theory and Practice of Logic Programming*, **12**(4-5), 485–503, (2012).
[21] M. Platzner, B. Rinner, and R. Weiss, 'Parallel qualitative simulation', *Simulation Practice and Theory*, **5**(7-8), 623–638, (1997).
[22] C. J. Price, L. Trave-Massuyes, R. Milne, L. Ironi, et al., 'Qualitative futures', *Knowledge Engineering Review*, **21**(4), 317–334, (2006).
[23] C. K. Tseng, I. H. Li, Y. H. Chien, M. C. Chen, and W. Y. Wang, 'Autonomous Stair Detection and Climbing Systems for a Tracked Robot', in *System Science and Engineering (ICSSE), IEEE International Conference on*, pp. 201–204, (2013).
[24] I. Vincent and Q. Sun, 'A combined reactive and reinforcement learning controller for an autonomous tracked vehicle', *Robotics and Autonomous Systems*, **60**(4), 599–608, (2012).
[25] T. Wiley, C. Sammut, and I. Bratko, 'Planning with Qualitative Models for Robotic Domains', in *Advances in Cognitive Systems (Poster Collection), Second Annual Conference on*, pp. 251–266, (2013).
[26] T. Wiley, C. Sammut, and I. Bratko, 'Qualitative Planning with Quantitative Constraints for Online Learning of Robotic Behaviours', in *Artificial Intelligence (AAAI), 28th AAAI Conference on (to appear)*, (2014).