Bound Founded Answer Set Programming

Rehan Abdul Aziz

Supervisors: Peter Stuckey and Geoffrey Chu
National ICT Australia, Victoria Laboratory,∗
Department of Computing and Information Systems
Room 6.22, Doug McDonell Building (Building 168),
The University of Melbourne, Australia
Email: `raziz@student.unimelb.edu.au`

Abstract

Answer Set Programming (ASP) is a powerful modelling formalism that is very efficient in solving combinatorial problems. ASP solvers implement the stable model semantics that eliminates circular derivations between Boolean variables from the solutions of a logic program. Due to this, ASP solvers are better suited than propositional satisfiability (SAT) and Constraint Programming (CP) solvers to solve a certain class of problems whose specification includes inductive definitions such as reachability in a graph. On the other hand, ASP solvers suffer from the *grounding bottleneck* that occurs due to their inability to model finite domain variables. Furthermore, the existing stable model semantics are not sufficient to disallow circular reasoning on the bounds of numeric variables. An example where this is required is in modelling shortest paths between nodes in a graph. Just as reachability can be encoded as an inductive definition with one or more base cases and recursive rules, shortest paths between nodes can also be modelled with similar base cases and recursive rules for their *upper bounds*. This deficiency of stable model semantics introduces another type of grounding bottleneck in ASP systems that cannot be removed by naively merging ASP with CP solvers, but requires a theoretical extension of the semantics from Booleans and normal rules to bounds over numeric variables and more general rules. In this work, we propose Bound Founded Answer Set Programming (BFASP) that resolves this issue and consequently, removes all types of grounding bottleneck inherent in ASP systems.

1 Motivation

Answer Set Programming (Baral 2003) is a useful modelling paradigm to solve search and planning problems. Modern ASP solving (Gebser et al. 2007; Gebser et al. 2012) builds on propositional satisfiability (SAT) solving (Mitchell 2005). However, ASP solvers have a competitive edge over SAT solvers in problems whose model involves some notion of transitive closure, e.g., reachability or connectivity in a graph. This is due to the difference in semantics of both systems; ASP solvers implement stable model semantics (Gelfond and Lifschitz 1988) which minimizes the number of variables that are true in a given logic program while a SAT solver only looks for an assignment that satisfies all the given clauses. In ASP, in order for a variable to be true, it must have some rule as a *support* that *justifies* it being true. Furthermore, no set of variables can circularly support one another. E.g. given two rules $a \leftarrow b$ and $b \leftarrow a$, the only valid solution in

stable model semantics is where a and b are both false, whereas in propositional semantics, both the variables being true is also a valid solution.

As ASP systems such as SMODELS and CLASP only deal with Boolean variables, they are inefficient for solving problems that are naturally modelled with integers, especially if they have large domains. These combinatorial problems (e.g. scheduling) are ubiquitous in Computer Science, which makes ASP a poor choice to model and solve them. The most obvious way to model these in ASP is to represent each integer's domain as a set of Boolean variables and impose constraints on these variables to ensure consistency. This incurs what is commonly known as the *grounding bottleneck* problem. Naturally, decomposing a large finite domain to Boolean variables blows up the problem size in the size of domains of integer variables. Constraint Programming solvers (Rossi et al. 2006) and Mixed Integer Programming solvers, on the other hand, are excellent candidates for these problems as they support numeric variables natively. Unfortunately, constraint solvers suffer from the same inefficiency as SAT solvers regarding problems like reachability that require inductive definitions. A hybrid system that has the best of both worlds, i.e., inductive rules for Boolean variables from ASP, and native support for integers and constraints over them from CP, addresses both the concerns. However, in this work, we propose that even this hybrid approach is not sufficient, and there exists a type of grounding bottleneck that is still not removed by combining the strengths of CP and ASP solvers. Let us illustrate this by first looking at a benchmark from ASP competitions, and then modifying its problem description.

Consider the Minimum Connected Dominating Set (MCDS) problem. A dominating set is a set of nodes such that every node in the graph is either in the set or has at least one neighbour in the set. The objective is to find a dominating set of minimum cardinality such that the subgraph induced by dominating nodes is connected. Let us look at the ASP encoding of the problem [1]. A vertex X is given in the input as $vtx(X)$, an edge from X to Y as $edge(X, Y)$ and the edge relation is symmetric.

$$
\begin{aligned}
R_1 \quad & \{dom(U) : vtx(U)\}. \\
R_2 \quad & in(V) \leftarrow dom(V). \\
R_3 \quad & in(V) \leftarrow edge(U, V) \land dom(U). \\
C_1 \quad & \leftarrow vtx(U) \land \neg in(U). \\
R_4 \quad & reach(U) \leftarrow dom(U) \land \not\exists_{V<U} : dom(V). \\
R_5 \quad & reach(V) \leftarrow reach(U) \land dom(V) \land edge(U, V). \\
C_2 \quad & \leftarrow dom(U) \land \neg reach(U). \\
O \quad & minimize\{dom(U) : vtx(U)\}.
\end{aligned}
$$

R_1 introduces a decision variable dom for each vertex specifying whether it is a dominating vertex or not. R_1 is a *choice rule* which means that for a given node U, this rule can be a justification for $dom(U)$ if it is true. R_2, R_3 and C_1 model that every node must either be a dominating node or have a neighbour that is dominating. This is done with the help of an auxiliary predicate in that becomes true when at least one of the conditions is met. C_1 says that there can be no node for which in is false. R_4 and R_5 define the predicate $reach$ that is used to model the connectivity constraint of the induced subgraph by the dominating nodes. R_4 encodes the base case for reachability, specifying that the node with the lowest index is reachable by definition.

---

[1] Based on the model from Potassco group in the second ASP competition: `http://dtai.cs.kuleuven.be/events/ASP-competition/encodings.shtml`

This choice is arbitrary and its purpose can be satisfied by any criterion to select a dominating node. R_5 is a recursive case for reachability and it says that a dominating neighbour of a reachable node is also reachable. The constraint C_2 says that all dominating nodes must be reachable. The objective, given by O, is to minimize the cardinality of the dominating set.

Let us modify MCDS such that the edges in the graph also have weights ($edge(U, V, W)$ means that the edge from U to V has weight W) and there is an additional constraint that the diameter (maximum distance between any two nodes) of the dominating subgraph is less than a certain given value K. This problem has applications in computer networks (Kim et al. 2009; Buchanan et al. 2013). Let $d(X, Y)$ represent the distance (shortest path) between two dominating nodes X and Y. In MCDS, it is sufficient to check for reachability of every dominating node from an arbitrary node to ensure connectedness. However, to enforce the new constraint, we need a distance variable for each pair of nodes in the dominating set. We can replace R_4, R_5 and C_2 in the above encoding of MCDS with the following:

$$R_4 \quad d(U, U) \leq 0 \leftarrow dom(U).$$
$$R_5 \quad d(U, T) \leq d(V, T) + W \leftarrow dom(T) \wedge dom(U) \wedge dom(V) \wedge edge(U, V, W).$$
$$C_2 \quad \leftarrow dom(U) \wedge dom(V) \wedge d(U, V) > K.$$

R_4 is the base case for d and it says that the distance from a dominating node to itself is at most 0. R_5 is a recursive rule that specifies that for two dominating neighbours and a dominating node T, the distance between one end of the node to T is at most the distance between the other end and T, plus the weight of the edge. Finally, the constraint C_2 establishes that the distance between any two dominating nodes must be at most K. It is unnecessary to include the previous reachability rules since finite distances between all pairs of dominating nodes implies that the dominating set is connected.

Rules like R_4 and R_5 on integer variables are clearly not supported by current ASP systems. The semantics that we wish to associate with the distance variables is that firstly, if there are no rules supporting them, then they are equal to ∞. Secondly, any rule for a distance variable justifies a value lower than ∞ and thirdly, the upper bounds of these variables cannot form a circular justification. E.g. if there are two rules: $a \leq b$ and $b \leq a$, then any solution where a and b are equal to a finite value should be rejected, and the only *stable* solution should be one where both are equal to ∞. The distance variable is essentially an *upper-bound founded* (ub-founded) variable, for which the upper bound needs to be justified by some rule. We can encode these upper-bound founded distance variables in ASP along with our desired semantics by replacing R_4, R_5, and C_2 as follows:

$$d_{ub}(U, V, N) \leftarrow dom(U) \wedge dom(V) \wedge d_{ub}(U, V, N - 1), N < M.$$
$$d_{ub}(U, U, 0) \leftarrow dom(U).$$
$$d_{ub}(U, T, D + W) \leftarrow dom(T) \wedge dom(U) \wedge dom(V) \wedge edge(U, V, W) \wedge d(V, T, D).$$
$$d(U, V, D) \leftarrow dom(U) \wedge dom(V) \wedge d_{ub}(U, V, D) \wedge \neg d_{ub}(U, V, D - 1).$$
$$\leftarrow dom(U) \wedge dom(V) \wedge d(U, V, D) \wedge D > K.$$

In the above encoding, M is a sufficiently large integer and $d_{ub}(U, V, N)$ specifies that the distance between the dominating nodes U and V is *at most* N (the subscript ub stands for upperbound). $d(U, V, D)$ is defined as the minimum value for which $d_{ub}(U, V, D)$ is true. Unfortunately, an ASP solver on this encoding quickly runs into the grounding bottleneck problem as

we increase edge weights and the bound on diameter. This is the motivation of this work, i.e., to support founded numeric variables and rules like R_4 and R_5 without grounding them.

The symmetric analog for a ub-founded variable is a *lower-bound founded* (lb-founded) variable, which is by default equal to $-\infty$ (*false* for Boolean) and further rules for it can justify a greater value on its lower bound. In this generalization, ASP variables are simply lb-founded Boolean variables. For simplicity, we only consider lb-founded variables, and refer to them as founded variables. This simplification is possible because we can replace all ub-founded variables, their rules, and their constraints by corresponding lb-founded variables with similar rules and constraints. E.g., for the above problem, let $d(U, V)$ represent the negative of the distance between U and V, then we can perform this transformation as follows:

R_4 $d(U,U) \geq 0 \leftarrow dom(U).$
R_5 $d(U,T) \geq d(V,T) - W \leftarrow dom(T) \wedge dom(U) \wedge dom(V) \wedge edge(U,V,W).$
C_2 $\leftarrow dom(U) \wedge dom(V) \wedge d(U,V) < -K.$

An important point in the encoding of MCDS is that if we remove the reachability condition from the problem specification, and R_4, R_5, and C_2 from the encoding, then the problem can be solved by a SAT solver just as efficiently as an ASP solver. Recall that the only shortcoming of SAT solvers is related to modelling properties like reachability since the propositional semantics that they are based on does not naturally model recursive definitions. This leads us to the important observation that besides founded variables like *reach* and d, there can be variables in a problem like the *dom* variables that are not founded. Let us call them *standard* variables, owing to the fact that these are the usual variables in CP solvers. Standard variables can be assigned any value as long as all the constraints associated with them are satisfied. Founded variables, on the other hand, need rules to define their values and without them, they are equal to some default value.

Since there are no rules and founded variables in CP and MIP solvers, MCDS with bounded diameter as defined above cannot be efficiently solved by them. The MIP formulation as given in (Buchanan et al. 2013) encodes each distance variable with K propositional variables, meaning that the problem size increases with K. Our encoding above that uses founded variables does not suffer from this problem. This leads us to distinguish between the two types of grounding bottlenecks. One is caused in a system by the absence of its support for standard integer variables. Let us call this type the *standard grounding bottleneck*. The other type of grounding bottleneck is caused by the lack of a system's capabilities to handle founded numeric variables, therefore, let us call it *founded grounding bottleneck*.

In the next section, we formally define the semantics of *Bound Founded Answer Set Programming* (BFASP), a formalism that generalizes the stable model semantics to bounds over numeric variables, and allows for a richer set of rules for founded variables. [2]

2 Bound Founded Answer Set Programming

Let $\mathcal{V}$ be the set of variables. We consider three types of variables: integer, real, and Boolean. Furthermore, we divide the set of variables in two disjoint sets: standard $\mathcal{S}$ and *founded* variables $\mathcal{F}$. A *domain* D maps each variable $x \in \mathcal{V}$ to a set of constant values $D(x)$. A *valuation* (or

---

[2] The MINIZINC encoding of MCDS with bounded diameter in BFASP is given Appendix A.

assignment) θ over variables $vars(\theta) \subseteq \mathcal{V}$ maps each variable $x \in vars(\theta)$ to a value $\theta(x)$. A *constraint* c is a set of assignments over the variables $vars(c)$, representing the solutions of the constraint. Given a constraint c, a variable $y \in vars(c)$ is *monotonically increasing (decreasing)* in c if for all solutions θ that satisfy c, increasing (decreasing) the value of y also creates a solution, that is θ' where $\theta'(y) > \theta(y)$, and $\theta'(x) = \theta(x), x \in vars(c) - \{y\}$, is also a solution of c.

A *positive-CP* P is a collection of constraints where each constraint is increasing in exactly one variable and decreasing in the rest. The *minimal* solution of a positive-CP is an assignment θ that satisfies P s.t. there is no other assignment θ' that also satisfies P and there exists a variable v for which $\theta'(v) < \theta(v)$. Note that for Booleans, $true > false$. A satisfiable positive-CP P always has a unique minimal solution. If we have bounds consistent propagators for all the constraints in the program, then the unique minimal solution can be found simply by performing bounds propagation on all constraints until a fixed point is reached, and then setting all variables to their lowest values.

A *rule* r is a pair (c, y) where c is a constraint, $y \in \mathcal{F}$ is the head of the rule and it is increasing in c. A bound founded answer set program (BFASP) P is a tuple $(\mathcal{S}, \mathcal{F}, C, R)$ where C and R are sets of constraints and rules respectively. Given a variable $y \in \mathcal{F}$, $rules(y)$ is the set of rules with y as their heads.

The reduct of a BFASP P w.r.t. an assignment θ, written P^θ, is a positive-CP made from each rule $r = (c, y)$ by replacing in c each variable $x \in vars(c) - \{y\}$, if it is a standard variable or if c is not decreasing in it, by its value $\theta(x)$ to create a positive-CP constraint c'. Let r^θ denote this constraint. If r^θ is not a tautology, it is included in the reduct. An assignment θ is a stable model of P iff i) it satisfies all the constraints in P and ii) it is the minimal model that satisfies P^θ.

Example 1

Consider a BFASP with standard variable s, integer founded variables a, b, Boolean founded variables x and y, and the rules: $(a \geq 0, a)$, $(b \geq 0, b)$, $(a \geq b + s, a)$, $(b \geq 8 \leftarrow x, b)$, $(x \leftarrow \neg y \land (a \geq 5), x)$. Consider an assignment θ s.t. $\theta(x) = true$, $\theta(y) = false$, $\theta(b) = 8$, $\theta(s) = 9$ and $\theta(a) = 17$. The reduct of θ is the positive-CP: $a \geq b + 9$, $b \geq 8 \leftarrow x$, $x \leftarrow a \geq 5$. The minimal model that satisfies the reduct is equal to θ, therefore, θ is a stable model of the program. Consider another assignment θ' where all values are the same as in θ, but $\theta'(s) = 3$. Then, $P^{\theta'}$ is the positive-CP: $a \geq 0$, $b \geq 0$, $a \geq b + 3$, $b \geq 8 \leftarrow x$, $x \leftarrow a \geq 5$. The minimal solution that satisfies this positive-CP is M where $M(a) = 3$, $M(b) = 0$, $M(x) = M(y) = false$. Therefore, θ' is not a stable model of the program.

3 Overview of the existing literature

There are several approaches in the literature that aim at removing the standard grounding bottleneck from ASP systems. A majority of these approaches work as follows: they introduce finite domain integer variables and constraints inside the ASP program. The ASP solver passes these to a CP solver while maintaining a Boolean variable to represent the truth value of each constraint that is in the ASP program. For a constraint c that appears in the program, this is done by *reifying* the constraint, i.e., introducing a Boolean variable b to represent whether the constraint is true. The ASP and CP solvers communicate using these introduced Boolean variables. E.g., if b is set true by the ASP solver, then the constraint c is enforced by the CP solver. Since the ASP solver treats CP as a blackbox, it cannot directly learn clauses from the propagation performed by

the CP solver. Examples of systems that use this approach are the AC SOLVER algorithm (Mellarkod et al. 2008), CLINGCON (Gebser et al. 2009) and EZCSP (Balduccini 2009). Recently, some systems have been introduced that overcome the limited learning by using a single solver that supports both founded Booleans as well as standard integer variables and constraints over them. One way to achieve this is to introduce standard integer variables inside an ASP solver, and extending ASP's propagation engine to work like a CP solver (de Cat et al. 2013; Drescher and Walsh 2012). The second approach is given in our earlier work (Aziz et al. 2013a), and extends an existing CP solver with founded Boolean variables and normal rules. To implement the stable model semantics over these, it implements the *source pointer technique* (Simons et al. 2002) to prune *unfounded sets* (Van Gelder et al. 1988) of variables as a propagator, similar to the ASP solver CLASP (Gebser et al. 2012).

Translating in terms of its supported features a specification that is missing in a system is another way to remove standard grounding bottleneck. There are two approaches in the literature to accomplish this. The first approach provides a translation from an ASP program augmented with numeric variables and constraints to a Mixed Integer Program (Liu et al. 2012). As discussed earlier, the non-recursive parts of the program are straight-forward to translate. The non-trivial part is encoding rules that involve positive recursion. This is done using the *level ranking* mapping as given in (Janhunen 2004). The fundamental idea of the translation is that if there is an unfounded set in the solution of the original program, then the mapping contains an inconsistent set of inequalities. The second approach (Drescher and Walsh 2010) encodes entire CP solving into ASP using the well-known eager CP decompositions to SAT. Unfortunately, this *a priori* translation of CP to SAT is already known to be highly inefficient in the CP community where it is much more efficient to translate *lazily* as in *lazy clause generation* (Ohrimenko et al. 2009).

Compared to the standard grounding bottleneck, the focus on the founded grounding bottleneck has been relatively weak. The formalism that is closest to BFASP in terms of removing this bottleneck is Fuzzy Answer Set Programming (FASP) (Nieuwenborgh et al. 2006; Blondeel et al. 2013). The fuzzy atoms in FASP correspond to founded real variables in BFASP, and each logical connective in FASP can be expressed as a rule form in BFASP. We have provided the translation in our previous work (Aziz et al. 2013), showing that BFASP subsumes FASP. Most importantly, from the implementation point of view, the MIP based unfounded set detection algorithm (Janssen et al. 2008) given for FASP only detects unfounded sets in a complete solution, which means that it cannot prune partial solutions that contain unfounded sets. Therefore, the algorithm has a similar shortcoming as CMODELS (Lierler and Maratea 2004) has in case of Boolean unfounded sets. Finally, lack of any good implementation for FASP makes it infeasible to carry out an empirical comparison of BFASP and FASP.

4 Goals and current status of the research

The broader goal of my PhD is to analyze the strengths and implementation techniques of ASP in order to enhance the existing modelling and solving capabilities of constraint solvers. This overall goal can be divided into the following subgoals.

- To define, study, implement, and evaluate BFASP in order to put it forward as a formalism, with an accompanying implementation, that does not suffer from any kind of grounding bottleneck. This subgoal has been completed, and the most important features of BFASP were published in ICLP 2013 (Aziz et al. 2013). The paper defines the semantics of BFASP

and presents an unfounded set algorithm that detects circular sets of bounds and prunes them during search. It presents performance comparison of our implementation of BFASP with ASP on three benchmarks, and with CP on one benchmark, and the results demonstrate the need for BFASP.

Prior to introducing BFASP, we extended a CP solver with founded Boolean variables and normal rules (Aziz et al. 2013a). We implemented two known algorithms for unfounded set detection from the ASP literature inside the CP solver CHUFFED and compared it with CLINGCON on problems that involve inductive definitions as well as standard integer variables.

- To define and study the language of BFASP. As compared to ASP languages like GRINGO that follow a very restrictive grammar, the grammar for BFASP is very permissive and a user can write complex expressions as rules. Therefore, the first task in this subgoal is to simplify these rules to a small set of primitive rules. In other words, we want to extend the *flattening* principles (Stuckey and Tack 2013) used in constraint languages to BFASP. Secondly, ASP grounders use *bottom-up grounding* that generates as few *useless rules* as possible. These are rules that can be removed without affecting the stable solutions of the program. The second task is to generalize ASP bottom-up grounding technique for BFASP. Finally, *magic set rewriting* is a useful technique in logic programming that only instantiates rules that are relevant to a given *query*. Considering variables appearing in constraints and objective function comprise our query, i.e. the set of variables whose values are of interest, the final task is to generalize the magic set transformation for BFASP.

- Identify research areas and benchmarks where BFASP can be applied and that cannot be efficiently solved by the current ASP, CP, and Constraint ASP (Gebser et al. 2009) systems. The doctoral programme could prove especially beneficial with regard to this subgoal.

References

AZIZ, R. A., CHU, G., AND STUCKEY, P. J. 2013. Stable model semantics for founded bounds. *Theory and Practice of Logic Programming 13,* 4–5, 517–532. Proceedings of the 29th International Conference on Logic Programming.

AZIZ, R. A., STUCKEY, P. J., AND SOMOGYI, Z. 2013a. Inductive definitions in constraint programming. In *Proceedings of the Thirty-Sixth Australasian Computer Science Conference*, B. Thomas, Ed. CRPIT, vol. 135. ACS, 41–50.

BALDUCCINI, M. 2009. Representing constraint satisfaction problems in answer set programming. In *ICLP09 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP09)*.

BARAL, C. 2003. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press.

BLONDEEL, M., SCHOCKAERT, S., VERMEIR, D., AND DE COCK, M. 2013. Fuzzy answer set programming: An introduction. In *Soft Computing: State of the Art Theory and Novel Applications*. Springer, 209–222.

BUCHANAN, A., SUNG, J. S., BOGINSKI, V., AND BUTENKO, S. 2013. On connected dominating sets of restricted diameter. *European Journal of Operational Research*.

DE CAT, B., BOGAERTS, B., DEVRIENDT, J., AND DENECKER, M. 2013. Model expansion in the presence of function symbols using constraint programming. In *IEEE International Conference on Tools with Artificial Intelligence*.

DRESCHER, C. AND WALSH, T. 2010. A translational approach to constraint answer set solving. *Theory and Practice of Logic Programming 10,* 4-6, 465–480.

DRESCHER, C. AND WALSH, T. 2012. Answer set solving with lazy nogood neneration. In *Technical Communications of the 28th International Conference on Logic Programming*. 188–200.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. MIT Press, 386.

GEBSER, M., KAUFMANN, B., AND SCHAUB, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187*, 52–89.

GEBSER, M., OSTROWSKI, M., AND SCHAUB, T. 2009. Constraint answer set solving. In *Proceedings of the 25th International Conference on Logic Programming*. Springer, 235–249.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming*. MIT Press, 1070–1080.

JANHUNEN, T. 2004. Representing normal programs with clauses. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence*. 358–362.

JANSSEN, J., HEYMANS, S., VERMEIR, D., AND COCK, M. D. 2008. Compiling fuzzy answer set programs to fuzzy propositional theories. In *Proceedings of the 24th International Conference on Logic Programming*. Springer Berlin Heidelberg, 362–376.

KIM, D., WU, Y., LI, Y., ZOU, F., AND DU, D.-Z. 2009. Constructing minimum connected dominating sets with bounded diameters in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 147–157.

LIERLER, Y. AND MARATEA, M. 2004. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *LPNMR*. 346–350.

LIU, G., JANHUNEN, T., AND NIEMELA, I. 2012. Answer set programming via mixed integer programming. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning*. AAAI Press, 32–42.

MELLARKOD, V. S., GELFOND, M., AND ZHANG, Y. 2008. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence 53,* 1-4, 251–287.

MITCHELL, D. G. 2005. A SAT solver primer. *Bulletin of the EATCS 85*, 112–132.

NIEUWENBORGH, D. V., COCK, M. D., AND VERMEIR, D. 2006. Fuzzy answer set programming. In *Proceedings of Logics in Artificial Intelligence, 10th European Conference, JELIA 2006*. Springer Berlin Heidelberg, 359–372.

OHRIMENKO, O., STUCKEY, P. J., AND CODISH, M. 2009. Propagation via lazy clause generation. *Constraints 14,* 3, 357–391.

ROSSI, F., BEEK, P. V., AND WALSH, T. 2006. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science, New York, NY.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1–2, 181–234.

STUCKEY, P. J. AND TACK, G. 2013. Minizinc with functions. In *Proceedings of the 10th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming*. Number 7874 in LNCS. Springer, 268–283.

VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1988. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, 221–230.

Appendix A MINIZINC Encoding of Minimum Connected Dominating Set with Bounded Diameter

```
int: N;                         %number of nodes
int: E;                         %number of edges
array[1..E] of 1..N: from;      %encodes an edge i (from[i], to[i])
array[1..E] of 1..N: to;
array[1..E] of int: weight;     %weight of an edge
int: K;                         %bound on diameter


array[1..N] of var bool: dom; %whether a node is dominating

% Dominating set constraint
constraint forall (n in 1..N) (
  dom[n] \/ exists(e in 1..E where from[e] = n) (dom[to[e]])
);

% Rules for negative distance
array[1..N,1..N] of var int: d :: founded;

rule forall (n in 1..N) (d[n,n] >= 0 :: head(d[n,n]));
rule forall (e in 1..E, n in 1..N) (
  d[from[e],n] >= d[to[e],n] - weight[e]
    <- dom[from[e]] /\ dom[to[e]] /\ dom[n] :: head(d[from[e],n])
);

% Diameter constraint
constraint forall (u,v in 1..N where u != v) (
  dom[u] /\ dom[v] -> d[u,v] >= -K
);

% Objective to minimize the cardinality of dominating set
solve minimize sum (n in 1..N) (bool2int(dom[n]));

% A toy instance
N=4;
E=6;
K=35;
from   =[1, 2, 2, 3, 3, 4];
to     =[2, 1, 3, 2, 4, 3];
weight =[20,20,30,30,40,40];
```