The Prospect for Answer Sets Computation by a Genetic Model

A. Bertoni, G. Grossi, A. Provetti
Dip. di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico, 39. Milan I-20135, Italy
{*bertoni,grossi,provetti*}*@dsi.unimi.it*

V. Kreinovich and **L. Tari**
Dept. of Computer Science
The University of Texas at El Paso
El Paso, Texas 79968 U.S.A.
{*vladik,luisb*}*@cs.utep.edu*

Abstract

We combine recent results from both Logic Programming and Genetic Algorithms to design a new method for the efficent computation of Answer Sets of logic programs. First of all the problem is reduced to the problem of finding a suitable coloring on directed graphs. Then the problem of finding a suitable coloring is relaxed to a combinatorial optimization problem and solved (in an approximate way) by a continuous discrete time system derived by a genetic model.

Introduction

Answer Sets Programming(ASP) is a new, emergent, type of logic programming where each solution to a problem is represented by an answer set of a function–free logic program[1] encoding the problem itself. The theoretical foundation of Answer Set Programming is the Stable Models semantics of Gelfond and Lifschitz (Gelfond & Lifschitz, 1988). The subsequent research, has clarified the relationship of this semantics with existing ones in Logic Programming and nonmonotonic reasoning, and led researchers to understand how to use it for AI applications.

Several implementations now exist for ASP (also referred to as A-Prolog and SLP), and their performance is rapidly improving; among them are CCalc (McCain & Turner, 1997), DeReS (Cholewiński et al., 1996), dlv (Eiter et al., 1997), SLG (Chen & Warren, 1996) and smodels (Niemelä & Simons, 1998).

In this position paper we outline a new computational scheme for ASP which is based on

i) a reformulation of ASP in terms of particular colorings of oriented graphs and

ii) a heuristic solution to the coloring problem based on the genetic model proposed by some of these authors in (Bertoni et al., 2000).

The scheme above is based on several theoretical considerations that are illustrated next.

First, following the approach described by (Brignoli et al., 1999; Costantini, 2000), the problem of computing the answer set of a given logic program (the instance of the problem) can be *reduced* to the same problem for a different, normally smaller instance, called the *kernel* of the program.

Second. the same works propose a novel graph representation of logic programs, called Extended Dependency Graphs (EDG), which allowed us to characterize stable models of a program in terms of distinguished 2-colorings of its EDG. These colorings are called *admissible.*

In other words, there is a one-to-one correspondence between the answer sets of a kernel program $Ker(\Pi)$ and *admissible* 2-colorings of the related Extended Dependency Graph $EDG(Ker(\Pi))$. Therefore, the ASP problem is rephrased in terms of finding particular colorings of directed graphs; this problem will be called Admissible Coloring Problem (ACP).

Since the ASP problem is NP-complete, so is ACP, hence exact solutions to ACP cannot be found in polynomial time (of course, in the worst case). Our approach, described in the rest of the paper, is to relax the problem to that of optimizing a suitable objective function (called *fitness* from now on).

More precisely, we try to solve ACP by means of a *continuous model of computation.* We propose:

1. to relax ACP to combinatorial optimization problem, by associating each instance of the problem to a suitable fitness function, so that admissible colorings are exactly global maxima of every fitness function,
2. to maximize a fitness function by using the genetic model proposed in (Bertoni et al., 2000). In this model, the fitness function corresponds to a discrete-time system whose attractors are local maxima of the fitness function itself.

From Answer Sets to graph coloring

In this Section we show a reduction from ASP computation to a particular type of graph coloring introduced in (Brignoli et al., 1999).

The first step consists in reducing the original program Π to a new, smaller program $Ker(\Pi)$ which is equivalent to the original as far as existence and number of answer sets are concerned.

[1] Or, via a syntactic transformation, a restricted default theory or even a $DATALOG^{\neg}$ program.

By lack of space, we cannot describe kernelization in detail, since it is based on several semantics considerations, such as that saying that stable models always contain all atoms that are true.

Three features make kernelization important:

1. $Ker(\Pi)$ is obtained by applying polinomial-time algorithms, the most important one being Dix et al. rewriting system (Brass et al., 1999). Therefore, in short time we are able to create a smaller instance for the same problem.
2. kernel programs have a standard presentation[2], e.g. there are no facts, and no positive conditions in rule bodies. This makes their graph representation (see below) and the subsequent algorithms easier to describe and implement.
3. each and all answer sets of Π consists of an answer set for $Ker(\Pi)$ *augmented* with atoms that are either

   i) true w.r.t. the Well-founded semantics (WFS), and therefore part of any answer set, or

   ii) obtained by simple forward reasoning from the relative answer set of $Ker(\Pi)$, $W^{+}(\Pi)$ (true atoms under WFS) and the rules of the program. As a result, we can *go back* from answer sets of the kernel to those of the original program in low polinomial time.

Theorem 1 *(Brignoli et al., 1999) A program* Π *has a stable model if and only if* $Ker(\Pi)$ *does.*

Now, we proceed to give a graph representation for logic programs, altough in this work we will employ it only for kernel programs. To do so, we introduce indexes for rules defining the same atom, e.g.,

$p \leftarrow not\ a.$
$p \leftarrow not\ b.$
$\ldots$

will be denoted

$p^1 \leftarrow not\ a.$
$p^2 \leftarrow not\ b.$
$\ldots$

Definition 1 *(Extended dependency graph) (EDG) For a logic program* Π*, its Extended Dependency Graph* $EDG(\Pi)$ *is the directed finite labeled graph* $\langle V, E, \{+,-\}\rangle$ *defined below.*

V.1 *For each rule in* Π *there is a vertex* $a_i^{(k)}$*, where* a_i *is the name of the head and* k *is the index of the rule in the definition of* a_i*, all* a_i*s;*

V.2 *for each atom* u *never appearing in a head, there is a vertex simply labeled* u*;*

E.1 *for each* $c_j^{(l)} \in V$*, there is a positive edge* $\langle c_j^{(l)}, a_i^{(k)}, +\rangle$*, if and only if* c_j *appears as a positive condition in the k-th rule defining* a_i*, and*

E.2 *for each* $c_j^{(l)} \in V$*, there is a negative edge* $\langle c_j^{(l)}, a_i^{(k)}, -\rangle$*, if and only if* c_j *appears as a negative condition in the k-th rule defining* a_i*.*

[2]In fact, they are described in (Costantini & Provetti 2001) as a canonical form.

It is easy to check that EDGs are isomorphic to programs (Brignoli et al., 1999). Since kernel programs have negative conditions only, in the rest of this article we will use a reduced notation, $\langle V, E\rangle$.

For a given a program Π let $\langle V, E\rangle = EDG(\Pi)$, where for simplicity $V = \{1, \ldots, l\}$, we define **coloring** as a function $x : V \to \{1, 0\}$, denoted with the characteristic vector $(x_1, \ldots, x_l)$.

Definition 2 (Non-admissible coloring) *A coloring* $x : V \to \{0,1\}$ *is non-admissible for the graph* $EDG(\Pi)$ *if and only if*

1. $\exists i\ x_i = 1$ *and* $\exists j\ (i, j, -) \in E$ *and* $x_j = 1$ *(violation of type I), or*
2. $\exists i\ x_i = 0$ *and* $\forall j\ (j, i, -) \in E$ *and* $x_j = 0$ *(violation of type II).*

A coloring for $EDG(\Pi)$ is **admissible** unless it is not admissible.

From colorings back to Answer sets

For any interpretation $S \subseteq$ of Π, an **associate coloring** col_S is a total function $V \to \{0,1\}$ that satisfies the condition: $a_i \in S$ if and only if $\exists k. col_S(a_i^{(k)}) = 1$.
Clearly, more than one coloring can be associated to S.

Theorem 2 *(Brignoli et al., 1999)*

An interpretation S *is a stable model of* Π *if and only if there is an associated coloring* col_S *which is admissible for* $EDG(\Pi)$*.*

Example 1 *Let un consider the following logic program:*

$p \leftarrow not\ a.$
$p \leftarrow not\ b.$
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

Clearly, there are two stable models, $S_1 = \{p, a\}$ *and* $S_2 = \{p, b\}$*. The EDG of the program has 4 vertices, i.e.,* $\{p, p', a, b\}$ *and 4 arcs. It has two admissible colorings, i.e.,* $x(p) = x(b) = 1$ *(everything else being mapped on 0) and* $x'(p) = x'(b) = 1$ *associated to* S_1 *and* S_2*, respectively.*

The genetic model

Genetic algorithms are probabilistic search algorithms inspired by mechanisms of natural selection. They have received considerable attention because of their applications to several fields such as optimization, adaptive control, and others (Goldberg, 1989; Holland, 1992). By simplifying natural laws, genetic algorithms simulate reproductive processes over a population of individuals or genotypes, typically represented by binary strings of fixed length l, in an arbitrary environment.

The states of a simple genetic system are populations represented by multi-sets of binary words, and its evolution is obtained by applying suitable stochastic rules.

In the model presented in (Bertoni et al., 2000) the stochastic rules are: *recombination* and *mutation*. In the recombination, also called *bit-based simulated crossover*, a weighted average of the alleles of the individuals along each bit position is done; these statistics are used to produce offsprings, whose alleles in different positions are independently generated. The use of this recombination rule instead of the biologically inspired one-point crossover, seems to present advantages in the simulation efficiency and it makes the analysis easier. In the mutation random changes in the genotypes are introduced.

More formally, given an arbitrary, fixed integer $n > 0$, a population P is a multi-set of n elements of Ω, where $\Omega = \{0,1\}^l = \{\omega_1, \dots, \omega_{2^l}\}$ is the class of length l binary strings. The population P can be represented by the "frequency vector" $\mathbf{F} = (F_{\omega_1}, \dots, F_{\omega_{2^l}})$, where $F_{\omega_k} = \frac{n_k}{n}$ and n_k is the number of occurrences of the word ω_k in P.

Let the pseudo boolean function $f : \Omega \to \mathbf{N}$ be the fitness function: the evolution of the genetic system can be described by the following steps:

1. at time 0 the state of the system is the initial population P_0;
2. if at time t the state of the system is the population P (represented by $\mathbf{F}$), then the population at time $t+1$ is obtained by applying the following stochastic rules:

   recombination:

   (a) calculate the ratio
   $$\phi_{k\mathbf{F}} = \frac{E_{\mathbf{F}}[x_k \cdot f]}{E_{\mathbf{F}}[f]}, \text{ for } k = 1 \dots l$$
   where $E_{\mathbf{F}}$ is the expectation under the probability distribution $\mathbf{F}$ on Ω, and $x_k(\omega)$ is the function which return the k-th bit of the word ω;[3]

   (b) generate $\{\omega_{s_1}, \dots, \omega_{s_n}\}$ with probability $\phi_{k\mathbf{F}}$ to obtain 1 in position k, independently from s_i and k, for $1 \le i \le n$ and $1 \le k \le l$;

   mutation: flip k-th bit of word ω_{s_i} with probability $0 < \eta \le \frac{1}{2}$ for any $1 \le k \le l$ and $1 \le i \le n$.

As it is proved in (Bertoni et al., 2000), in case of infinite populations the stochastic genetic system becomes a discrete time non linear deterministic system whose states are vectors $\mathbf{g} = (g_1, \dots, g_l) \in [0,1]^l$, where g_k is the probability to obtain a word with 1 in position k.

In order to derive the dynamics of the system, consider now the fitness function $f : \Omega \to \mathbf{N}$. The function f is a pseudo boolean function and therefore it can be represented by a multivariate polynomial of degree at most one in each variable defined on $[0,1]^l$ and coincident with f on Ω, that is:

$$f(x_1, \dots, x_l) = \sum_{y_1, \dots, y_l \in \{0,1\}} \alpha_{y_1 \cdots y_l} x_1^{y_1} \cdots x_l^{y_l}$$

For sake of simplicity and with abuse of notation, we denote with f both the function and the associated polynomial. Notice that, since f is a polynomial of degree at most one in each variable, global maxima of f are on elements of $\{0,1\}^l$. Moreover for every k ($1 \le k \le l$) we can rewrite f as

$$f(\mathbf{x}) = x_k b_k(\mathbf{x}) + a_k(\mathbf{x}),$$

where $b_k(\mathbf{x})$ and $a_k(\mathbf{x})$ are polynomials that do not depend on x_k and whose variables have degree at most one. Observe that $b_k(\mathbf{x}) = \frac{\partial}{\partial x_k} f(\mathbf{x})$ and $a_k(\mathbf{x}) = f(\mathbf{x}) - x_k b_k(\mathbf{x})$.

As shown in (Bertoni et al., 2000), in case of infinite population the genetic model can be described by the equations

$$g_k(t+1) = g_k(t) \frac{b_k(\mathbf{g}(t)) + a_k(\mathbf{g}(t))}{g_k(t) b_k(\mathbf{g}(t)) + a_k(\mathbf{g}(t))} (1 - 2\eta) + \eta.$$

For $\eta \approx 0$ the following result characterizes the attractor of the previous system and justifies the use of the genetic system as local optimizer of the fitness function:

Theorem 3 *Every element $\mathbf{x} \in \Omega$ is a fixed point for the discrete time deterministic system and every attractor is an element of Ω; moreover, if $b_k(\mathbf{x}) \neq 0$ for all k, then $\mathbf{x} \in \Omega$ is an attractor if and only if*

$$x_k = HS\,(b_k(\mathbf{x})) \quad k = 1, \dots, l,$$

where $HS\,(x) = \begin{cases} 1, & \textit{if } x \ge 0; \\ 0, & \textit{otherwise}. \end{cases}$

It can be proved that a good initial condition (population) is that of maximal uncertainty, i.e. $g(0) = (\frac{1}{2}, \dots, \frac{1}{2})$. In conclusion, the algorithm GC is sketched below

---

Input: A fitness function $f : \Omega \to \mathbf{N}$,
mutation rate η;

for $k = 1,\ l$ **do** $G_k := \frac{1}{2}$;

while (not end cond.) **do**
  for $k = 1, l$ **do**
    $G_k := (1 - 2\eta) G_k \frac{b_k(\mathbf{G}) + a_k(\mathbf{G})}{f(\mathbf{G})} + \eta$;

for $k = 1, l$ **do**
  if $G_k \ge \frac{1}{2}$ **then** $X_k := 1$
    else $X_k := 0$;

Output: $\mathbf{X} \in \Omega$.

---

[3] In other words, the numerator of the ratio $\phi_{k\mathbf{F}}$ represents the sum of the fitness values of the words with 1 in position k.

A fitness function for ACP

In this section we define a fitness function for the *ACP* problem and we show how GC can be used to optimize it.

Let Π be a kernel program and $G = \langle V, E \rangle$ be its extended dependency graph, where $V = \{1, \ldots, l\}$ is the set of vertices and $E \subseteq V \times V$ the set of arcs of G; we say that an arc $e = (i, j)$ is *incident* with the vertex j and write $i \to j$. Now, let $M = (m_{ij})_{l \times l}$ be the adiacency matrix of G, i.e. $m_{ij} = 1$ iff $i \to j$ otherwise $m_{ij} = 0$, and let $A = (m_{ij})_{l \times l}$ be the simmetric matrix such that $a_{ij} = 1$ iff (i, j) or (j, i) belong to E, otherwise $a_{ij} = 0$; the set of edges (i, j) such that $a_{ij} = 1$ and $i < j$ is denoted by E'. The sets $\mathrm{In}_i = \{j \in V : j \to i\}$ and $\mathrm{Out}_i = \{j \in V : i \to j\}$ denote respectively *in-neighborhood* and *out-neighborhood* of the vertex i.

We are now able to give a fitness function $f : \{0,1\}^l \to \mathbf{N}$ whose maxima coincide with the admissible colorings for G.

To do this, we introduce boolean functions which consider violations of I and II type as penalty factors, defined as follows:

- *violations of type I*: for each edge $(i, j) \in E'$ let us introduce the polynomial
$$g_{ij}(x_1, \ldots, x_l) = 1 - x_i x_j.$$
It is easy to show that sufficient and necessary condition for avoiding violations of type I is:
$$\begin{aligned} \sum_{(i,j) \in E'} g_{ij}(x_1, \ldots, x_l) &= \sum_{i<j} a_{ij}(1 - x_i x_j) \\ &= |E'| - \sum_{i<j} a_{ij} x_i x_j \\ &= |E'|. \end{aligned}$$
- *violations of type II*: for each vertex i let us introduce the polynomial
$$r_i(x_1, \ldots, x_l) = 1 - (1 - x_i) \prod_{j \in \mathrm{In}_i} (1 - x_j),$$
Also in this case it is easy to show that sufficient and necessary condition for avoiding violations of type II is:
$$\sum_{i=1}^{l} r_i(x_1, \ldots, x_l) = l.$$

The fitness function we find is then given by the sum of the polynomials defined above:

$$\begin{aligned} f(x_1, \ldots, x_l) = \sum_{i=1}^{l} r_i(x_1, \ldots, x_l) \\ + \sum_{(i,j) \in E'} g_{ij}(x_1, \ldots, x_l). \end{aligned}$$

As regards the admissibility of a given coloring we can than conclude

Theorem 4 *A coloring $(y_1, \ldots, y_l)$ for the extended dependency graph $G = \langle V, E \rangle$ is admissible iff*

$$\begin{aligned} f(y_1, \ldots, y_l) &= \sup_{(x_1, \ldots, x_l) \in \Omega} f(x_1, \ldots, x_l) \\ &= l + |E'|, \end{aligned}$$

where $l = |V|$.

To apply the genetic algorithm GC to ACP we only need to derive the b_k polynomials. For the fitness function specified above we have:

$$\begin{aligned} b_k(x_1, \ldots, x_l) = \prod_{j \in \mathrm{In}_k} (1 - x_j) + \\ \sum_{i \in \mathrm{Out}_k} (1 - x_i) \prod_{j \in \mathrm{In}_i \setminus \{k\}} (1 - x_j) - \sum_j a_{kj} x_j. \end{aligned}$$

Recall that b_k is the partial derivative of f w.r.t. x_k.

Relationship with other approaches

For nonmonotonic reasoning, work along the lines of Kautz and Selman's WALKSAT has being carried out, among others, in the Default Logic framework [4] by Saubion et al. (Nicolas et al., 2000).

Perhaps the most similar approach is that of Blair (Blair et al., 1999) where he relaxes the ASP problem in two directions: the function associated to the logic program becomes continuous and its fixed points are studied over a continuous time domain. Vice versa, our system is discrete-time only.

Relationship with SAT solvers A substantial amount of work on efficient approximation of logical decision problems focuses on solving the propositional satisfiability problem (SAT). Several solutions (WALKSAT, WSAT etc.) are available, and in principle, they could be applied to ASP by reducing it to satisfaction of a CNF formula.

Even tough we are not yet able to show enough experimental evidence to support our conjecture that *direct reduction of ASP to SAT may not be a good strategy for efficient ASP computation,* because the translation would not reflect the causal aspect of the logic programming '$\leftarrow$' (or ':−') connective as opposed to material implication '$\subset$'[5].

Acknowledgements

Luis Tari is supported by the MIE, Model Institution for Education with a *Research Experience for Undergraduates* scholarship for Fall 2000.

Howard Blair, Stefania Costantini, Ottavio D'Antona, Michael Gelfond and Frédéric Saubion

[4] Indeed, ASP can be seen as a special case of Default logic with the restriction that conclusions be atomic. However, computing extensions (models) of default logic theories is a Σ_2^p problem, whereas ASP is in Σ_1^p (see (Liberatore, 1999)).

[5] (McCain & Turner, 1997), among others, discuss this aspect.

have provided useful advice on several aspects of this work.

References

Apt, K. R. and Bol, R., 1994. *Logic programming and negation: a survey,* J. of Logic Programming, 19/20.

Bertoni A., Campadelli P., Carpentieri M. and Grossi G., 2000. *A Genetic Model: Analysis and Application to MAXSAT*, Evolutionary Computations 8(3):291–310.

Baral, C. and Gelfond. M., 1994. *Logic programming and knowledge representation,* J. of Logic Programming, 19/20.

Blair, H.A., Dushin, F., Jakel, D.W., Rivera, A.J. and Sezgin, M., 1999. *Continuous models of computation for logic programs: importing continuous mathematics into logic programming's algorithmic foundations* In K.R. Apt, editor, The Logic Programming Paradigm: A 25-Year Perspective. Springer, pp. 231–255.

Brass S., Dix J., Freitag B., Zukowski, U., 1999. *Transformation-based Bottom-up Computation of the Well–Founded Model.* J. of Logic Programming.

Brignoli G., Costantini S., D'Antona O. and Provetti A., 1999. *Characterizing and computing stable models of logic programs: the non-stratified case.* Proc. of Conference on Information Technology, pp. 197–201.

Chen W., and Warren D.S., 1996. *Computation of stable models and its integration with logical query processing,* IEEE Trans. on Data and Knowledge Engineering, 8(5):742–747.

Cholewiński P., Marek W. and Truszczyński M., 1996. *Default reasoning system DeReS.* Proc. of KR96, Morgan-Kauffman, pp. 518-528.

Costantini S., 2000. *About Stable Models of Non-Stratified LogicC Programs.* Proc. of IEEE Conf. on Logic in Computer Science LICS 2000.

Costantini S. and Provetti A., 2001. *Canonical Forms for Answer sets Programming.* Univ. of Milan Technical Report, to be submitted.

Dung P.M., 1992. *On the Relation between Stable and Well–Founded Semantics of Logic Programs,* Theoretical Computer Science, 105.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F., 1997. *A deductive system for non-monotonic reasoning.* Proc. Of the 4th LPNMR Conference, Springer Verlag, LNCS 1265, pp. 363–374.

Gelfond, M. and Lifschitz, V., 1988. *The stable model semantics for logic programming,* Proc. of 5th ILPS conference, pp. 1070–1080.

M. Gelfond and V. Lifschitz., 1991. *Classical negation in logic programs and disjunctive databases.* New Generation Computing, pp. 365–387.

Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Holland, J. H., 1992, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge.

Kautz H. and Selman B., 1996. *Pushing the envelope: Planning, Propositional Logic and Stochastic Search*, Proc. of AAAI'96.

Liberatore P., 1999. *Algorithms and Experiments on Finding Minimal Models.* Technical Report of University of Rome "La Sapienza."

Lifschitz V., 1999. *Answer Set Planning.* Proc. of LPNMR'99 Conference.

McCain N. and Turner H., 1997. *Causal theories of actions and change.* Proc. of AAAI'97 Conference, pp. 460–465.

Marek, W., and Truszczyński M., 1999. *Stable models and an alternative logic programming paradigm.* The Journal of Logic Programming.

Nicolas P., Saubion F. and Stéphan I., 2000. *Genetic Algorithms for Extension Search in Default Logic* Proc. of NMR'00 Conference.

Niemelä I. and Simons P., 1998. *Logic programs with stable model semantics as a constraint programming paradigm.* Proc. of NM'98 workshop. Extended version submitted for publication.

A. Provetti and L. Tari, 2000. *Answer Sets Computation by Genetic Algorithms, Preliminary Report.* Proc. of GECCO 2000, "late-breaking papers" track. pp. 303–308.

Vose, M. D., 1996. *Modeling Simple Genetic Algorithms.* Evolutionary Computation, 3(4):453–472.

Whitley, D., 1994. *A Genetic Algorithm Tutorial.* Lecture notes.