# Genes and Ants for Default Logic

**Pascal Nicolas, Frédéric Saubion, Igor Stéphan**
LERIA, Université d'Angers
2 Bd Lavoisier, F-49045 Angers Cedex 01
pascal.nicolas | igor.stephan | frederic.saubion@univ-angers.fr

### Abstract

Default Logic and Logic Programming with stable model semantics are recognized as powerful frameworks for incomplete information representation. Their expressive power are suitable for non monotonic reasoning, but the counterpart is their very high level of theoretical complexity. The purpose of this paper is to show how heuristics issued from combinatorial optimization and operation research can be used to built non monotonic reasonning systems.

## Introduction

*Default Logic* (Reiter 1980) appears as a natural framework in order to formalize common sense reasoning from incomplete information and then it allows *non monotonic reasoning*. The non monotonicity of this logic relies on the fact that adding a new axiom may invalidate previous deductions. Therefore, these deductions are only plausible and their set is called an extension. Due to its level of theoretical complexity ($\Sigma_2^p - complete$ (Gottlob 1992)), the computation of an extension is a great challenge. Previous works (Cholewiński *et al.* 1999; Niemelä 1995; Schaub 1998) have already investigated this computational aspect of default logic and even if some systems have good performances on certain classes of default theories, there is no very efficient system for general extension calculus. The Stable Model Semantics for Logic Programming (Gelfond & Lifschitz 1988) can be viewed as a particular case of Default Logic (Bidoit & Froidevaux 1991) and its complexity is $NP - complete$ (Bidoit & Froidevaux 1991). In many cases the framework of logic programs is suitable for the encoding of many problems and the efficient system *smodels* (Niemelä & Simons 1996) is available for this task.

In this paper, we adopt the point of view of Default Logic and deal with Stable Models of Logic Program as extensions of a default theory without refeering to their specific semantics. We present different heuristics and show how they can be used to handle this extension computation problem. The purpose of these algorithms is to progressively improve a given initial configuration in order to reach a solution. The three general following approaches are considered here. Genetic Algorithms are based on the principles of natural selection. Populations of possible solutions evolve through a process of mutation and crossover in order to generate better and better configurations. Ant Colony Optimization is inspired by the observation of the collective behaviour of ants when they are seeking food. The possible solutions are now paths inside a graph, and paths become better and better during the process. At last, Local Search relies on an incremental improvement of a potential solution to a given problem by local movements from a configuration to its neighbours. Part of this work has been developped in (Nicolas, Saubion, & Stéphan 2000b) and (Nicolas, Saubion, & Stéphan 2000a).



## Problem Description

In Default Logic (Reiter 1980) knowledge is represented by means of a *default theory* $(W, D)$ where $W$ contains the "sure" knowledge (in this work it is a set of propositional formulas) and $D$ is a set of *default rules* (or defaults). A *default* $\delta = \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$ is an inference rule ($\alpha$, $\gamma$ and all $\beta_i$ are propositional formulas) whose meaning is "if the *prerequisite* $\alpha$ is proved, and if for all $i = 1, \ldots, n$ each *justification* $\beta_i$ is individually consistent (in other words if nothing proves its negation) then one concludes the *consequent* $\gamma$[1]".

Given a default theory it is possible to infer a set of plausible conclusions called an *extension* and defined by Reiter as the fixpoint of a special operator. But, we prefer to recall here the equivalent following pseudoiterative characterization because it is closer to our approach of the extension computation.

**Theorem 1** *(Reiter 1980) Let $(W, D)$ be a default theory and $E$ a formula set. We define $E_0 = W$ and for all $k \geq 0$,*

$$E_{k+1} = Th(E_k) \cup \left\{ \gamma \;\middle|\; \begin{array}{l} \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D, E_k \vdash \alpha, \\ and\ E \not\vdash \neg\beta_i, \forall i = 1, \ldots, n \end{array} \right\}$$

*Then, $E$ is an extension of $(W, D)$ iff $E = \bigcup_{k=0}^{\infty} E_k$.*

For a set of formulas $E$, $Th(E)$ denotes as usual the set of logical consequences of $E$, and $E \vdash \phi$ has its common sense of deduction in classical logic. It is important to note that a default theory may have one or multiple extensions and

[1] If $\delta$ is a default rule, $pre(\delta)$, $jus(\delta)$ and $cons(\delta)$ respectively denotes the prerequisite, the set of justifications and the consequent of $\delta$. These definitions will be also extended for sets of defaults.

sometimes no extension at all. In the theorem 1 we can remark that $E$, the whole extension to build, is used in its own definition. This non constructive characterization is also an argument to choose a "guess and check" method as we have done in this work.

Furthermore, given a default theory $(W, D)$, to compute its extension $E$ is equivalent to find its *Generating Default Set* $\Delta$ since $E = Th(W \cup cons(\Delta))$ (Risch 1996).

**Definition 1** *Let $E$ be an extension of a default theory $(W, D)$, its Generating Default Set is*

$$GD(W,D,E) = \left\{ \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D \;\middle|\; \begin{array}{c} E \vdash \alpha \; and \\ E \not\vdash \neg\beta_i, \forall i = 1, \ldots, n \end{array} \right\}$$

To end this technical part, we recall the notions of *grounded* default set and *incrementally non-conflicting* default set.

**Definition 2** *(Schwind 1990) Given a default theory $(W, D)$, a set of defaults $\Delta \subseteq D$ is grounded if $\Delta$ can be ordered as a sequence $(\delta_1, \ldots, \delta_n)$ satisfying : $\forall i = 1, \ldots, n, W \cup cons(\{\delta_1, \ldots, \delta_{i-1}\}) \vdash pre(\delta_i)$.*

**Lemma 1** *(Schwind 1990) Every generating default set is grounded.*

**Definition 3** *(Nicolas, Saubion, & Stéphan 2000a) Given a default theory $(W, D)$, a set of defaults $\Delta \subseteq D$ is incrementally non-conflicting if $\Delta$ can be ordered as a sequence $(\delta_1, \ldots, \delta_n)$ satisfying : $\forall i = 1, \ldots, n, \forall \beta \in jus(\delta_i), W \cup cons(\{\delta_1, \ldots, \delta_i\}) \not\vdash \neg\beta$.*

This last definition is strongly related to the non constructive characterization of an extension as mentioned above.

Given a default theory $(W, D)$, the problem we address in this paper consists in computing an extension $E$ of this theory. An Extension Computing Problem (ECP) can be defined w.r.t. our heuristic approach by the following components :

**Definition 4 ECP**

- *A default theory $(W, D)$*
- *The set $\mathcal{CGD} = 2^D$ of possible configurations called candidate generating default sets.*

These candidate generating default sets characterize associated candidate extensions which can be defined as :

**Definition 5** *Given a Default theory $(W, D)$, a candidate generating default set $C \in \mathcal{CGD}$, the candidate extension associated to $C$ is*

$$CE(W, D, C) = Th(W \cup \{cons(\delta) \mid \delta \in C)\}$$

Given an $ECP$, a solution is a candidate generating default set $C \in \mathcal{CGD}$ such that $CE(W, D, C)$ is an extension w.r.t. theorem 1.

The last step of our heuristic approach consists in defining an evaluation function in order to compute the fitness of a candidate generating default set w.r.t. to the notion of solution. This evaluation relies on the four intermediate functions described below.

$f_0$ rates if the candidate extension is consistent or not.

$$f_0(C) = \begin{cases} 0 & if \; CE(C) \text{ is consistent} \\ 1 & otherwise \end{cases}$$

$f_1$ rates the correctness of the candidate generating default set with respect to the definition 1.

$$f_1(C) = \Sigma_{i=1}^{n} \pi(\delta_i) \; where \; n = card(D)$$

with $\pi$ defined as follows.

| $\delta_i \in C$ | $CE(C) \vdash \alpha_i$ | $\exists j, CE(C) \vdash \neg\beta_i^j$ | $\pi$ |
|---|---|---|---|
| $true$ | $true$ | $false$ | 0 |
| $true$ | $true$ | $true$ | $k$ |
| $true$ | $false$ | $true$ | $k$ |
| $true$ | $false$ | $false$ | $k$ |
| $false$ | $true$ | $false$ | $k$ |
| $false$ | $true$ | $true$ | 0 |
| $false$ | $false$ | $true$ | 0 |
| $false$ | $false$ | $false$ | 0 |

$k$ is a positive number that represents a penalty given to each default that has been wrongly applied or wrongly not applied.

$f_2$ rates the level of groudedness of the candidate generating default set.

$$f_2(C) = card(\Gamma)$$

where $\Gamma$ is the biggest grounded set $\Gamma \subseteq CGD(C)$.

$f_3$ definitely checks this property

$$f_3(C) = \begin{cases} 0 & if \; CE(C) \text{ is grounded} \\ 1 & otherwise \end{cases}$$

Then, we can give the whole definition of the evaluation function.

**Definition 6** *Given a Default theory $(W, D)$, a candidate generating default set $C \in \mathcal{CGD}$, the evaluation of $C$ is defined by* $eval\colon \mathcal{CGD} \to \mathbb{Z} \cup \{\top, \bot\}$

*if* $f_0(C) = 1$
*then* $eval(C) = \top$
*else* *if* $f_1(C) = 0$ *and* $f_3(C) = 0$
*then* $eval(C) = \bot$
*else* $eval(C) = f_1(C) - f_2(C)$

**Theorem 2** *A solution of an ECP is a set $C \in \mathcal{CGD}$ such that $eval(C) = \bot$.*

We now describe the different methods that we propose to solve an ECP.

## Genetic Algorithms

Genetic Algorithms (Michalewicz 1996; Holland 1975) are based on the principle of natural selection. We first consider a *population* of individuals which are represented by their *chromosomes*. Each chromosome represents a potential solution to the given problem. An evaluation process and genetic operators determine the evolution of the population in order to get better and better individuals. The different components of a genetic algorithm are:

1. a representation of the possible configurations : in most cases, chromosomes will be strings of bits representing its *genes*,
2. a way to generate an initial population,
3. an *evaluation function*: it rates each potential solution,
4. genetic operators that define the evolution of the population : two different operators will be considered : *Crossover* allows to generate two new chromosomes (the offsprings) by crossing two chromosomes of the current population (the parents), *Mutation* arbitrarily alters one or more genes of a selected chromosome,
5. parameters : maximum population size $p_{size}$ and probabilities of crossover $p_c$ and mutation $p_m$. We choose $p_{size} \mid \exists N, p_{size} = \frac{N(N+1)}{2}$.

A representation scheme consists of the two following elements : a chromosome language $\mathcal{G}$ defined by a chosen size and an interpretation mapping to translate chromosomes in term of generating default set, which provides the semantics of the chromosomes. In our context, for each default $\frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma}$ we encode in the chromosome the prerequisite $\alpha$ with one bit, and all justifications $\beta_1, ..., \beta_n$ conjointly with one other bit. Therefore, given a set of defaults $D = \{\delta_1, \cdots, \delta_n\}$ the size of the chromosome will be $2n$ and the chromosome language $\mathcal{G}$ is the regular language $(0+1)^{2n}$ (i.e. strings of $2n$ bits). Given a chromosome $G \in \mathcal{G}$, $G|_i$ denotes the value of $G$ at occurrence $i$. Occurrences of $G$ are elements of $\{1..2n\}$. The interpretation mapping, defining the semantics of the previous chromosomes (the semantics of chromosome is also called its phenotype), can be formally described as :

**Definition 7** *Given the set of default D and chromosome language $\mathcal{G}$, an interpretation mapping is defined as*

$\phi$: $\mathcal{G} \times D \to \{true, false\}$ *such that :*

$$\forall \delta_i \in D, \phi(G,\delta_i) = \begin{cases} true \text{ if } G|_{2i-1} = 1 \text{ and } G|_{2i} = 0 \\ false \text{ in other cases} \end{cases}$$

Therefore, the chromosomes encode the candidate generating default sets as :

**Definition 8** *Given a default set D, a chromosome $G \in \mathcal{G}$, the candidate generating default set associated to G is :*

$$CGD(D,G) = \{\delta_i \in D \mid \phi(G,\delta_i) = true\}$$

Intuitively, for a default $\delta_i$, if $G|_{2i-1} = 1$ then its prerequisite is considered to be in the candidate extension and if $G|_{2i} = 0$ no negation of its justifications is assumed to belong to the candidate extension induced by $G$. $CE(W,D,G)$ and $CGD(D,G)$ will be simply denoted $CE(G)$ and $CGD(G)$ when it is clear from the context. Remark that since we have to compute the set of logical consequences of $W$ and of the consequents of the supposed applied defaults, a theorem prover will be needed in our system.

**Example 1** *Let* $(W,D) = (\{a\}, \{\frac{a:b}{c}, \frac{a:\neg c}{\neg b}, \frac{d:e}{f}\})$ *be a default theory. We get :* $CGD(100011) = \{\frac{a:b}{c}\}$ *and* $CE(100011) = Th(\{a,c\})$ *which is really an extension but also* $CGD(101011) = \{\frac{a:b}{c}, \frac{a:\neg c}{\neg b}\}$ *and* $CE(101011) = Th(\{a,c,\neg b\})$ *which is not an extension.*

Generation of the initial population is crucial to the efficiency of genetic algorithms. The most simple way is a random generation but this does not take into account the considered default theory. A more efficient way consists in generating chromosomes with already grounded phenotypes. For a default theory $(W,D)$, the useful subset of $D$ is always grounded but usually $D$ is not a generating default set. So, the interesting phenotypes are the grounded (consistent) subsets of $D$. We introduce a *probability of insertion* of a default in the candidate generating default set $p_i$ to randomly create a candidate and we randomly associate to each default $\delta$ of $D$ a number $p_\delta \in [0,1]$. The induction definition below gives by fixpoint the candidate generating default set $\Delta_\infty$.

- $\Delta_0 = \emptyset, D_0 = D$,
- $\forall j > 0, \forall \delta \in D_{j-1}, W \cup cons(\Delta_{j-1}) \vdash pre(\delta)$,

$$\begin{aligned} \Delta_j &= \Delta_{j-1} \cup \{\delta\} \; if \; p_\delta < p_i \; and \\ & \quad\; W \cup cons(\Delta_{j-1} \cup \{\delta\})) \not\vdash \bot \\ &= \Delta_{j-1} \; otherwise \\ D_j &= D_{j-1} \setminus \{\delta\} \end{aligned}$$

Then a chromosome $G_\infty$ can be chosen randomly from $\{G | CGD(D,G) = \Delta_\infty\}$. We also guarantee that all the chromosomes of the initial population are different.

The process is similar to generate an initial population with incrementally non-conflicting grounded phenotypes. The following condition is added to the inductive part of the construction :

$$\forall i = 1, \ldots, n, \forall \beta \in jus(\delta), W \cup cons(\Delta_{j-1} \cup \{\delta\}) \not\vdash \neg\beta$$

However, we never completely check if all defaults are not conflicting together because our goal is not to build a generating default set. All our thesis is this task is too difficult for a classical algorithm so we just try to give good starting points of our searching method.

The purpose of the selection stage is, starting from an initial population $P$, to generate a selected population $P_{sel}$ containing chromosomes with the best rates according to the evaluation function. Genetic operators, which define the evolution of the population, will be applied on this intermediate population to get the next population deriving from the initial $P$. The selection process is based on an ordering $\prec$ of the individuals, natural extension of the usual ordering of $\mathbb{Z}$ extended with: $\forall x \in \mathbb{Z}, x \prec \top$ and $\forall x \in \mathbb{Z}, \bot \prec x$. Given a population $P$ of size $p_{size}$, we built an ordered population

$$P_\prec = (G_i)_{i>1} \; such \; that$$
$$\begin{cases} \forall i,j, \; i < j \Rightarrow eval(G_i) \preceq eval(G_j) \\ \forall i,j, \; i \neq j \Rightarrow G_i \neq G_j. \end{cases}$$

The first condition implies that the chromosomes are ordered w.r.t. to their evaluation and the second condition implies that two identical chromosomes are represented only once in $P_\prec$. Note that if two chromosomes have the same evaluation value, they are ordered arbitrarily.

We choose the ranking selection to generate the parent population. Remind that the population size is such that $\exists N, p_{size} = \frac{N(N+1)}{2}$, i.e. $p_{size} = \Sigma_{k=1}^{N} k$.

To keep a large diversity of selected chromosomes as parents, we introduce a Hamming distance $Hd$ that parents must respect. Hamming distance is the number of differing bits between two binary chromosomes. We define the family of sets of chromosomes $(P_i)_{i>1}$ as follows:

- $P_0 = \emptyset$,
- $\forall G \in P_{i-1}, G_i$ the $i^{th}$ chromosome of $P_{\prec}$,
  if $Hamming\_distance(G, G_i) \geq Hd$ then $P_i = P_{i-1} \cup \{G_i\}$ else $P_i = P_{i-1}$.

The selected population $P_{sel}$ is defined as $P_i$ with $card(P_i) = N$. The parents population $P_{parents}$, that will be used for crossover and mutation, is a multiset of chromosomes such that each $G_i,\ i < N$ in $P_{sel}$ occurs $N - i + 1$ times in $P_{parents}$. This construction is required to preserve the maximum size of the population $p_{size}$.

As mentioned before, genetic operators are now applied on the selected population $P_{parents}$. Crossover is performed in the following way:

- select randomly two chromosomes in $P_{parents}$
- generate randomly a number $r \in [0, 1]$
- if $r < p_c$ then the crossover is possible;
  - select a random position $p \in \{1, \ldots, 2n - 1\}$
  - the two chromosomes $(a_1, ..., a_p, a_{p+1}, ..., a_{2n})$ and $(b_1, ..., b_p, b_{p+1}, ..., b_{2n})$ are replaced by the two new chromosomes $(a_1, ..., a_p, b_{p+1}, ..., b_{2n})$ and $(b_1, ..., b_p, a_{p+1}, ..., a_{2n})$.
- if the crossover does not occur then the two chromosomes are put back in $P_{parents}$.

Mutation is defined as :

- For each chromosome $G \in P_{parents}$ and for each bit $b_j$ in $G$, generate a random number $r \in [0, 1]$,
- if $r < p_m$ then mutate the bit $b_j$ (i.e. flip the bit).

The population obtained after these evolution operations becomes the current population and will be the new input of the whole process described previously. This full process is repeated to generate successive populations and one has to define the number of populations to be explored. The best chromosome of each population w.r.t. the evaluation function represents the current best solution to the problem.

This methodology has been implemented in our system `GADEL` that is written in Prolog and is able to deal with every kind of Reiter's default theories. The performances of `GADEL` are described below in the section Results.

## Ant Colony Optimization

Ant Colony Optimization (ACO) metaheuristics (Dorigo, Bonabeau, & Theraulaz 2000; Corne, Dorigo, & Glover 1999) have been inspired by the observation of the collective behaviour of ants when they are seeking food. For instance, we suppose that there are many ants in a nest and that we deposit food in a place linked to the nest by two different paths $P_1$ and $P_2$, such that $P_1$ is shorter than $P_2$. At the beginning of their exploration approximatively the same number of ants will choose one path or the other. But, after few minutes, most of the ants will use the shortest path $P_1$. The emergence of this shortest preferred path is explained by the following points.

- every ant deposits a little bit of *pheromone* all along its walk
- every ant directs itself by doing a probabilistic choice biaised by the amount of pheromone that it finds on each possible path
- the pheromone evaporates

Thus, the amount of pheromone on $P1$ will increase faster than on $P2$ since in a same duration a greater number of ants take this path. And consequently, a greater number of ants will choose $P1$ since its attractivity becomes greater. And so on, by reinforcement, the amount of pheromone on $P2$ decreases and this on $P1$ increases directing almost all ants on this shortest path.

This collective behaviour based on a kind of shared memory (the pheromone) can be used for the resolution of every combinatorial problem that can be represented as the search of a certain path in a graph. For the ECP in Default Logic we propose the following encoding.

**Definition 9** *Let $(W, D)$ a default theory, its default graph is*

$$G(W, D) = (D \cup \{in, out\}, A)$$

*where in and out are two particular vertices added to the default set, and A is the arc set defined by*

$$\begin{aligned} A &= \{(in, \delta), \forall \delta \in D | W \vdash pre(\delta) \\ &\quad and\, \forall \beta \in jus(\delta) W \cup cons(\delta) \not\vdash \neg\beta\} \\ &\cup\ \{(\delta, \delta') \in D^2, \delta \neq \delta'\} \\ &\cup\ \{(\delta, out), \forall \delta \in D\} \end{aligned}$$

*In addition, each arc $(i, j) \in A$ is weighted by an artificial pheromone $\varphi_{i,j}$ that is a positive real number.*

We do not systematically put an arc from $in$ to every default in $D$, since we want to start the search by defaults that can be applied in $W$. In addition, after this initialization phase, we remove from $A$ the arcs

$$(\delta, \_)\ and\ (\_, \delta)\ if\ \exists \beta \in jus(\delta),\ W \cup cons(\delta) \vdash \neg\beta$$

and

$$(\delta, \delta')\ if\ W \cup cons(\delta) \cup cons(\delta') \vdash \bot$$

By this way we reduce the search space, because in the first case such defaults (like $\frac{:a}{\neg a}$) can never be applied, and in the second case the two defaults are incompatible together. Note that in this case it does not forbid this two defaults to appear in the same path as it is defined below. It is obvious, that many other efforts could be done to prune this graph but this could become very expensive.

**Definition 10** *Given a default theory $(W, D)$, a path $P$ from $in$ to $out$ in $G(W, D)$, the candidate generating default set associated to $P$ is $P \cap D$.*

In the sequel, we identify vertices and defaults and we indifferently use $P$ as a generating default set or as a path in the graph. The goal of ant colony is to find a path that corresponds to a true generating default set. At the beginning, the pheromone on every arc of the graph is initialized to 1 in order to give equal chance to all paths. During the process this pheromone globally evaporates and increases on arcs that are on "good" paths in order to concentrate a great number of ants on goods paths.

In order to guide each ant during its journey from $in$ to $out$ we also used a local evaluation based on the next function $loc$.

**Definition 11** *Let $P$ a path in the graph and $\delta$ a default. We say that:*

- *$\delta$ is* grounded *in $P$, if $W \cup cons(P) \vdash pre(\delta)$*
- *$\delta$ is* compatible *with $P$, if $\forall \beta \in jus(\delta) W \cup cons(P) \nvdash \neg\beta$*

*and we define*

$$
\begin{aligned}
loc(P, \delta) &= 0.9 \text{ if } \delta \text{ is grounded in } P \text{ and compatible with } P \\
&= 0.5 \text{ if } \delta \text{ is only compatible with } P \\
&= 0.2 \text{ if } \delta \text{ is only grounded in } P \\
&= 0.1 \text{ otherwise}
\end{aligned}
$$

This local function combined with the recorded pheromone leads to the definition of the attractivity of a vertex $\delta$ for an ant staying on the last vertex of a partial path $P$ between $in$ and $out$. The coefficients were chosen intuitively and can be adapted to improve the performance of the system.

**Definition 12** *Let $G(W, D) = (V, A)$ a default graph, $P$ a path from vertex $in$ to vertex $v_i$. We define $\mathcal{R}(v_i, P) = \{v_j \in V \backslash P \ s.t. \ (v_i, v_j) \in A\}$ the set of vertices reachable from $v_i$ and the attractivity of each vertex $v_j \in \mathcal{R}(v_i, P)$*

$$A(v_i, v_j, P) = \frac{\varphi_{i,j} * (loc(P, v_j))^\alpha}{\sum_{v_k \in \mathcal{R}(v_i,P)} \varphi_{i,k} * (loc(P, v_k))^\alpha}$$

*where $\alpha$ is a positive number that permits us to give more or less influence of the local evaluation*

This attractivity is the basis for the random walk of each ant from $in$ to $out$ as it is described in the next algorithm.

**Function ant_travel**
$current \leftarrow in$
$P \leftarrow \emptyset$
while $current \neq out$ do
    compute $A(current, v)$ for all $v \in \mathcal{R}(current, P)$
    choose vertex $next \in \mathcal{R}(current, P)$ with
        probability $A(current, next, P)$
    add next at the end of $P$
    $current \leftarrow next$
endwhile
return $P$

And the whole algorithm is the following.

**Procedure ACO_DL**
**input**
    $(W, D)$ a default theory
    $nbant$ the number of ants in the colony
    $maxiter$ the maximum number of iterations
    $alpha$ the coefficient for local evaluation
initialize the graph $G(W, D)$
$stop \leftarrow false$
$iter \leftarrow 1$
while not $stop$ do
    for $i = 1$ to $nbant$ do $P[i] \leftarrow ant\_travel$
    update $\varphi$ by means of the best $k$ paths
    $\forall i, j \varphi_{i,j} \leftarrow \varphi_{i,j} * 0.99$  * evaporation *\
    $solution \leftarrow eval(bestpath) = \bot$
    $stop \leftarrow (iter > maxiter) \ or \ solution$
endwhile

At each step, the best paths are determined with respect to the $eval$ function defined in Definition 6 and the update of pheromone by means of the $k^{th}$ best path $P$ is done by.

$$\varphi(i, j) \leftarrow \varphi(i, j) + 0.9^{k-1}, \ \forall arc(i, j) in P$$

Also, the chosen coefficient can be changed to improve the system.

This methodology has been implemented in our system `ANTDEL` that is written in Java and is able to deal with every kind of normal logic program that we considered as default theories. The performances of `ANTDEL` are described in the section Results.

## Local Search

Local Search is a class of powerful methods to tackle difficult optimization problems. The development of modern metaheuristics such as Tabu Search or Simulated Annealing (Michalewicz & Fogel 2000; Aarts & Lenstra 1997) has greatly increase their use and their efficiency. A general local search procedure can be defined as

**Procedure local search**
choose an initial starting point $x$ in $\mathcal{S}$
While not termination condition do
    $x \leftarrow$improve($x$)
Endwhile
return($x$)

where $\mathcal{S}$ is the search space. The sub-procedure improve($x$) returns a new point $y$ in the neighborhood of $x$ which is better than $x$ if such a point exists. Designing a local search algorithm consists in choosing the well suited notion of neighborhood together with an appropriate termination condition and the evaluation process. There exists many extensions of this basic principle : descent method, descent with random walk, metropolis, simulated annealing, tabu search ...

The evaluation of possible moves from a point to one of its neighbors will be based on the previous evaluation function. We just focus here on the basic structures : the definition of a search space and of a neighborhood. Here, the search space is the previously defined $\mathcal{CGD}$.

Concerning the moves in this search space, according to the definition of candidate extensions associated to individuals, they will be defined w.r.t. the notion of applied default. We impose that two neighbor candidate generating default sets differ only by one of their defaults. The neighborhood can be defined as a function : $\mathcal{N}: \mathcal{CGD} \rightarrow 2^{\mathcal{CGD}}$ such that $\mathcal{N}(C) = \{C' \in \mathcal{CGD} \mid C' = C \cup \{\delta\}, \delta \notin C \vee C' = C - \{\delta\}, \delta \in C\}$. For our experiments, we choose to implement a simple local search method : Descent with Random Walk. We recall this approach in our context :

**Input** : Initial individual $C$
Probability of random walk $p_{RW}$
Number of iterations Depth-LS
$Best \leftarrow C$
$Current \leftarrow C$
iter $\leftarrow 0$
While iter $\leq$ Depth-LS do
proba $\leftarrow$ random
if proba $\geq P_{RW}$
then, w.r.t. $eval$ function, choose the best
$C' \in \mathcal{N}(Current) \cup \{Current\}$
else choose randomly $C' \in \mathcal{N}(Current)$
$Current \leftarrow C'$
if $eval(Current) < eval(Best)$
then $Best \leftarrow Current$
iter $\leftarrow$ iter + 1
Endwhile
return $Best$

Note that the random walk principle is added to avoid local minima. In our system, this procedure will be performed on a given number of individuals randomly chosen in the current population.

## Results

The system GADEL is implemented in Sicstus Prolog 3.8.3. and we have evaluated its performance on a family of examples from graph theory : the Hamiltonian cycle problem for a ladder graph. Each problem $ham\_N$ ($2N$ vertices in the graph) has been generated and encoded in a default theory ($8N-3$ defaults) by means of system *tbase* as it is described in (Cholewiński *et al.* 1999) and has exactly two different extensions.

| $Hd$ | $NS$ | $T$ | $NP$ | $T1$ | $TS1$ | $PS$ |
|---|---|---|---|---|---|---|
| 0 | 23 | 260.2 | 57.6 | 4.5 | 69.5 | 14.2 |
| 2 | 23 | 251.4 | 52.7 | 4.7 | 43.3 | 7.8 |
| 4 | 26 | 269.6 | 44.4 | 6.0 | 130.7 | 20.5 |
| 6 | 22 | 260.6 | 67.8 | 3.8 | 83.6 | 19.7 |
| 8 | 27 | 141.9 | 44.7 | 3.1 | 92.0 | 27.4 |
| 10 | 27 | 129.4 | 73.3 | 1.7 | 106.9 | 59.2 |
| 12 | 26 | 127.0 | 89.5 | 1.4 | 104.5 | 72.5 |
| 14 | 16 | 115.2 | 134.2 | 0.8 | 69.2 | 76.6 |

Table 1: Influence of Hamming distance

Table 1 refers to the influence of the Hamming distance for the problem $ham\_5$ (30 runs per Hamming distance $Hd$ with parameters $p_{size} = 465$, $p_c = 0.8$, $p_m = 0.1$, $p_i = 0.9$, an initial incrementally non-conflicting grounded population, a one point crossover and a maximum number of populations equal to 200). $NS$ is the number of successful runs, $T$ the average time in seconds of a run, $NP$ the average number of populations, $T1$ the average time in seconds to do one iteration, $TS1$ the average time in seconds to do a successful run and $PS$ the average number of populations of the successful runs. It shows the importance of population diversity to increase the stability of the method (in number of iterations) and to speed up each iteration by decreasing the size of the population. It demonstrates also that a too high selective pressure ($Hd \geq 13$) strongly reduces the chances to have a successful run by decreasing too much the size of the selected population (and then the offsprings).

The next study is based on the benchmark $ham\_6$ with the parameters : $p_{size} = 210$, $p_m = 0.1$, $p_c = 0.8$, $Hd = 20$, $p_{RW} = 0.05$. The figure 1 shows the improvement due to the local search procedure w.r.t. the number of generations to be explored to get a solution. The local search is performed on 5 chromosomes at each generation. Of course, increasing the number of local search iterations increases the computation time. Based on our experiments, it seems that a local search of depth 5 provides the best results w.r.t. the ratio number of generations - time. The next figures 2 and 3

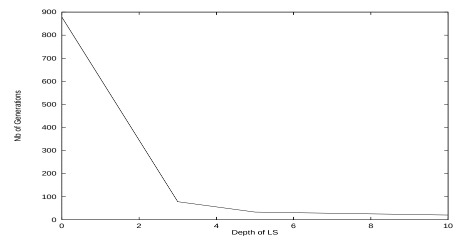


Figure 1: Generation number w.r.t LS Depth

show the influence of the number of chromosomes that are improved by the local search on the number of generations to be explored and on the computation time. These exper-

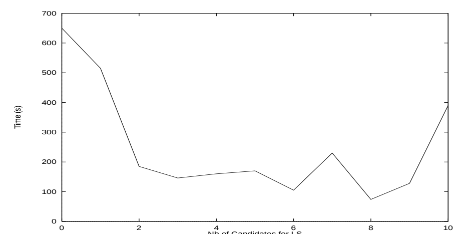


Figure 2: Computation time w.r.t. Number of improved candidates

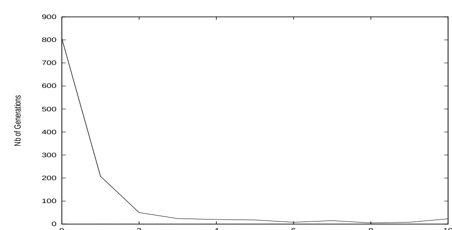


Figure 3: Generation number w.r.t. Number of improved candidates

iments show that the introduction of the local search algorithm in the genetic algorithms process clearly improves its performance. The only difficulty is then to adjust the different parameters to get the best results.

We compare GADEL and its local search improvement with DeRes (Cholewiński *et al.* 1999) because both systems accept any kind of propositional closed default theories. In table 2 the first column gives the used default theories. The second and third columns show respectively number of defaults for this theory (nd) and number of applied defaults for an extension of this theory (nad). The fourth and fifth columns give respectively average number of populations (np) and CPU time in seconds (cpu) for GADEL, sixth and seventh columns give respectively average number of populations (np) and CPU time in seconds (cpu) for GADEL+LS and finally the eighth column gives the CPU time in seconds (cpu) for DeRes to compute one extension with the full prover option.

| | | | GADEL | | GADEL+LS | | DeRes |
|---|---|---|---|---|---|---|---|
| Problem | nd | nad | np | cpu | np | cpu | cpu |
| $ham\_4$ | 29 | 8 | 22.0 | 12.2 | 4.4 | 15.2 | 11.1 |
| $ham\_5$ | 37 | 10 | 83.5 | 111.2 | 12.2 | 105.3 | 338.6 |
| $ham\_6$ | 45 | 12 | 260.1 | 609.3 | 46.3 | 458.2 | 8868.1 |

Table 2: Comparison

The system ANTEL is implemented in Java Solaris_JDK_1.2.1_04c and is able to deal with every Reiter's default theory that is equivalent to a logic program. ANTDEL is not currently able to deal with any default theory because we have not yet implemented a theorem prover inside it but there is no theoretical restriction to envisage this task. The performances related in the next table have been obtained with colonies of 100 ants and 200 iterations at most and each problem have been tested 100 times. %suc is the ratio of successfull tests over the whole number of tests. ni (respectively cpu) is the average number of iterations (respectively time in sec) of the successfull tries. np is the number of paths that are updated after each iteration and $\alpha$ is the coefficient tha goves more or less importance to the local evaluation.

| problem | $\alpha$ | np | %suc | ni | cpu |
|---|---|---|---|---|---|
| ham3x2 | 1 | 10 | 15 | 97 | 72 |
| ham3x2 | 2 | 10 | 87 | 64 | 62 |
| ham3x2 | 4 | 10 | 100 | 9 | 20 |
| ham3x2 | 6 | 10 | 98 | 5 | 12 |
| ham5x2 | 6 | 10 | 44 | 64 | 85 |
| ham5x2 | 6 | 20 | 53 | 56 | 73 |

These very few and first results show that the local evaluation has to be fixed to at least 4 if one wants to obtain acceptable results. But, this is only a first approach and, for instance, we are working on a more complex structure of pheromone in order to better exploit the notion of good path.

## Conclusion

In this paper, we have introduced various heuristics methods to compute an extension of a default theory. At this time we have implemented two systems GADEL and ANTDEL whose performances have been described.

Future works will consist in improving these systems either by including other optimization techniques or by studying the parallel aspects of our algorithms in order to include distributed computations.

## References


Aarts, E., and Lenstra, J., eds. 1997. *Local Search in Combinatorial Optimization*. John Wiley and Sons.

Bidoit, N., and Froidevaux, C. 1991. General logical databases and programs: Default logic semantics and stratification. *Information and Computation* 91(1):15–54.

Cholewiński, P.; Marek, V.; Mikitiuk, A.; and Truszczyński, M. 1999. Computing with default logic. *Artificial Intelligence* 112:105–146.

Corne, D.; Dorigo, M.; and Glover, F. 1999. *New Ideas in Optimization*. Mac Graw Hill.

Dorigo, M.; Bonabeau, E.; and Theraulaz, G. 2000. Ant algorithms and stimergy. *Future Generation Computer Systems* 16:851–871.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc of ICLP*.

Gottlob, G. 1992. Complexity results for nonmonotonic logics. *Journal of Logic and Computation* 2(3):397–425.

Holland, J. 1975. *Adaptation in Natural and Artificial Systemes*. University of Michigan Press.

Michalewicz, Z., and Fogel, D. 2000. *How to Solve It: Modern Heuristics*. Springer Verlag.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag.

Nicolas, P.; Saubion, F.; and Stéphan, I. 2000a. Combining heuristics for default logic reasoning systems. In *Proc of the* $12^{th}$ *IEEE ICTAI2000*.

Nicolas, P.; Saubion, F.; and Stéphan, I. 2000b. Gadel : a genetic algorithm to compute default logic extensions. In *Proc of ECAI*, 484–488.

Niemelä, I., and Simons, P. 1996. Efficient implementation of the well-founded and stable model semantics. In Maher, M. J., ed., *Proc of the Joint International Conference and Syposium on Logic Programming*, 289–303. MIT Press.

Niemelä, I. 1995. Towards efficient default reasoning. In Mellish, C., ed., *Proc of the IJCAI*, 312–318. Morgan Kaufmann Publishers.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13(1-2):81–132.

Risch, V. 1996. Analytic tableaux for default logics. *Journal of Applied Non-Classical Logics* 6(1):71–88.

Schaub, T. 1998. *The Automation of Reasoning with Incomplete Information: From semantic foundations to efficient computation*, volume 1409 of *LNAI*. Springer Verlag.

Schwind, C. 1990. A tableaux-based theorem prover for a decidable subset of default logic. In Stickel, M., ed., *Proc CADE*. Springer Verlag.