HEURISTICS FOR A DEFAULT LOGIC REASONING SYSTEM

PASCAL NICOLAS, FRÉDÉRIC SAUBION, and IGOR STÉPHAN
LERIA, University of Angers
2 Bd Lavoisier, F-49045 Angers Cedex 01
Pascal.Nicolas,Igor.Stephan,Frederic.Saubion@univ-angers.fr

In Artificial Intelligence, Default Logic is recognized as a powerful framework for knowledge representation when one has to deal with incomplete information. Its expressive power is suitable for non monotonic reasoning, but the counterpart is its very high level of theoretical complexity. Today, some operational systems are able to deal with real world applications. However, finding a default logic extension in a practical way is not yet possible in whole generality. This paper which is an extended version of[18] shows how heuristics such as Genetic Algorithms and Local Search techniques can be used and combined to build an automated default reasoning system. We give a general description of the required basic components and we exhibit experimental results.

Keywords: Non Monotonic Reasoning, Default Logic, Genetic Algorithms, Local Search techniques.

1. Introduction

Default Logic has been introduced by Reiter[22] 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 contrary to classical logic. Since they rely on present and absent informations, deductions are thus only plausible and their set is called an *extension*. But, due to the level of theoretical complexity of default logic ($\Sigma_2^p - complete$[9]), the computation of an extension is a great challenge. Previous works[4,20,24] 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 efficient system for general extension calculus.

In this paper, we show that heuristics issued from combinatorial optimization[1,17] can be successfully used and combined to build an efficient default reasoning system with the ability to deal with any propositional finite default theory without restriction on formulas.

Based on the principle of natural selection, genetic algorithms[16,12] have been quite successfully applied to combinatorial problems such as scheduling or transportation problems. The principle of genetic algorithms is to consider populations of individuals represented by their chromosomes. If individuals are considered as potential solutions to a given problem, applying a genetic algorithm consists in generating better and better individuals w.r.t. the problem by *selecting*, *crossing* and *mutating* them. This approach seems very useful for problems with huge search spaces and for which no tractable algorithm is available, such as our problem of default theory's extension search.

Local Search is a class of powerful methods to tackle difficult optimization problems such as Traveling Salesman Problem[15]. The development of modern metaheuristics such as Tabu Search [8] or Simulated Annealing [14] has greatly increase their use and their efficiency. The main principle of local search consists in, starting from an initial state, to incrementally improve a potential solution to a given problem. Specific control mechanisms can be introduced to escape local optima (e.g. temperature in Simulated Annealing or Tabu list in Tabu Search).

In a previous work[19] we have described a first system **GADeL** (Genetic Algorithm for Default Logic), based only on genetic algorithms, that has provided very promising results. In this work, we introduce local search techniques into genetic algorithms, as it has already be done for other combinatorial problems[11], to improve the performances of our system the new architecture of which is schematized in figure 1. The basic principle of this new system is that genetic algorithms allow

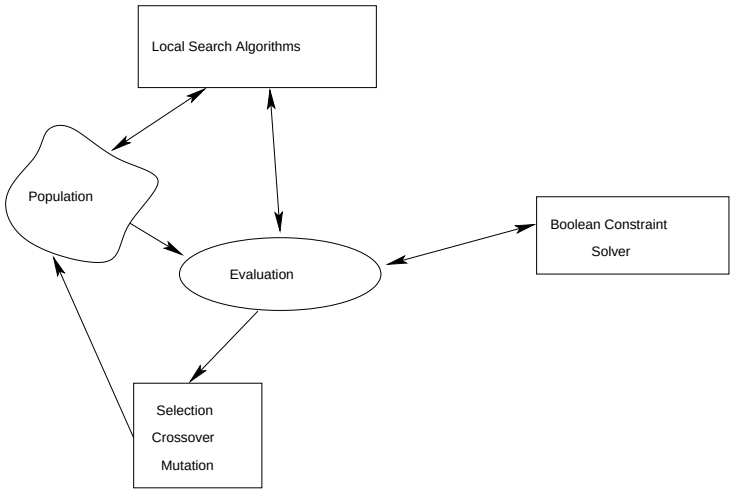

Fig. 1. System **GADeL**

us to reach quickly promising regions of our search space. Then, local search per-

forms an acute exploration of these regions. Both algorithms need an evaluation function to check the fitness of the potential solution. Since we are working on a logical problem, we need a theorem prover to achieve this evaluation. Moreover, new specific genetic operators are added to control selection and initialization and we have built a theorem prover based on a boolean constraint solver to get better results.

The paper is organized as follows : section 2 is a preliminary section where basic definitions and concepts related to default logic are presented. Section 3 provides the formal description of our system and explains how the key principles of genetic algorithms and local search are used. The section 4 provides the validation of our work by giving some experiments that show the benefits of our new approach.

2. Default Logic

In Default Logic[22] 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 (α, γ and all β_i are propositional formulas) whose meaning is "if the *prerequisite* α is proved, and if for all $i = 1, \ldots, n$ the *justification* β_i is individually consistent (in other words if nothing proves its negation) then one concludes the *consequent* γ. By sequel, we shall use the following notations. If δ is a default rule, $pre(\delta)$, $jus(\delta)$ and $cons(\delta)$ respectively denotes the prerequisite, the set of justifications and the consequent of δ. These definitions will be also extended for sets of defaults.

Given a default theory it is possible to infer a set of its plausible conclusions called an *extension* and defined by Reiter as the fix-point 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 [22] *Let (W, D) be a default theory and E a formula set. We define $E_0 = W$ and for all $k \geq 0$,*
$E_{k+1} = Th(E_k) \cup \left\{ \gamma | \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D, E_k \vdash \alpha, \text{ and } E \nvdash \neg\beta_i, \forall i = 1, \ldots, n \right\}$
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 sometimes no extension at all as we can see below.

Example 1

- $(W_1, D_1) = (\{a, b \vee c\}, \{\frac{a : \neg b}{d}, \frac{c : e}{e}, \frac{d : f}{g}\})$ *has a unique extension* $Th(W_1 \cup \{d, g\})$.
- $(W_2, D_2) = (\{a, b \vee c\}, \{\frac{a : \neg b}{\neg b}, \frac{a : \neg c}{\neg c}\})$ *has two extensions* $E = Th(W_2 \cup \{\neg b\})$ *and* $E' = Th(W_2 \cup \{\neg c\})$
- $(W_3, D_3) = (\{a\}, \{\frac{a : b}{\neg b}\})$ *has no extension.*

The non monotonicity of default logic can be illustrated as follows. Let us consider the default theory (W,D)= $(\{a\}, \{\frac{a:\neg b}{c}\})$, it has only one extension $E = Th(\{a, c\})$. We say that $\frac{a:\neg b}{c}$ has been *applied.* It is easy to check that by adding b in W we "loose" the conclusion c. The new default theory $(W', D) = (\{a, b\}, \{\frac{a:\neg b}{c}\})$ has only one extension $E' = Th(\{a, b\})$ because the default rule is now *blocked.* So adding a new axiom does not necessarily increase the set of deductions as it is always the case in classical logic.

This non monotonicity specificity is also a source of great complexity of extension calculus. In fact, in order to build an extension we could think to use the following naive greedy algorithm based on a forward chaining starting from W. At each step we apply a default δ_k whose prerequisite is satisfied and justification not contradicted by a current set S_k containing W and consequents of defaults already applied. When a fix-point S is reached we could believe that S is an extension. But, in general, this is not the case, because a default δ_k can now be invalidated by S, $\exists \beta \in jus(\delta_k), S \vdash \neg\beta$. So, from the first step to the last one we have to check a global consistency condition w.r.t. the set we try to build. This feature is clearly mentioned in the theorem 1 where 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* Δ since $E = Th(W \cup cons(\Delta))$[23].

Definition 1 *Let E be an extension of a default theory (W, D)*

$$GD(W, D, E) = \left\{ \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D, E \vdash \alpha \text{ and } E \nvdash \neg\beta_i, \forall i = 1, \ldots, n \right\}$$

is called the Generating Default Set of E.
All defaults in GD(W,D,E) are named the applied defaults.

Since computing an extension is equivalent to find its Generating Default Set, it is obvious that the search space is 2^D. Furthermore, checking if a set $\Delta \subseteq D$ is a true Generating Default Set can be done by computing $E_\Delta = Th(W \cup cons(\Delta))$ and verifying if E_Δ satisfies the theorem 1. This verification is obviously $NP-complete$ and then the complexity of the whole problem that we address in this paper is $\Sigma_2^p - complete$ as it has been established in[9].

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

Definition 2 [25] *Given a default theory (W, D), a set of defaults $\Delta \subseteq D$ is grounded if Δ can be ordered as a sequence $(\delta_1, \ldots, \delta_n)$ satisfying the property:*
$\forall i = 1, \ldots, n, W \cup cons(\{\delta_1, \ldots, \delta_{i-1}\}) \vdash pre(\delta_i)$.

Lemma 1 [25] *Every generating default set is grounded.*

Definition 3 *Given a default theory (W, D), a set of defaults $\Delta \subseteq D$ is incrementally non-conflicting if Δ can be ordered as a sequence $(\delta_1, \ldots, \delta_n)$ satisfying the property: $\forall i = 1, \ldots, n, \forall \beta \in jus(\delta_i), W \cup cons(\{\delta_1, \ldots, \delta_i\}) \nvdash \neg\beta$.*

This last definition is strongly related to the non constructive characterization of an extension as mentioned above and will be helpful for our work.

3. Description of the Method

As schematized in the introducing figure 1, the main engine of our system relies on genetic algorithms.

Genetic Algorithms[16,12] 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. Considering our extension search problem, potential solutions will be called *candidate extensions* represented by chromosomes and the purpose of our algorithm is to generate a candidate which is indeed an extension (i.e. satisfying theorem 1).

We now introduce the different components of a genetic algorithm:

1. a representation of the potential solutions : 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 $p_{size} \mid \exists N, p_{size} = \frac{N(N+1)}{2}$.

3.1. *Representation*

A representation scheme consists of the two following elements : a chromosome language $\mathcal{G}$ defined by a chosen size and an interpretation to translate chromosomes in term of possibly applied defaults, which provides the semantics of the chromosomes (also called its phenotype). In our context, for each default $\frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma}$ we encode in the chromosome the fact that the default is applied or not. Therefore, given a set of defaults $D = \{\delta_1, \cdots, \delta_n\}$ the size of the chromosome will be n and the chromosome language $\mathcal{G}$ is the regular language $(0+1)^n$ (i.e. strings of n bits). Given a chromosome $G \in \mathcal{G}$, $G|_i$ denotes the value of G at occurrence $i \in \{1..n\}$.

Our chromosomes are introduced to encode candidate extensions (i.e. potential solutions to our problem). In fact, building an extension consists in finding its *Generating Default Set* (see definition 1). Thus, the candidate extension CE(W,D,G)

associated to each chromosome can also be characterized by its candidate generating default set $CGD(D,G)$. These two sets are easily defined w.r.t. the interpretation mapping.

Definition 4 *Given a default set D, a chromosome $G \in \mathcal{G}$, the candidate generating default set associated to G is :*

$$CGD(D,G) = \{\delta_i \in D \mid G|_i = 1\}$$

Definition 5 *Given a default theory (W,D), a chromosome $G \in \mathcal{G}$, the candidate extension associated to G is :*

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

$CE(W,D,G)$ 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 2 *Let $(W,D) = (\{a\}, \{\frac{a:b}{c}, \frac{a:\neg c}{\neg b}, \frac{d:e}{f}\})$ be a default theory. We get : $CGD(100) = \{\frac{a:b}{c}\}$ and $CE(100) = Th(\{a,c\})$ which is really an extension but also $CGD(110) = \{\frac{a:b}{c}, \frac{a:\neg c}{\neg b}\}$ and $CE(110) = Th(\{a,c,\neg b\})$ which is not an extension.*

We have to mention that this representation differ from the one adopted in[18] which was more sophisticated (two positions were used for each default). Even if this first encoding was more flexible and richer, it appears, after acute experiments, that the present simpler representation provides the best performances because it drastically reduces the search space (2^N possible configurations instead of 2^{2N}).

3.2. *Generation of the Initial Population*

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 phenotype. 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 p_i a *probability of insertion* of a default in the candidate generating default set to randomly create a candidate and we randomly associate to each default δ of D a number $p_\delta \in [0,1]$. The induction definition below gives by fix-point the candidate generating default set Δ_∞.

- $\Delta_0 = \emptyset, D_0 = D,$
- $\forall j > 0, \forall \delta \in D_{j-1}, W \cup cons(\Delta_{j-1}) \vdash pre(\delta),$
  $\Delta_j = \Delta_{j-1} \cup \{\delta\}$ *if* $W \cup cons(\Delta_{j-1} \cup \{\delta\})) \nvdash \bot$ *and* $p_\delta < p_i$
  Δ_{j-1} *otherwise*
  $D_j = D_{j-1} \setminus \{\delta\}$

Then a chromosome G_∞ 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 :

$$\forall i = 1, \ldots, n, \forall \beta \in jus(\delta), W \cup cons(\Delta_{j-1} \cup \{\delta\}) \nvdash \neg\beta$$

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 that this task is too difficult for a classical algorithm so we just try to give good starting points of our searching method.

3.3. *Evaluation*

Definition 6 *Given a chromosome language $\mathcal{G}$, an evaluation function is a mapping eval: $\mathcal{G} \rightarrow \mathcal{A}$, where $\mathcal{A}$ is any set such that there exists an ordering $<$ on it (to achieve the selection process).*

Here, the evaluation function is mainly based on two criteria : the notion of generating default set (definition 1) and the notion of grounded default set (definition 2). These two aspects are rated by four intermediate functions f_0, f_1, f_2 and f_3. Obviously, the evaluation process is not only based on one chromosome but also on the underlying default theory (W, D) .

A first function f_0 rates if the candidate extension is consistent or not:

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

This check is done because an extension can never be inconsistent (except if W is inconsistent but this is out the scope of our work) and it will be a way to discard bad potential solutions.

For a default $\delta_i = \frac{\alpha_i : \beta_i^1, \ldots, \beta_i^{k_i}}{\gamma_i}$, we defined a function π described in table 1. Given the position $G|_i$ associated to the default δ_i in the chromosome, the first point is to determine w.r.t. this value if this default is supposed to be involved in the construction of the candidate extension (i.e. its conclusion has to be added to the candidate extension or not). Then, we check if this application is relevant.

A *yes* in the penalty column π means that a positive value is assigned to $\pi(G, i)$. Note that only cases 1 to 4 correspond to default considered to be applied (i.e. such that $G|_i$). The conditions $CE(G) \vdash \alpha_i$ and $\exists j, CE(G) \vdash \neg\beta_i^j$ use the classical notion of logical consequence $\vdash$ and will be checked by a theorem prover. The global evaluation function f_1 is then defined by

$$f_1(G) = \Sigma_{i=1}^{n} \pi(G, i)\ where\ n = card(D)$$

Case	$G\|_i$	$CE(G) \vdash \alpha_i$	$\exists j, CE(G) \vdash \neg\beta_i^j$	π
1	1	$true$	$false$	no
2	1	$true$	$true$	yes
3	1	$false$	$false$	yes
4	1	$false$	$truee$	yes
5	0	$true$	$false$	yes
6	0	$true$	$true$	no
7	0	$false$	$false$	no
8	0	$false$	$true$	no

Table 1. Evaluation

Justifications of the penalties:

- Case 1 : the default has been applied (because $G|_i = 1$) and it is correct with respect the candidate extension
- Cases 2,3,4 : the default should not have been applied because either its prerequisite is not satisfied ($CE(G) \nvdash \alpha_i$) or one of its justification is contradicted ($\exists j, CE(G) \vdash \neg\beta_i^j$).
- Case 5 : the default has not been applied (because $G|_i = 0$) while it should since its prerequisite is in the candidate extension and no negation of its justifications is deducible from it.
- Other cases : the default has not been applied and it is correct since either its prerequisite is not satisfied, either one of its justifications is not coherent with the candidate extension.

Another part of the evaluation is based on the fact that every generating default set is grounded. From the definition 2, we introduce a function f_2 to evaluate the "groundedness" of a candidate generating default set $CGD(G)$ as :

$$f_2(G) = card(\Gamma) \text{ where } \Gamma \text{ is the biggest grounded set } \Gamma \subseteq CGD(G)$$

and a function f_3 to definitely check this property

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

Then, we can give the definition of our global evaluation function.

Definition 7 *eval is an evaluation function of chromosomes s.t.*

$eval: \mathcal{G} \to \mathbb{N} \cup \{\top\}$

$if\ f_0(G) = 1 \quad then\ eval(G) = \top$

$else\ if\ f_1(G) = 0\ and\ f_3(G) = 0 \quad then\ eval(G) = 0$

$else\ eval(G) = f_1(G) + f_2(G)$

We can see that a chromosome G such that $eval(G) = 0$ corresponds to a default set satisfying all properties to be the generating default set of an extension.

3.4. *Theoretical validation*

We give the following theoretical result that ensures the completeness and correctness of the evaluation function and then the correctness of our computation methodology.

Theorem 2 *Let (W, D) be a default theory, G a chromosome and a candidate generating default set $\Delta = CGD(D, G)$.*
(W, D) has an extension $E = Th(W \cup cons(CGD(G)))$ if and only if $eval(G) = 0$.
By this way, our original problem of computing an extension of a default theory is now mapped into the search of a chromosome G such that $eval(G) = 0$.

Proof. $\longrightarrow$: Let $E = Th(W \cup cons(\Delta))$ be an extension of (W, D). Since E is an extension, it is consistent[22], and since Δ is its generating default set[23], it is obviously grounded. Thus, $f_0(G) = f_3(G) = 0$.

Let us suppose that $f_1(G) > 0$. Then, according to the definition of our evaluation function f_1 (see table 1), it means that there exists a default $\delta = \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D$ for which a penalty has been assigned. Let us examine the two possible cases:

- $\delta \in \Delta$: penalty can arise from cases 2, 3 or 4, but no one of them is possible since $E \vdash \alpha$ and $E \nvdash \beta_i, \forall i = 1, \ldots, n$ by definition of a generating default set
- $\delta \notin \Delta$: penalty can arise from case 5 but it is impossible since it would indicate that δ should be a generating default of E.

Thus $f_1(G) = 0$ and then $eval(G) = 0$.

$\longleftarrow$: Let $\Delta = CGD(D, G)$ such that $eval(G) = 0$ (that is $f_0(G) = f_1(G) = f_3(G) = 0$) and $E = Th(W \cup cons(\Delta))$. Since $f_3(G) = 0$, it means that Δ is grounded. So, we can order it like $\Delta = (\delta_1, \ldots, \delta_p)$ and we have

$$\forall i = 1, \ldots, p, W \cup cons(\{\delta_1, \ldots, \delta_{i-1}) \vdash pre(\delta_i)$$

that is equivalent to

$$\forall i = 1, \ldots, p, pre(\delta_i) \in Th(W \cup cons(\{\delta_1, \ldots, \delta_{i-1}))$$

from which we can build the sequence

$$\begin{aligned} E_0 &= W \\ E_{i+1} &= Th(E_i) \cup \{cons(\delta_i)\}, \forall i = 0, \ldots, p-1 \end{aligned}$$

Because of the groundedness of Δ, we obtain

$$\begin{aligned} E_0 &= W \\ E_{i+1} &= Th(E_i) \cup \{cons(\delta_i) | E_i \vdash pre(\delta_i)\}, \forall i = 0, \ldots, p-1 \end{aligned}$$

Since $f_1(G) = 0$, we can deduce : $\forall \beta \in jus(\delta_i), E \nvdash \neg\beta$ and then the sequence becomes

$$\begin{aligned} E_0 &= W \\ E_{i+1} &= Th(E_i) \cup \{cons(\delta_i) | E_i \vdash pre(\delta_i), E \nvdash \neg\beta, \forall \beta \in jus(\delta_i)\} \quad (*) \\ & \quad \forall i = 0, \ldots, p-1 \end{aligned}$$

From $f_1(G) = 0$, we can also deduce that for all other defaults $\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D \backslash \Delta$, we have either $E \nvdash \alpha$, either $\exists j, E \vdash \neg\beta_j$. So, in $(*)$ we can delete the explicit reference to i in the defaults and we can extend the sequence for all positive integers. So we have

$$\begin{aligned} E_0 &= W \\ E_{k+1} &= Th(E_k) \cup \{cons(\delta) | E_k \vdash pre(\delta), \beta \in jus(\delta), E \nvdash \neg\beta\}, \forall k > 0 \quad (**) \end{aligned}$$

Finally, let us remark that by construction E is exactly the set $\bigcup_{k=0}^{\infty} E_k$. Thus we have obtain in $(**)$ the pseudoiterative characterization of an extension given in Reiter's theorem 1, and we can conclude that E is an extension of (W, D). $\square$.

3.5. *Selection*

The purpose of the selection stage is, starting from an initial population P, to generate a selected population $P_{parents}$. According to Darwin's principle the basic idea is to choose the best chromosomes (according with the evaluation function) as parents of the next generation of chromosomes. This is achieved by the process illustrated in figure 2.

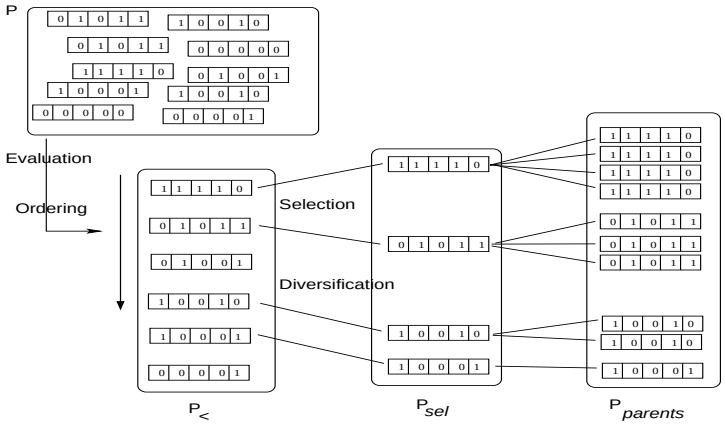

Fig. 2. Generation of the parents

First, the selection process is based on an ordering $\prec$ of the individuals, natural extension of the usual ordering of $\mathbb{N}$ extended with: $\forall n \in \mathbb{N},\ n \prec \top$. Given the population P of size p_{size}, we built an ordered population

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

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.

To keep a large diversity of selected chromosomes as parents, we introduce a Hamming distance Hd that parents must respect. Hamming distance[10] is the number of differing bits between two binary chromosomes. We define by induction 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 $card(P_i) = N$. In the figure 2, we have chosen $Hd = 2$ that is why the third chromosome in $P_<$ has not been kept in P_{sel} since it is not sufficiently different of the second one.

At last, we use the ranking selection to generate the parent population that is defined as a multiset of chromosomes such that each G_i, $1 \leq i \leq N$, in P_{sel} occurs $N - i + 1$ times in $P_{parents}$. By this way the best chromosome is duplicated N times, the second one $N - 1$ times, and so on ... This construction is required to preserve the maximum size of the population p_{size} such that $\exists N, p_{size} = \frac{N(N+1)}{2}$, i.e. $p_{size} = \Sigma_{k=1}^{N} k$.

3.6. *Crossover and Mutation*

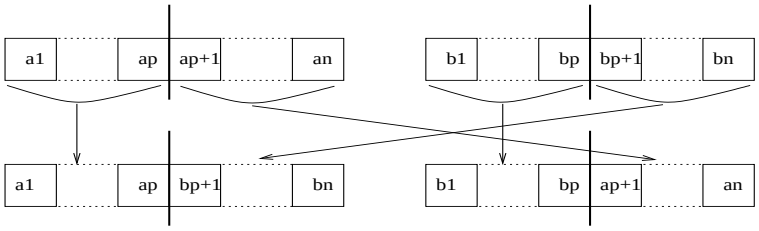

Fig. 3. Crossover

As mentioned before, genetic operators are now applied on the selected population $P_{parents}$. As illustrated in figure 3, 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 $p \in \{1, \ldots, n-1\}$
  - the two chromosomes $(a_1, ..., a_p, a_{p+1}, ..., a_n)$ and $(b_1, ..., b_p, b_{p+1}, ..., b_n)$ produce the two new chromosomes $(a_1, ..., a_p, b_{p+1}, ..., b_n)$ and $(b_1, ..., b_p, a_{p+1}, ..., a_n)$ that are put in $P\_children$
- if the crossover does not occur then the two chromosomes are simply put in $P_{children}$.

Mutation is defined as :

- For each chromosome $G \in P_{children}$ and for each bit $G|_i$ in G, generate a random number $r \in [0, 1]$,
- if $r < p_m$ then mutate the bit $G|_i$ (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 in subsections 3.2, 3.3 and 3.5. 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.

To resume, the whole process of Genetic Algorithms is illustrated in figure 4 Obviously, if a chromosome G such that $eval(G) = 0$ appears in a population the search stops since, according to theorem 2, a solution $CE(W, D, G)$ has been reached.

$$\longrightarrow P \longrightarrow P_{\prec} \longrightarrow P_{sel} \longrightarrow P_{parents} \longrightarrow P_{children}$$

Fig. 4. Main steps of the Genetic Algorithms

3.7. *Local Search*

This section now describes the basic concepts related to local search and we refer the reader to[1,17] for more details. 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 and 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 are many extensions of this basic principle : descent method, descent with random walk, 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 $\mathcal{S}$ and of a neighborhood. The search space $\mathcal{S}$ is defined as the sets of all possible chromosomes of given size.

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 extensions differ only by one of their generating defaults. As previously explained, the i^{th} default of the theory is encoded in an individual G by its position $G|_i$, therefore the neighbors of G are the chromosomes that differ from G by the fact that only one default changes of status (i.e. if it was supposed to be applied in the candidate extension related to G then it is not in the neighbor and vice versa). The neighborhood can be defined as a function :

$$\mathcal{N}: \mathcal{S} \rightarrow 2^{\mathcal{S}}$$

such that :

$$\mathcal{N}(G) = \{H \in \mathcal{G} \mid \exists i \in \{1..n\}, H|_i = 1 - G|_i \; and \; \forall j \in \{1..n\}, j \neq i, H|_j = G|_j\}$$

3.8. *Choice of Local Search Heuristics*

Our idea is not to use local search as a single search heuristic but as a way to speed up our previous evolutionary based search. Therefore, it will be only applied on the best individuals of a given generation. Such good candidates can be very close to a solution and the idea is to use a local improvements more as a repair technique than as a pure search method. We have also to consider carefully the search landscape induced by our neighborhood relation. Due to the non-monotonic aspect of Default Logic, two neighbors can have a very different evaluation. For instance, a default changing from 0 to 1 may have a consequent which is in contradiction with a lot of justifications of others defaults and therefore can turn the evaluation from

good to very bad. Due to this chaotic landscape, a pure local search strategy does not appear as a good method. This has been experimentally confirmed.

For these reasons, we decided to use a Descent with Random Walk algorithm. The purpose of the descent is to quickly improve the good candidates in few steps by slight changes in order to get the nearest local optimum. The random walk principle is added to escape this local optimum and to perform a short exploration of the neighborhood in the case of a solution could be reached only in a few search steps.

We recall the algorithm in our context :

Input :
Initial individual G
Probability of random walk p_{RW}
Number of iterations Depth-LS

$Best \leftarrow G$
$Current \leftarrow G$
iter $\leftarrow 0$
While iter $\leq$ Depth-LS do
proba $\leftarrow$ random
if proba $\geq P_{RW}$ then
choose the best $G' \in \mathcal{N}(Current) \cup \{Current\}$
w.r.t. *eval* function
else
choose randomly $G' \in \mathcal{N}(Current)$
$Current \leftarrow G'$
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.

The tuning of the random walk parameter p_{RW} will be experimentally determined in the next section. Due to the specific and limited use of local search, we have observed that its average best value is not closely related to a particular problem and is valid for a large class of tests. The important parameter is the depth of the local search which will be also determined on different problems. But as mentionned above, only a few steps of local search will be performed to repair a configuration.

3.9. *The System*

We are now able to formulate the general algorithm of our whole system by

connecting the previously defined components.

Input :

Population size p_{size}
Chromosome size G_{size}
Number of generations to be explored Max-generation
Hamming distance parameter Hd
Probability of mutation p_m
Probability of crossover p_c
Probability for random walk in local search p_{RW}
Number of iterations in local search Depth-LS
Number of individuals to improve by local search Nb-Candidate-LS

Pop $\leftarrow$ Create-initial-population(p_{size}, G_{size},Hd)
generation $\leftarrow$ 1
While generation $\leq$ Max-generation and not($\exists G \in Pop | eval(G) = 0$) do
    Parents $\leftarrow$ Evaluation-Selection(Pop,Hd)
    Children $\leftarrow$ Crossover-mutation(Parents,p_m,p_c)
    Pop-LS $\leftarrow$ Descent-RW(Children,Nb-Candidate-LS,Depth-LS,p_{RW})
    Pop $\leftarrow$ Children $\cup$ Pop-LS
    generation $\leftarrow$ generation +1
Endwhile

4. Experimental Results

Our system **GADeL** is implemented in Sicstus Prolog 3.8.6. and we have evaluated its performance on a family of examples from graph theory. The source code of this implementation is available at :
`http://www.info.univ-angers.fr/pub/stephan/Research/Download.html`
In order to have scalable and understable examples we have used the three kinds of graphs presented in figure 5.

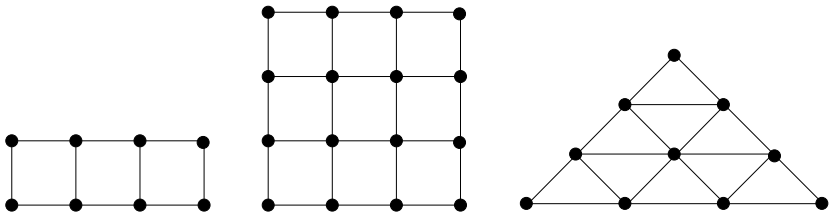

Fig. 5. Graphs for experimental studies

We have studied two kinds of problem on these graphs: the Hamilton cycle problem and the 3-coloring problem. Both of them have been generated and encoded in a default theory by means of system *tbase* as it is described in[4]. We refer to the different problems by the following conventions:

- ham_lad_N is an hamiltonian cycle problem on a ladder graph with $2N$ vertices
- ham_sim_N is an hamiltonian cycle problem on a simplex graph with N vertices on each side
- col_board_N is a 3-coloring problem on a board graph with N^2 vertices

4.1. *Efficiency of the Genetic Algorithm*

Table 2 refers to the influence of the Hamming distance for the Hamilton problem on a ladder graph with 14 vertices *ham_lad_7* (30 runs per Hamming distance 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 500).

Hamming distance Hd									
1	2	3	4	5	6	7	8	9	10
Number of successful runs									
22	27	21	30	30	30	30	26	25	18
Average CPU time (s/run)									
1784	1311	2086	1420	1281	1426	1431	2239	2158	2710
Average number of populations (/run)									
193	143	241	154	137	153	154	225	226	288
Average CPU time (s/iteration)									
9.2	9.1	9.2	9.2	9.3	9.3	9.3	10.0	9.5	9.4
Average CPU time (s/successful run)									
859	900	1403	1420	1281	1426	1431	1787	1684	1380
Average number of populations (/successful run)									
91	98	132	154	137	153	154	179	180	147

Table 2. Influence of Hamming distance for *ham_lad_7*

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 strongly reduces the chances to have a successful run by decreasing too much the size of the selected population (and then the offsprings).

Figure 6 details the results of *ham_lad_7* on 500 runs with Hamming distance equal to 5 and a maximum number of populations equal to 500 ($p_{size} = 465$,

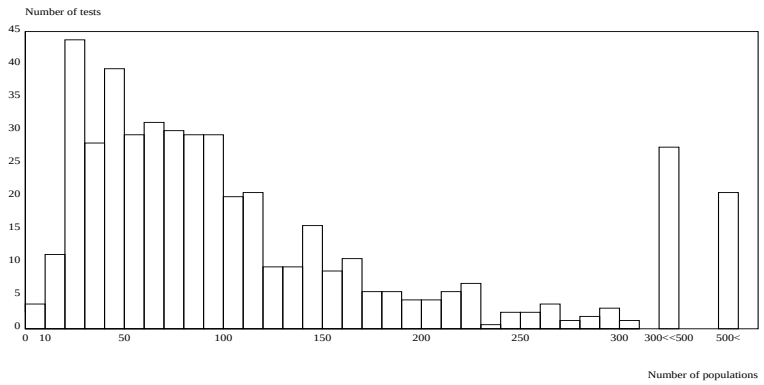

Fig. 6. *ham_lad_7* for $Hd = 5$

$p_c = 0.8$, $p_m = 0.1$, $p_i = 0.9$, an initial incrementally non-conflicting grounded population). This figure suggests to stop computation after 300 populations ($\cong 1700s$) in case of failure and to restart with a brand new one since 90% of tests end after 300 populations at most.

4.2. *Efficiency of Local Search*

This study is based on the benchmark *color_board_5* with the parameters: $p_{size} = 210$, $p_m = 0.1$, $p_c = 0.8$, $Hd = 5$, $p_{RW} = 0.05$.

The next figure 7 shows 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.

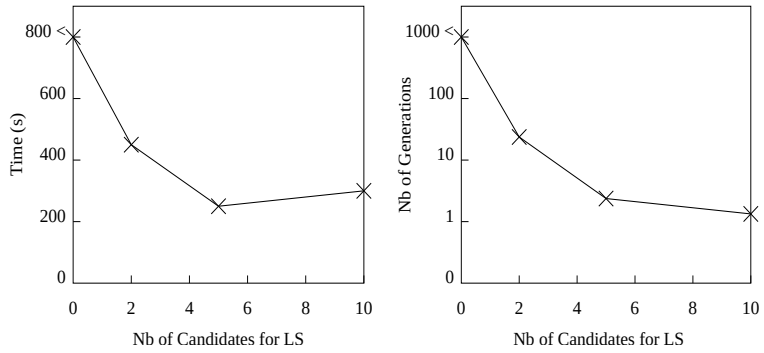

Fig. 7. Computation times and Number of generations w.r.t. Number of improved candidates

The figure 8 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 (which is the best value w. r. t. previous results) 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 10 provides the best results w.r.t. the ratio number of generations time.

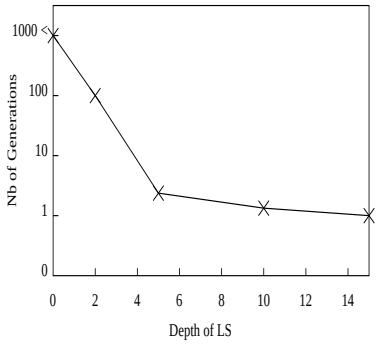

Fig. 8. Number of generations w.r.t Depth of LS

After this tuning, experiments in table 3 (nd is number of defaults for this theory, nad is number of applied defaults for an extension of this theory) confirm that the introduction of the local search algorithm in the genetic algorithms process improves its performances.

			GA		GA+LS	
Problem	nd	nad	np	cpu	np	cpu
color_board_3	63	9	0.1	8.5	0.1	9.6
color_board_4	120	16	12.8	136.0	1.0	44.9
color_board_5	195	25		> 3600	1.4	229.1
ham_lad_4	29	8	2.2	14.4	2.0	15.6
ham_lad_5	37	10	16.0	107.6	13.0	125.9
ham_lad_6	45	12	46.9	367.3	22.6	359.7
ham_lad_7	53	14	137.2	1281.1	85.4	1220.2
ham_sim_3	47	10	49.8	461.8	25.6	425.7
ham_sim_4	76	15		> 3600		> 3600

Table 3. Comparison between GA and GA+LS

One has to remark that the number of populations explored (np) clearly decreased with local search. This improvement is very important for the *color_board* benchmarks. Concerning computation time, the efficiency of local search is limited by the cost of the neighborhood evaluation, which is partly due to our Prolog implementation.

4.3. *Comparison*

			GADeL		DeRes
Problem	nd	nad	np	cpu	cpu
color_board_3	63	9	0.1	8.5	0.02
color_board_4	120	16	1.0	44.9	0.05
color_board_5	219	25	1.4	229.1	0.15
ham_lad_4	29	8	2.2	14.4	18
ham_lad_5	37	10	16.0	107.6	530
ham_lad_6	45	12	22.6	359.7	> 3600
ham_lad_7	53	14	85.4	1220.2	> 3600
ham_sim_3	47	10	25.6	425.7	> 3600
ham_sim_4	76	15		> 3600	> 3600

Table 4. Comparison between **GADeL** and DeRes

We compare **GADeL** and its local search improvement with DeRes[5] because both systems accept any kind of propositional closed default theories. In table 4 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** and the sixth column gives the CPU time in seconds (cpu) for DeRes to compute one extension.

We have to mention that problems *color_board_N* are stratified[3]. Stratification is a syntactic property of a default theory that is exploited by DeRes in order to speed up its search. This point explains why DeRes has very efficient performances on these problems. But, the stratification property is rarely satisfied in whole generality. For instance it is not satisfied any more as soon as $\exists \delta \in D / conseq(\delta) \in W$ and it is the case for *ham_lad_N* and *ham_sim_N*. For these two difficult problems, our experiments show the validity of our system. Even if the CPU times are high, the numbers of generated populations remains rather weak w.r.t. to the size of the search space that is 2^{nd} ($2^{53} \cong 9 \times 10^{15}$ for *ham_lad_7*).

5. Conclusion

Previous works have studied the automation of non monotonic reasoning, but except system Deres[5,4], all are concerned by subcases of Default Logic. [13,2] deal with non monotonic inheritance networks. [20] is about full Default Logic but without implementation. However, it has been the basis for the development of a very efficient system named Smodels[21] that is able to compute the stable models of a logic program that, again, is a subcase of Default Logic that has a lower complexity (NP-complete). Today, system DLV[7] which deals with disjunctive logic programs is certainly the most efficient system for non monotonic reasoning. So, in our

knowledge, Deres[5,4] and our system **GADeL** are the only ones that are able to deal with full Default Logic with a reasonable amount of time. The experiments that we have described in subsection 4.3 show that our approach is the better one on the general class of non stratified default theories. A future study on how to integrate the stratification property in our heuristics would be helpful to improve the performance of **GADeL**.

During past years, many variants of Default Logic have been proposed (see[24]) and all of them could be put in the scope of **GADeL** only by redefining the function *eval* (see definition 7). But, we think that it is not a good idea for all variants that are semi-monotonic since in these cases a greedy algorithm is sufficient.

In this work we have extended our system **GADeL** (first described in [19]) by adding local search heuristics and new operators to our global genetic algorithm system. By this way we have significantly improved its performances. We think that it is because genetic algorithms provides a good way to explore the whole search space while local search adds efficiency in the last steps of the search. Some new specific processes have also been added to the genetic algorithms part : a non-conflicting generation of an initial population and diversification of the individuals thanks to Hamming distance. Moreover, this system can be finely tuned to any given problem by adjusting the relative amount of genetic algorithms and local search. Even if programmed in Prolog, it provides good computation time.

As future work, efficiency could also be increased by considering a parallel treatment of genetic algorithms and local search [6].

References

[1] E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley and Sons, 1997.

[2] B. Boutsinas. On managing nonmontonic transitive relationships. In *8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 374–382. IEEE Computer Society Press, 1996.

[3] P. Cholewiński. Reasoning with stratified default theories. In *Proceedings of the Third International Conference on Logic Programming and Nonmonotonic Reasoning*, 1995.

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

[5] P. Cholewiński, V. Marek, and M. Truszczyński. Default reasoning system DeReS. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, 1996.

[6] T.G. Crainic and M. Gendreau. *Meta-Heuristics, Advances and Trends in Local Search Paradigms for Optimization*, chapter Towards an evoutionary method-cooperating multi-thread parallel tabu search hybrid. 1999.

[7] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The kr system dlv:progress report, comparisons and benchmarks. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann Publishers, 1998.

[8] F. Glover and M. Laguna. *Modern Heuristics for Combinatorial Optimization*, chapter Tabu Search. Oxford, 1993.

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

[10] R.W. Hamming. Error detecting and error correcting codes. *Bell System Techn. 29*, 1950.

[11] J.K. Hao and P. Galinier. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

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

[13] J. Horty, R. Thomason, and D. Touretzky. A skeptical theory of inheritance in nonmonotonic semantic networks. *Artificial Intelligence*, 42:311–348, 1990.

[14] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing : an experimental evaluation. *Science*, (220):671–680, 1983.

[15] S. Lin and B.W. Kernighan. An efficient heuristics for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

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

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

[18] P. Nicolas, F. Saubion, and I. Stéphan. Combining heuristics for default logic reasoning systems. In *Proceedings of the* 12^{th} *IEEE International Conference on Tools with Artificial Intelligence (ICTAI2000)*, 2000.

[19] P. Nicolas, F. Saubion, and I. Stéphan. Gadel : a genetic algorithm to compute default logic extensions. In *Proceedings of the European Conference on Artificial Intelligence*, pages 484–488, 2000.

[20] I. Niemelä. Towards efficient default reasoning. In C. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 312–318. Morgan Kaufmann Publishers, 1995.

[21] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of LPNMR*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 420–429, 1997.

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

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

[24] T. Schaub. *The Automation of Reasoning with Incomplete Information: From semantic foundations to efficient computation*, volume 1409 of *Lecture Notes in Artificial Intelligence.* Springer Verlag, 1998.

[25] C. Schwind. A tableaux-based theorem prover for a decidable subset of default logic. In M. Stickel, editor, *Proceedings of the Conference on Automated Deduction.* Springer Verlag, 1990.