On Kernels, Defaults and Even Graphs

Yannis Dimopoulos   Vangelis Magirou
Christos Papadimitriou

MPI–I–93–226   June 1993

On Kernels, Defaults and Even Graphs

Yannis Dimopoulos Vangelis Magirou
Christos Papadimitriou

MPI–I–93–226 June 1993

Author's Address

Yannis Dimopoulos
Max-Planck-Institut für Informatik,
Im Stadtwald,
66123 Saarbrücken, Germany,
yannis@mpi-sb.mpg.de

Vangelis Magirou
Athens University of Economics,
Department of Informatics,
76 Patission Str., 10434,
Athens, Greece,
vmag@isosun.ariadne-t.gr

Christos Papadimitriou
University of California at San Diego,
CS & EE Department,
La Jolla, CA 92093-0114, USA,
christos@cs.ucsd.edu

Abstract

Extensions in prerequisite-free, disjunction-free default theories have been shown to be in direct corespondence with kernels of directed graphs; hence default theories without odd cycles always have a "standard" kind of an extension. We show that, although all "standard" extensions can be enumerated explicitly, several other problems remain intractable for such theories: Telling whether a non-standard extension exists, enumerating all extensions, and finding the minimal standard extension. We also present a new graph-theoretic algorithm, based on vertex feedback sets, for enumerating all extensions of a general prerequisite-free, disjunction-free default theory (possibly with odd cycles). The algorithm empirically performs well for quite large theories.

Keywords

Default Reasoning, Logic Programming, Graph Theory, Kernels, Tractability, Algorithms.

1 Introduction

Default Logic, first introduced in [10], is a most important formalism of common-sense reasoning. A propositional *default* is a rule of the form $a : Mb_1, \ldots, Mb_n/w$ where $a, b_1, \ldots, b_n, w$ are propositions. Intuitevely such a default means that if I believe a (called the *prerequisite*) and it is consistent to believe $b_1, \ldots, b_n$ (called the *justifications*), then I should believe w (called the *consequent*). A propositional *default theory* is a pair $\Delta = (D, W)$, where D is a set of propositional defaults and W is a set of propositions.

The semantics of a default theory are formalized in terms of an important model-theoretic concept called an *extension*. Let $\Delta = (D, W)$ be a default theory. A set of propositions E is an *extension* of Δ iff it satisfies the equation $E = \cup_{i=0}^{\infty} E_i$ where $E_0 = W$ and $E_{i+1} = Th(E_i) \cup \{w \mid a : Mb_1 \wedge Mb_2 \ldots Mb_n/w, a \in E \text{ and } \neg b_j \notin E\}$.

In this paper we are interested in *propositional, disjunction-free, prerequisite-free, seminormal* defaults, in which W and the prerequisites of all defaults are empty, while the justifications and the consequent are conjuctions of literals. Seminormal is a default of the form $a : Mb \wedge c/b$. Restricting ourselves to the case of prerequisite-free theories is not a serious limitation, since in [4] it is shown that for every disjunction-free default theory there is a semantically equivalent disjunction-free, prerequisite-free theory. On the other hand seminormal defaults seem to be adequate for practical applications. The disjunction-free case is a real limitation, but a widely studied one.

In [3] a graph-theoretic approach to disjunction-free, prerequisite-free, seminormal, default theories was proposed. We can associate with each disjunction-free, prerequisite-free, seminormal, default theory D a directed graph $G(\Delta) = (N, E)$ where the nodes N are associated to the defaults and the edges depict the interaction between the default rules. In particular a edge $(n_i, n_j) \in E$ if the negation of some proposition in the consequent of default d_i occurs in the justification of d_j.

Extensions of Δ were shown in [3] to correspond to *kernels* of of a graph. A kernel of a directed graph $G = (N, E)$ is a set of nodes $K \subseteq N$ such that the nodes in K are pairwise non-adjacent, and for every node $i \in N - K$ there is some node $j \in K$ such that $(j, i) \in E$ —in other words, a kernel is an independent dominating set.

This connection between kernels and extensions of default theories was further exploited in [8], and in [9], [4] in the context of Logic Programs with negation. Thus, algorithms for finding a kernel in a graph, or even better for enumerating all kernels, have direct applications for default theories and logic programs under stable model semantics ([7]). Unfortunately, the problem of finding whether or not a graph has a kernel is known to be NP-complete [6]. However, there are certain important positive results that are obtained by the graph-theoretic connection: Directed graphs that have no odd cycles (called *even* directed graphs in this paper) always have a "standard" kind of a kernel, which includes every other node of each (even) cycle (these "standard" kernels are implicit in the proof of Richardson's Theorem, [2]). Hence, prerequisite-free, disjunction-free default theories that happen to have no odd cycles always have a certain "standard" kind of default, which can be found in linear time. In fact, *in this paper we present an algorithm which enumerates all such "standard" kernels (and thus extensions) with linear delay between two solutions output.*

Unfortunately, we also show two negative complexity results related to kernels of even graphs: Even graphs may have kernels other than the standard ones. Besides, it is NP-complete to tell whether a given even graph has such a "non-standard" kernel (and thus it is NP-complete to tell whether an even default theory has other defaults than those enumerated by our algorithm). Furthermore, suppose that you do not want to enumerate all standard kernels (or defaults) of an even graph, but want to identify a "high-quality" one. It turns out that it is NP-complete even to find the standard kernel with the minimum number of nodes —or even one with at most twice the minimum number of nodes (or any constant multiple of the minimum). Thus, we cannot find among all standard extensions the one that is the most conservative (in that it fires as few defaults as possible), or even to find an extension which is approximately the most conservative (under a most liberal definition of "approximately").

In Section 3 we present an algorithm which computes all the kernels of an arbitrary graph. The running time of the algorithm is $O(2^{|F|})$, where F is a *feedback vertex set* of the graph. A feedback vertex set of a graph is a set of nodes which, when removed from the graph, leave the graph with no cycles. This means that in the cases of sparse graphs the algorithm is expected to be fast. Experiments reported in Section 3 confirm this expectation. Interesting is also the fact that for very dense graphs the algorithm performs well, as well. These results are commented in Section 3.

2 Even Graphs

Recall that a directed graph is even if it has no odd cycles. For even graphs *Richardson's Theorem* [2] guarantees the existence of a kernel. Furthermore the proof of the theorem easily leads to an algorithm which computes a kernel of such a graph in polynomial time. An outline of the algorithm is the following:

Let $G = (N, E)$ be the graph. Initially $K = \emptyset$. Repeat the following until G is empty: First, find the strongly connected components of the graph. Since the graph is odd cycle-free, each component is a bipartite graph, say $C_i = (N_{i1}, N_{i2}, E_i)$. For each such component $C_i = (N_{i1}, N_{i2}, E_i)$ with no incoming edges, select $j \in \{1, 2\}$, set $K := K \cup N_{ij}$, and delete from G N_{i1}, N_{i2}, and all nodes v which there is an edge (u, v) with $u \in N_{ij}$. □

We say that an algorithm for generating configurations is *polynomial delay* [5] if there is only a polynomial delay between any two configurations generated. Such algorithms may behave exponentially because of the number of the exponentially many different configurations, but this is obviously unavoidable. The time complexity of a polynomial delay algorithm is $O(p(n)C)$ where n is the size of input and C the number of configurations output. In [5] a polynomial delay algorithm is presented for computing all maximal independent sets of an undirected graph, which in some sense is a problem related to the one we study. It is easy to extend the algorithm above in order to generate all *standard* kernels of an odd cycle-free graph, with *linear* delay. That is, between any two consecutive standard kernels produced, the algorithm takes time $O(|E|)$.

Proposition 1. There is a linear delay algorithm for generating all standard kernel of an odd cycle-free graph.□

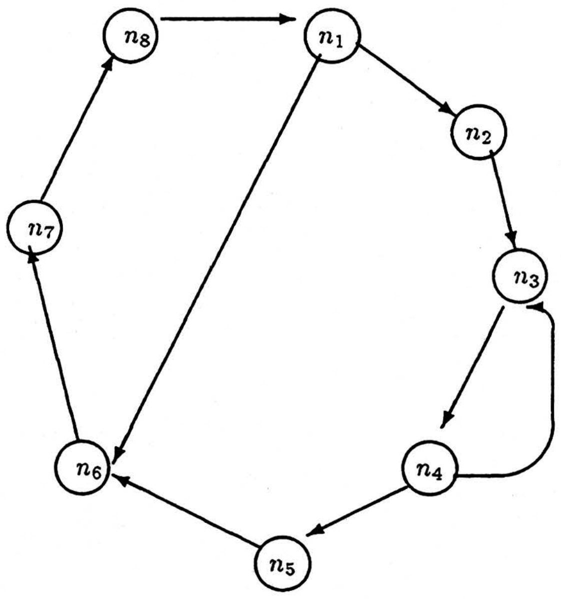

Figure 1

Figure 1 shows that even graphs may have *nonstandard* kernels. In this graph our algorithm would compute only two kernels, namely $K_1 = \{1, 3, 5, 7\}$ and $K_2 = \{2, 4, 6, 8\}$ even though an additional kernel $K' = \{1, 4, 7\}$ also exists.

The next question is whether a polynomial delay algorithm exists for enumerating all kernels of the graph, standard or not. The following result provides a negative answer:

Theorem 1: Let $G = (N, E)$ be a strongly connected even graph. Telling whether this graph has another kernel besides the two standard ones is NP-complete.

Proof: The proof is by reduction from 3-SAT. Given a formula in CNF $C = \{C_1, C_2, \ldots, C_n\}$, $C_i = C_{i1} \vee C_{i2} \vee C_{i3}$ we construct a graph like the one shown in Figure 2 as follows: For every literal x_i (and its negation) we construct a node i (i' respectively) in the set K. For every clause C_i we also put a node in K. The set L contains a node y_i for every pair of nodes i, i', and a node

C_{ij} for every literal in the clause C_i. We construct the set of edges E of the graph as follows: a) There is an edge from each node C_i in K to every node in L, b) there is an edge from every node y_i in L to every node in K, c) Two edges (x_i, y_i) and (x'_i, y_i), d) an edge (C_{ij}, C_i) for every C_i, e) A bidirectional edge (k, C_{ij}) or (k', C_{ij}), depending on whether k or k' corresponds to C_{ij}.

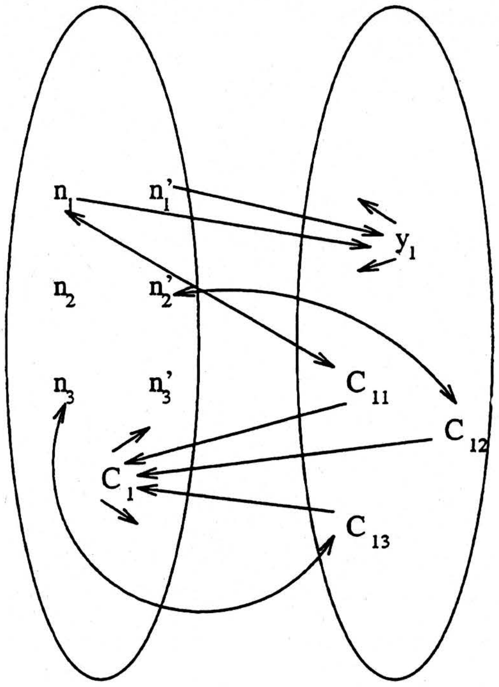

Figure 2

The graph constructed is strongly connected, does not contain odd cycles since it is bipartite, and K and L are both kernels for the graph. The problem is whether a third kernel M exists. If such a kernel exists then observe that none of the nodes C_i can belong to the kernel. The same holds for the nodes y_i. Hence, for each C_i at least one of the nodes C_{ij} must belong to the kernel M. On the other hand at least one of the i, i' must belong to M. Therefore M divides the graph into four parts: a) S, $S \subset K$, b) $\Gamma^+(S)$, $\Gamma^+(S) \subset L$, c) $L - \Gamma^+(S)$, and d) $\Gamma^+(L - \Gamma^+(S))$. Note that $K - S = \Gamma^+(L - \Gamma^+(S))$ due to the edges (C_{ij}, k), and $S \cup (L - \Gamma^+(S)) = M$.

We give the following interpretion to these sets of nodes: The set S contains all the false literals, while $\Gamma^+(S)$ contains the nodes y_i as well as all false occurences of literals in C. On the other hand $L - \Gamma^+(S)$ contains all the true occurences, while $\Gamma^+(L - \Gamma^+(S))$ contains all true literals and all satisfiable formulas. Notice that both x_i and x'_i can not be assigned the value true since the y'_is can not belong to M. It is obvious now that the true literals in this interpretation induce a satisfiable truth assignment for C.

Conversely, it is easy to verify that any satisfiable truth assignment induces a new kernel for the graph.□

An immediate consequence of Theorem 1 is the following negative result:
Corollary 1: There exists no polynomial delay algorithm for the problem of finding all kernels of a graph without odd cycles (unless P=NP).□
Indeed, if such an algorithm existed it would calculate in polynomial time a kernel different from K, L or show that such a kernel does not exist, thus determining in polynomial time the solution of any 3-SAT problem. On the other hand, Theorem 1 does not rule out the existence of a total polynomial algorithm for the problem.

Let FewP [1] be the subclass of NP consisting of problems solvable by nondeterministic machines that are guaranteed to have at most polynomially many accepting computations. The eventuality P=FewP is almost as improbable as P=NP —for example, it would imply that there are no one-way functions, and thus public-key cryptography is impossible.

Corollary 2: There exists no algorithm for enumerating all kernels of a graph without odd cycles which is polynomial in K and the number of kernels of G (unless P=FewP).□

Corollary 3: Unless P=NP, there is no polynomial-delay algorithm for enumerating all extensions of prerequisite-free, disjunction-free default theories with no odd cycles. Unless P=FewP, there is no algorithm for enumerating all extensions of prerequisite-free, disjunction-free default theories with no odd cycles which is polynomial in the number of variables and the number of extensions output.□

In [3] another negative result is proved, namely that determining whether in an even graph there is kernel including (or excluding) a given node is NP-complete. This problem is easy to solve for standard kernels.

But suppose that we are interested only in standard kernels, and we wish to find the one that is the most "conservative," in that it has the fewest nodes. In fact, suppose that you would be content with a standard kernel which has no more than $c \cdot m$ nodes, where $c > 1$ is a constant and m is the minimum number of nodes in any standard kernel. *We show that this problem is NP-hard for any $c > 1$* —even for large c's such as one million.

Theorem 2: Let $k \geq 1$ be any integer, however large. Unless P=NP there is no polynomial-time algorithm that finds a kernel of an even graph with at most $k \cdot m$ nodes, where m is the number of nodes in the smallest kernel of the graph.
Proof: First for $k = 1$. Reduction from SAT. We have a node for each literal and each clause. The edges are of the form $(x, \bar{x})$ and $(\bar{x}, x)$ for all variables x, and (λ, C) for each literal λ occurring in clause C. The graph is even. We claim that the graph has a kernel with n nodes (where n is the number of variables) iff the formula is satisfiable.

Obviously all kernels must have a node among $x, \bar{x}$ for all variables, so n is the smallest possible size of a kernel. And each kernel with n nodes must be a satisfying truth assignment (the nodes must dominate all clauses).

Now, for $k > 1$, we have $k \cdot n$ copies of each node C, and all edges (λ, C_i). Obviously, if the formula is satisfiable then the minimum kernel has size n, and if it is unsatisfiable then it has minimum kernel size at least $(k+1) \cdot n$.QED

Corollary 4: For simple even default theories, unless P=NP we cannot find in polynomial time a default which has approximately minimum number of TRUE's (or activates the approximately minimum number of defaults).QED

3 Feedback Vertex Sets and Kernels

The algorithm developed in the last section, works only with even graphs, and can only compute a subset of the kernels. It exploits the fact that given a node $j \in \Gamma^-(i)$, $i \in K$ (where K is the kernel) it is always possible to find a node $k \in K$ such that $j \in \Gamma^+(k)$. This is not possible for graphs with odd cycles and thus a different approach is needed.

The algorithm we present in this section is composed of two stages. In the first stage the graph is preprocessed in order to compute a feedback vertex set F. This set is used in the second stage of the algorithm which is an exhaustive backtracking algorithm which computes all the kernels of the graph.

3.1 The Basic Algorithm

The basic algorithm is a backtracking recursive algorithm which assigns to each one of the nodes in the graph one of the values 0, 1, -1, -2 (we denote by u_i the value assigned to the node i). These values which may change while the algorithm proceeds, have the following meaning:

- If $u_i = 0$ then the algorithm has not yet decided about the value of node i.
- If $u_i = 1$ then the node i must belong to the kernel.
- If $u_i = -1$ then the node i must not belong to the kernel.
- If $u_i = -2$ then the node i is not to belong in the kernel, but there is no node j, $(j, i) \in E$ which has been assinged the value 1 so far.

In the sequel we give a general description of the operation of the algorithm. It starts by assigning to all nodes the value 0. In each step the algorithm assigns to a node i with $u_i = 0$, the value 1. The value $u_k = -1$ is assigned to all the nodes $k \in \Gamma^+(i)$, while the value the value $u_k = -2$ is given to all nodes $k \in \Gamma^-(i)$. After assigning these values, the algorithm enters a second phase in which the feasibility of the u_i's values is checked. If this holds some of the values 0 or -2 may be changed to 1 or -1 respectively.

A value assignment may not be feasible because it is possible for a node i to have a value $u_i = -2$, but for each $k \in \Gamma^-(i)$ the values satisfy $u_k < 0$. This case is obviously contradictory and the algorithm has to backtrack and assign the value -2 to the last node assigned the value 1. Each time a node is assigned the value 1 or -2 an attempt is made at some of the nodes to change their values from 0 or -2 to 1, -1, or -2 according to the following criteria:

- If there is a node i, $u_i = -2$ and for all $k \in \Gamma^-(i)$ it holds that $u_k < 0$ except for a single node $j \in \Gamma^-(i)$ for which $u_j = 0$, then assign $u_j = 1$.
- If there is a node i with $u_i = 0$ and for all nodes $k \in \Gamma^-(i)$, $u_k < 0$ holds, then set $u_i = 1$.
- If some node i is assigned the value 1 for one of the previous reasons then
  1. All nodes $k \in \Gamma^+(i)$ are assigned the value $u_k = -1$
  2. All nodes $k \in \Gamma^-(i)$ having the value 0 will get the value -2.

The above procedure iterates until there is no node the value of which can be changed or a contradiction is created. Then the algorithm is recursively called with the new partial value assignment.

If a point is reached where there is no node having the value 0 then the total value assignment is checked in order to be determined whether this assignment is kernel or not. In both cases the algorithm backtracts in order to exhaustively compute all the kernels of the graph. Backtracking means the assignment of the value -2 to the node which was assigned the value 1 last.

3.2 The Use of Feedback Vertex Set

The basic algorithm presented in the previous section does not possess any criterion for choosing the nodes which will be assigned the value 1 in each recursive call. An obvious selection criterion would be to choose the node with the highest outgoing degree. In this case the nodes which have no definite value in the next step will be the as few as possible. But the worst case time of the algorithm is still $O(2^{|N|})$ where $| N |$ is the number of the nodes of the graph. In this section we present an algorithm which achivies a speed up of the computations, especially in the case of sparse graphs where the basic algorithm of the previous section is remarkably inefficient.

The basic idea stems from the fact that, although for acyclic graphs the problem of finding a kernel is polynomial, it becomes exponential when cycles are allowed. We can therefore conclude that the culprit for the computational burden is the presence of cycles. If all the nodes which create cycles are removed then the remaining acyclic graph has exactly one kernel which can be computed in polynomial time.

Given a graph $G = (N, E)$, a set of nodes $F \subseteq N$ such that the graph $G' = (N - F, E')$ is acyclic is called feedback vertex set. A value assignment to the nodes of F can lead to a kernel or not, something that can be verified in polynomial time. The time complexity of the algorithm now becomes $O(2^{|F|})$ times the time needed to determine a kernel of an acyclic graph. Since the most important parameter of the performance of the algorithm is the size of the graph's feedback vertex set it is reasonable to look for the smallest such set. However it has been proved that the corresponding decision problem is NP-complete ([6]). This implies that the attempt to determine the feedback vertex set with the minimal cardinality will increase exponentially the complexity of the whole problem. Hence we use a heuristic which computes a hopefully small feedback vertex set. The algorithm which we use is polynomial but there is no guarantee that the feedback set computed is minimal. Our method determines first the strongly connected components of the graph, $S_1, S_2, \ldots S_n$. For each of them a feedback vertex set is computed, by a depth search which marks the nodes of the components which cause circularity. The union of all these sets is the feedback vertex set of the graph.

3.3 Computational experience

In this section we give a high-level description of the basic parts of the algorithm. Following this, we summarize the computational experience with the algorithm.

Description of the Algorithm

```
Procedure Find-Kernels ($G, V, F$);
begin
   Choose a node $i \in F$, $u_i = 0$.
   If such a node exists then
      begin
         Assign $u_i = 1$ and call Check-and-Expand($G, V', cont$);
         if cont then call Find-Kernels($G, V', F$);
         Backtrack, assign $u_i = -2$ and call Check-and-Expand($G, V', cont$);
         if cont then call Find-Kernels($G, V', F$);
      end
      else if Leads-to-Kernel($G, V'$) then output $V'$;
end.
```

```
   Main Algorithm
begin
Find the strongly connected components of the algorithm $S_1, S_2, \ldots, S_n$;
(*These sets are ordered according to their level (depth) in the components tree*)
foreach $S_i$ do find a feedback vertex set $F_i$;
```

```
Compute the union of $F_i$, $F$ preserving their height-out degree ordering;
foreach $i$ do $u_i = 0$;
Call Find-Kernels($G, V, F$);
end
```

The procedure *Check-and-Expand* is responsible for checking the consistency of value assignments (assigning the appropriate value to the boolean variable *cont*) and expanding those assignements as described in section 3.1. The procedure Leads-to-Kernel simply checks whether a value assignment is a kernel.

In Table 1 we give some results of execution of the algorithm. Rows correspond to number of nodes, while columns to graph densities.

	5%	10%	15%	20%	25%	30%	35%	40%	45%
50	0.24	0.61	0.95	1.28	1.37	1.34	1.31	1.34	1.34
60	0.49	1.49	2.37	3.08	3.14	3.11	2.91	2.69	2.51
70	0.96	3.0	5.2	6.2	5.9	5.4	4.8	4.5	3.9
80	2.17	7.1	12.6	14.1	13.8	11.0	9.7	7.9	7.1
90	3.5	16.0	29.3	30.0	24.8	20.3	16.6	13.8	11.8
100	6.6	47.8	69.0	63.0	50.1	41.8	29.5	22.9	18.9

CPU time in seconds.

Table 1

The algorithm was implemented in Pascal and the program executed in a VAX-8810 under VMS Ver 5.4.2 operating system. The time given is in seconds and each of them is the average of the C.P.U. time of the execution of 10 different instances.

As it was expected the algorith is fast for sparse graphs. The small feedback vertex set of those graphs leads to the efficient computation of their kernels. Recall that the edges between nodes correspond to interaction between rules. We would expect that in most of the cases emerging from default theories or logic programs, the graph will be sparse. Under this assumption the algorithm becomes particularly useful for practical applications. On the other hand intersting enough is the fact that the algorithm is also efficient for very dense graphs (density $\geq 50\%$). This can be explained by the fact that choosing the nodes with the highest degree in each step, radically reduces the size of the graphs to be considered in the next steps. Furthermore our experience has shown that dense graphs have more kernels than sparse graphs. This is because in dense graphs a small number of independent nodes with a high out-degree can dominate the rest of the nodes. The algorithm performs worst in graphs with densities between 15% and 20%. In this case on the one hand the feedback set is large enough, while on the other hand the degree of the nodes is not large enough to considerably reduce the size of the problem.

This program differs from the one we described in that the procedure Check-and-Expand is called in every even level of the search tree. The results have clearly shown that in this way, better running times are accomplished. This is due to the fact that even though such a test reduces the search space it is, overall, time consuming. Our experimentation with the algorithm showed that calling the procedure every 2 or 3 levels gives the best running time.

Another sound, incomplete but efficient procedure for default theories could be the outcome of the combination of the two algorithms presented in this paper. Namely given the graph $G(\Delta)$ of a default theory Δ instead of computing a feedback vertex set, compute a set of nodes S such that the graph $G'(\Delta)$ obtained after removing this set, is odd-loop-free. Then apply a procedure similar to the one described above by first assigning values to the nodes of S and then computing, given the assignment of the nodes in S, the kernels of $G'(\Delta)$. Obviously the procedure can compute only a subset of the kernels of $G(\Delta)$ but yet the time complexity is polynomial in size of $G(\Delta)$ and the number of kernels of $G'(\Delta)$.

4 Conclusions

In this paper we presented complexity results showing that the problem of computing all the kernels of a graph without odd cycles is intractable. Nevertheless we presented a polynomial delay algorithm which computes a, hopefully large, subset of the kernels. This algorithm may serve as a sound, incomplete but tractable procedure for reasoning in default logic. We also presented an algorithm which produces all the kernels of an arbitrary graph and its complexity is exponential in the size of a feedback vertex set. This algorithm performs particurarly well in the case of sparse graphs.

The results presented here are applicable in wide range of nonmonotonic formalisms such as default logic, autoepistemic logic, logic programming and TMS. Graph theory allows us to tackle the computational problems of those formalization within a unified framework.

References

[1] E. Allender, The complexity of sparse sets in P, in *Proc. Symp. on Structure in Complexity Theory*, Lecture Notes in CS 223, pp. 1-11, 1986.

[2] C. Berge, *Graphs and Hypergraphs* (North Holland, 1973).

[3] Y. Dimopoulos and V. Magirou, A graph theoretic approach to Default Logic, *Information & Computation*, to appear.

[4] Y. Dimopoulos and A. Torres, Graph Theoretical Structures in Logic Programs and Default Theories, in preparation, 1993.

[5] D. Johnson, C. Papadimitriou and M. Yannakakis, On generating all Maximal Independent Sets, *Information Processing Letters*, Vol. 27, 1988, pp. 119-123.

[6] M. Garey and D. Johnson, *Computers and Intractability* (Freeman, 1979).

[7] M. Gelfond and V. Lifschitz, The Stable Model semantics for Logic Programming, in: *Logic Programming: Proceedings of the 5th International Conference and Symposium*, eds. R. Kowalski and K. Bowen, 1988, pp. 1070-1080.

[8] C. Papadimitriou and M. Sideri, On finding Extensions of Default Theories, in: *Proc. of the International Conference on Database Theory*, eds. J. Biskup and R. Hull, 1992, Lecture Notes in CS 646.

[9] Papadimitriou C., Yannakakis M., "Tie-Breaking Semantics and Structural Totality", *Symposium on the Principles of Database Systems*, 1992.

[10] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence*, 13, 1980.