Evolutionary Computing for the Satisfiability Problem

Jin-Kao Hao, Frédéric Lardeux and Frédéric Saubion

LERIA, Université d'Angers
2 Bd Lavoisier, F-49045 Angers Cedex 01
`Jin-Kao.Hao@univ-angers.fr`
`Frederic.Lardeux@univ-angers.fr`
`Frederic.Saubion@univ-angers.fr`

Abstract. This paper presents GASAT, a hybrid evolutionary algorithm for the satisfiability problem (SAT). A specific crossover operator generates new solutions, that are improved by a tabu search procedure. The performance of GASAT is assessed using a set of well-known benchmarks. Comparisons with state-of-the-art SAT algorithms show that GASAT gives very competitive results. These experiments also allow us to introduce a new SAT benchmark from a coloring problem.

1 Introduction

The satisfiability problem (SAT) [9] consists in finding a truth assignment that satisfies a well-formed Boolean expression E. SAT is one of the six basic core NP-complete problems and has many applications such as VLSI test and verification, consistency maintenance, fault diagnosis, planning ... SAT is originally stated as a decision problem but one may take an interest in other related SAT problems:

– model-finding: to find satisfying truth assignments
– MAX-SAT: to find an assignment which satisfies the maximum number of clauses
– model-counting: to find the number of all the satisfying truth assignments.

During the last decade, several improved solution algorithms have been developed and important progress has been achieved. These algorithms have enlarged considerably our capacity of solving large SAT instances. Recent international challenges organized these years [16, 24] continue to boost the worldwide research on SAT.

These algorithms can be divided into two main classes. A complete algorithm is designed to solve the initial decision problem while an incomplete algorithm aims at finding satisfying assignments (model-finding). The most powerful complete algorithms are based on the Davis-Putnam-Loveland procedure [3]. They differ essentially by the underlying heuristic used for the branching rule [5, 18, 26]. Specific techniques such as symmetry-breaking, backbone detecting or equivalence elimination are also used to reinforce these algorithms [1, 17, 6].

Existing incomplete algorithms for SAT are mainly based on local search [25, 15, 20] and evolutionary algorithms [4, 13, 8, 7, 11]. The very simple hill-climber GSAT [23] and its powerful variant Walksat [22] are famous examples of incomplete algorithms. Though incomplete algorithms are little helpful for unsatisfiable instances, they represent the unique approach for finding models of very large instances. At this time, Unitwalk [14] appears as the best incomplete algorithm. A current trend for designing more powerful algorithms consists in combining in some way the best elements from different approaches, leading to hybrid algorithms [21].

In this paper, we present GASAT, a new hybrid algorithm embedding a tabu search procedure into the evolutionary framework. At a very high level, GASAT shares some similarities with the hybrid algorithm proposed in [8]. GASAT distinguishes itself by the introduction of original and specialized crossover operators, a powerful Tabu Search (TS) algorithm and the interaction between these operators.

In the following sections, we introduce the GASAT algorithm as well as its main components. The performance of GASAT is then assessed on a large range of well-known SAT instances and compared with some best known incomplete algorithms. In this section, we also propose a new benchmark based on a coloration problem. We show thus GASAT competes very favorably with these state-of-the-art algorithms. Future extensions and improvements are discussed in the last section.

2 Basic Components

In this section, we define some basic notions and notations related to evolutionary computation, local search and propositional logic.

2.1 The SAT Problem

An instance of the SAT problem is defined by a set of boolean variables (also called atoms) $\mathcal{X} = \{x_1, ..., x_n\}$ and a boolean formula $\phi: \{0,1\}^n \rightarrow \{0,1\}$. A literal is a variable (also called atom) or its negation. A clause is a disjunction of literals. The formula ϕ is in conjunctive normal form (CNF) if it is a conjunction of clauses. A (truth) assignment is a function $v: \mathcal{X} \rightarrow \{0,1\}$. The formula is said to be satisfiable if there exists an assignment satisfying ϕ and unsatisfiable otherwise. In this paper, ϕ is supposed to be in CNF.

2.2 Search Sapce, Fitness and Selection

Our search method relies on the management of a population of configurations representing potential solutions. Therefore, we present now the basic materials for this population.

Representation The most obvious way to represent an individual for a SAT instance with n variables is a string of n bits where each variable is associated to one bit. Other representation schemes are discussed in [11]. Therefore the search space is the set $\mathcal{S} = \{0,1\}^n$ (i.e. all the possible strings of n bits) and an individual X obviously corresponds to an assignment. $X|i$ denotes the the truth value of the i^{th} atom and $X[i \leftarrow \alpha]$ denotes an individual X where the i^{th} atom has been set to the value α. Given an individual X and a clause c, we use $sat(X,c)$ to denote the fact that the assignment associated to X satisfies the clause c.

Fitness Function Given a formula ϕ and an individual X, the fitness of X is defined to be the number of clauses which are not satisfied by X:
$eval{:}\, \mathcal{S} \rightarrow I\!N, eval(X) = card(\{c|\neg sat(X,c) \wedge c \in \phi\})$
where $card$ denotes as usual the cardinality of a set. This fitness function will be used in the selection process and induce an order $>_{eval}$ on the population. The smallest value of this function equals 0 and an individual having this fitness value corresponds to a solution.

Selection Process The selection operator is a function which takes as input a given population and extracts some individuals according to a selection criterion. These selected individuals are the chosen parents for the evolution process and will evolve by crossover operations. To insure an efficient search, it is necessary to keep some diversity in the population. Actually, if the selected parents are too similar, some region of the search space $\mathcal{S}$ will not be explored. The diversity of the selected population is achieved by introducing the notion of hamming distance ham [12] between strings of bits. This distance corresponds simply to the number of different bits between two strings. Therefore we define the function $select{:}\, 2^{\mathcal{S}} \times I\!N \times I\!N \rightarrow 2^{\mathcal{S}}$ such that $select(P,n,d)$ is the set of the n best X in P according to $eval$ and $\forall X, Y \in select(P,n,d), ham(X,Y) \geq d$. For the sake of simplicity, the parameter d will be automatically adjusted and will not appear in the selection function.

3 The Hybrid Genetic Algorithm: GASAT

3.1 The GASAT Algorithm

The final algorithm is obtained by combining a crossover stage with a local search improvement. Given an initial population, the first step consists in selecting its best elements (i.e. assignments) w.r.t. the function $eval$. Then, crossovers are performed on this selected population. Each child is individually improved using a tabu search. If the child is better than the assignments already in the population then it is added to the current population and the whole process goes on. The general algorithm is described in figure 1.

Data: a set of CNF clauses $\mathcal{F}$, $Maxflip$,$Maxtries$

```
Result: a satisfying truth assignment if found

Begin
CreatePopulation(P)
$tries \leftarrow 0$
While no $X \in P$ satisfies $\phi$ and $tries < Maxtries$
/* Selection */
    $P' \leftarrow Select(P, n)$
    Choose $X, Y \in P'$
/* Crossover */
    $Z \leftarrow crossover(X, Y)$
/* TS improvement */
    $Z \leftarrow Tabu(Z, Maxflip)$
/* Insertion condition of the child */
    If $\forall X \in P, Z >_{eval} X$ then replace the oldest $X$ in $P$ by $Z$
    $tries \leftarrow tries + 1$
EndWhile
If there exists $X \in P$ satisfying $\phi$
    then return the corresponding assignment
    else return the best assignement found
End
```

Fig1 : GASAT: general structure

This algorithm can be adjusted by changing the selection function, the crossover operator or the local search method but also by modifying the insertion condition. A variant of the condition used in this algorithm could be to insert the child whatever its evaluation is w.r.t. the whole population. Such a condition would bring more diversity but could also disturb the general convergence of the search.

3.2 Basic Stages

Crossover Operator A crossover or recombination operator has to take into account as much as possible the semantics of the individuals in order to control the general search process and to obtain an efficient algorithm. In the SAT problem, the variables are the atoms and a constraint structure is induced by the clauses. Therefore, while the representation focuses on the atoms, an efficient crossover should take into account the whole constraint structure.

We first define a function $flip\colon \{0,1\} \rightarrow \{0,1\}$ such that $flip(x) = 1-x$. This function induced a neighborhood relation in $\mathcal{S}$ which will be used in the tabu search mechanism when the algorithm changes from a configuration to another one by flipping one of its bits. Then, we define a function $imp\colon \mathcal{S} \times I\!N \rightarrow I\!N$ such that $imp(X|i) = card(\{c \mid sat(X[i \leftarrow flip(X|i)], c) \wedge \neg sat(X, c)\}) - card(\{c \mid \neg sat(X[i \leftarrow flip(X|i)], c) \wedge sat(X, c)\})$. This function computes the improvement obtained by the flip of the i^{th} component of X and was previously introduced in

GSAT and Walksat [23, 22]. It corresponds to the gain of the solution according to the function *eval* (i.e. the number of false clauses which become true by flipping the atom i minus the number of satisfied clauses which become false). Remark that if this number is negative then the number of false clauses would increase if the flip is performed. This function induces a natural order $>_{imp}$ on the atoms which is extended to the assignments ($X >_{imp} Y$ iff there exists a position i such that $\forall j, X|i >_{imp} Y|j$). This crossover operator produces a child Z from two parents X and Y.

Definition 1. Crossover
For each clause c such that $\neg sat(X,c) \wedge \neg sat(Y,c)$ *and for all positions i such that the variable* x_i *appears in c, we compute* $\sigma_i = imp(X|i) + imp(Y|i)$ *and we set* $Z|k = flip(X|k)$[1]*where k is the position such that* σ_k *is maximum. For all the clauses c such that* $sat(X,c) \wedge sat(Y,c)$ *and such that* $X|i = Y|i = 1$ *(resp.* $X|i = Y|i = 0$*) if the atom* x_i *appears positively (resp. negatively) in c, we set* $Z|i = 1$ *(resp.* $Z|i = 0$*). The positions in Z which have received no value by the previous operations are randomly valued.*

Example The following simple example illustrates the crossover. The problem has five variables, $\{x_1, x_2, x_3, x_4, x_5\}$ and seven clauses.

$(x_1 \vee x_3 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_5 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee x_4)$

Let X and Y be two individuals with two false clauses each:

	x_1	x_2	x_3	x_4	x_5
X	1	1	0	0	1
Y	0	1	0	0	1

1) The second and the last clause are false for X and Y. So we compute σ for all the x_i.
For the second clause : $\sigma_2 = 3$, $\sigma_3 = 4$, $\sigma_5 = 2$
For the last clause : $\sigma_2 = 3$, $\sigma_4 = 4$

	x_1	x_2	x_3	x_4	x_5
Z			1	1	

2) With the true clauses for X and Y, x_2 and x_5 can be valued.

	x_1	x_2	x_3	x_4	x_5
Z		1	1	1	1

3) x_1 is randomly valued.

	x_1	x_2	x_3	x_4	x_5
Z	0	1	1	1	1

The main idea of this operator is to keep the structure of the clauses satisfaction induced by both parents by repairing false clauses and maintaining true ones.

[1] One should remark that, if a clause is false for both parents, then all the variables appearing in this clause have necessarily the same value in both parents. This comes from the fact that a clause can be false only if all of its literals are false.

At this step, we have defined the evolutionary part of our algorithm which insures the general search with enough diversity to reach some promising regions of $\mathcal{S}$. At this time, a tabu search process will intensify the search process to fully exploit these regions by locally improving the configurations in order to get a solution if possible. Moreover, TS can also explore the search space alone as a diversification process thanks to the tabu list.

Tabu Search Procedure TS is an advanced local search method using a memory to avoid local optima [10]. The principle is quite simple: it acts as a descent (at each iteration, it makes the best move), but once visited, a configuration is made tabu, that is, the algorithm is not allowed to visit it again for a given number of iterations. Since the memorization of the last configurations could be costly, a common variation consists in making tabu only the moves. Now, a move is done if it is the best one and if it is not tabu. Once executed, the move is included in the tabu list, which acts as a fixed size FIFO queue, and is removed from this list after λ iterations. To consider improvement of the best-to-date solution, the tabu constraint is disabled in the case of moves leading to a better solution than the best-to-date. TS has already been experimented for SAT problem [20].

In our context, it is clear that the moves are possible flips of the value of a given assignment and that the tabu list will be a list of flip index already performed. Here the initial configuration will be given as entry to the TS and not generated randomly.

In order to simplify the tuning of our general algorithm and to guarantee a stable behavior, we propose an automatic adjustment of the tabu list length λ based on some experimental results. Therefore, in the remaining of the paper, we will simplify our TS function as $TS(X, Maxflip)$.

4 Experimental Results

In this section, we evaluate the performance of GASAT on several classes of satisfiable and unsatisfiable benchmark instances and compare it with Walksat [22], one of the well-known incomplete algorithms for SAT, and with UnitWalk [14] (combination of local search and unit propagation), the best up-to-now incomplete randomized solver presented to the SAT2002 competition [24].

4.1 Experimental Conditions

Due to the incomplete and non-deterministic nature of GASAT, Walksat and UnitWalk, each algorithm runs 20 times on each benchmark. This number of runs depends on the time spent by each algorithm.

For the three solvers, we limit the execution time to two hours for each run. We impose a maximum of 10^7 flips for Walksat and GASAT (Walksat with maxtries=10, as suggested in the literature, and maxflip=10^6 and GASAT with

10^3 crossovers, 10^4 flips for each TS step and a tabu list fixed to be 40% of the length of the individuals). UnitWalk runs with the default parameters. Therefore, we can consider that we allow the same power to each of the algorithm in order to get a fair evaluation.

4.2 Benchmark Instances and Evaluation Criterions

Two classes of instances are used : Structured and Random instances.

- Structured instances: `aim-100-1_6-yes1-4, aim-100-2_0-yes1-3` (random instances with only one solution), `mat25.shuffled, mat26.shuffled` (multiplication of two $n \times n$ matrices using m products [19]), `color-15-4, color-22-5` (chessboard coloring instances) `g125.18, g250.29` (graph coloring instances) and `dp10u09.suffled, dp11u10.suffled` (instances generated from bounded model checking from the well known dining philosophers example).
- Random instances: `glassy-v399-s1069116088, glassy-v450-s325799114` (random instances presented to SAT2002 Competition), `f1000, f2000` (DIMACS random instances).

The chessboard coloring benchmark `color` is new. It corresponds to the SAT encoding of a problem studied in [2] and that we have generalized. The purpose is to color a chessboard with k colors such that the four corners of every rectangle included in the board are not of the same color.

Two criterions are used to evaluate GASAT and compare it with Walksat and UnitWalk. The first one is the success rate (%) which is the number of successful runs divided by the total number of runs. This criterion is the most important one since it corresponds to the search power of an algorithm. The second criterion is the average running time in second (sec.) for successful runs on a Sun Fire 880 (4 CPU UltraSPARC III 750 Mhz, 8 Go de RAM)[2].

4.3 Comparative Results on Structured instances

Tables 1 and 2 show respectively the results of GASAT, Walksat and UnitWalk on the chosen structured and random instances.

From these tables, one observes first no clear dominance of one algorithm over the other ones. However, the results show that GASAT performs globally better on the structured instances than random ones. This seems not surprising given that its crossover operator relies on structural information. One observes a particular good performance of GASAT for the four (large) coloring instances compared with Walksat and UnitWalk. At the same time, one observes for random instances that, although GASAT gives competitive results for the glassy instances, it performs less well on the f1000 and f2000.

[2] The version of Walksat is v39 and UnitWalk's is 0.98. GASAT is implemented in C and uses some functions of Walksat.

Benchmarks				GASAT		Walksat		UnitWalk	
instances	var	cls	sat	%	sec.	%	sec.	%	sec.
aim-100-1_6-yes1-4	100	160	Y	10	84.53	(*1 clause*)		100	0.006
aim-100-2_0-yes1-3	100	200	Y	100	20.86	(*1 clause*)		100	0.019
mat25.shuffled	588	1968	N	(*3 clauses*)		(*3 clauses*)		(*8 clauses*)	
mat26.shuffled	744	2464	N	(*2 clauses*)		(*2 clauses*)		(*8 clauses*)	
color-15-4	900	45675	Y	100	479.248	(*7 clauses*)		(*16 clauses*)	
color-22-5	2420	272129	?	(*5 clauses*)		(*41 clauses*)		(*51 clauses*)	
g125.18	2250	70163	Y	100	378.660	(*2 clauses*)		(*19 clauses*)	
g250.29	7250	454622	?	(*8 clauses*)		(*34 clauses*)		(*57 clauses*)	
dp10u09.suffled	7611	20770	N	(*39 clauses*)		(*2 clauses*)		(*22 clauses*)	
dp11u10.suffled	9197	25271	N	(*56 clauses*)		(*3 clauses*)		(*20 clauses*)	

Table 1. Structured instances (if no assignment is found then the best number of false clauses is written between parentheses)

Benchmarks				GASAT		Walksat		UnitWalk	
instances	var	cls	sat	%	sec.	%	sec.	%	sec.
glassy-v399-s069116088	399	1862	Y	(*5 clauses*)		(*5 clauses*)		(*17 clauses*)	
glassy-v450-s325799114	450	2100	Y	(*8 clauses*)		(*9 clauses*)		(*22 clauses*)	
f1000	1000	4250	Y	100	227.649	100	9.634	100	1.091
f2000	2000	8500	Y	(*6 clauses*)		100	21.853	100	17.169

Table 2. Random instances (if no assignment is found then the best number of false clauses is given between parentheses)

4.4 Discussions

Note that we report the best number of false clauses when no assignment is found. We could have reported the average performance but the standard deviation is very small for all the solvers.

When the benchmarks are successfully solved, we mention the execution time. From a more abstract point of view, as already mentioned, we approximatively provide the same computation resource in terms of basic operations (such as flips) to each algorithm. Therefore the computation time allowed may be considered to be quite fair.

We do not mention comparisons with complete solvers since, due to the size of the benchmarks, we are more interested in the MAX-SAT problem which is not usually addressed by complete solvers. Moreover, while these solvers can be efficient on the smallest satisfiable structured instances, they do not provide interesting results on larger structured or random instances, especially when these instances are unsatisfiable.

We should mention that we have experimented separately the two main components of our algorithm (crossover and tabu search). TS and crossover alone are able to find solutions but their combination provide the best average results.

5 Conclusion

We have presented the GASAT algorithm, a genetic hybrid algorithm for the SAT problem (and MAX-SAT). This algorithm is built on a specific crossover operator which relies on structural information among clauses and a simple tabu search operator. By their global and local nature, the crossover and tabu search operators act interactively to ensure a good compromise between exploration and exploitation of the search space. Moreover, a selection mechanism based on the Hamming distance is also employed to preserve the diversity of the population.

GASAT was evaluated on both structured and random benchmark instances. Its performance was compared with two state-of-the-art algorithms (Walksat and UnitWalk). The experimentations show that GASAT gives globally very competitive results. In particular, it performs better than the two competing algorithms on the graph coloring (structured) instances. Meanwhile, it seems that GASAT performs less well on some random instances.

The algorithm reported in this paper is a preliminary version of GASAT. Studies are on the way to have a better understanding of its behavior with respect to different classes of problem instances. We are also working on other issues to improve further upon its performance; in particular, a diversification process for tabu search based on unit propagation and criterions for choosing the variables to be flipped in the crossover.

Acknowledgments The work presented in this paper is partially supported by grants from the LIAMA laboratory and the PRA program. We would like to thank the anonymous referees for their helpful comments and remarks.

References

1. Belaid Benhamou and Lakhdar Sais. Theoretical study of symmetries in propositional calculus and applications. In *CADE'92*, pages 281–294, 1992.
2. May Beresin, Eugene Levine, and John Winn. A chessboard coloring problem. *The College Mathematics Journal*, 20(2):106–114, 1989.
3. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, Jul 1962.
4. Kenneth A. De Jong and William M. Spears. Using genetic algorithm to solve NP-complete problems. In *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 124–132, San Mateo, CA, 1989.
5. Olivier Dubois, Pascal André, Yacine Boufkhad, and Jacques Carlier. SAT versus UNSAT. In *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 415–436, 1996.
6. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In Bernhard Nebel, editor, *Proc. of the IJCAI'01*, pages 248–253, San Francisco, CA, 2001.
7. Agoston E. Eiben, Jan K. van der Hauw, and Jano I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

8. Charles Fleurent and Jacques A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1997.
9. Michael R. Garey and David S. Johnson. *Computers and Intractability , A Guide to the Theory of NP-Completeness.* W.H. Freeman & Company, San Francisco, 1978.
10. Fred Glover and Manuel Laguna. *Tabu Search.* Kluwer Academic Publishers, 1998.
11. Jens Goettlieb, Elena Marchiori, and Claudio Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50, 2002.
12. Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950.
13. Jin-Kao Hao and Raphael Dorne. A new population-based method for satisfiability problems. In *Proc. of the 11th European Conf. on Artificial Intelligence*, pages 135–139, Amsterdam, 1994.
14. Edward A. Hirsch and Arist Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. PDMI preprint 9/2001, Steklov Institute of Mathematics at St.Petersburg, 2001.
15. Wengi Huang and Renchao Jin. Solar, a quasi physical algorithm for sat. *Science in China (Series E)*, 2(27):179–186, 1997.
16. Henry Kautz and Bart Selman. Workshop on theory and applications of satisfiability testing (SAT2001). In *Electronic Notes in Discrete Mathematics*, volume 9, June 2001.
17. Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proc. of the AAAI'00*, pages 291–296, 2000.
18. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of the IJCAI'97*, pages 366–371, 1997.
19. Chu Min Li, Bernard Jurkowiak, and Paul W. Purdom. Integrating symmetry breaking into a dll procedure. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT2002)*, pages 149–155, 2002.
20. Bertrand Mazure, Lakhdar Saïs, and Éric Grégoire. Tabu search for SAT. In *Proc. of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 281–285, Providence, Rhode Island, 1997.
21. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of the 38th Design Automation Conference (DAC'01)*, Jun 2001.
22. Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. of the AAAI, Vol. 1*, pages 337–343, 1994.
23. Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the AAAI'92*, pages 440–446, San Jose, CA, 1992.
24. Laurent Simon, Daniel Le Berre, and Edward A. Hirsch. The sat2002 competition. Technical report, Fifth International Symposium on the Theory and Applications of Satisfiability Testing, May 2002.
25. William M. Spears. Simulated annealing for hard satisfiability problems. In *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 533–558, 1996.
26. Hantao Zhang. SATO: An efficient propositional prover. In *Proc. of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, 1997.