Critical Parallelization of Local Search for MAX-SAT

Andrea Roli[1]* and Christian Blum[2]

[1] DEIS
Università degli Studi di Bologna
Viale Risorgimento, 2 – Bologna (Italia)
aroli@deis.unibo.it
[2] IRIDIA
Université Libre de Bruxelles
Av. Roosvelt, 50 – Bruxelles (Belgium)
cblum@ulb.ac.be

Abstract. In this work we investigate the effects of the parallelization of a local search algorithm for MAX-SAT. The variables of the problem are divided in subsets and local search is applied to each of them in parallel, supposing that variables belonging to other subsets remain unchanged. We show empirical evidence for the existence of a critical level of parallelism which leads to the best performance. This result allows to improve local search and adds new elements to the investigation of criticality and parallelism in combinatorial optimization problems.

1 Introduction

Combinatorial Optimization Problems constitute an important challenge for Artificial Intelligence and related fields. Even though the most difficult ones are NP-hard and they require exponential computational time in the worst case, the design of efficient and effective algorithms is a very active research field. Especially for very large instances, *complete* algorithms (i.e., they find the optimal solution in bounded time) may require too much computational time, and *incomplete* (*approximate*) algorithms are preferred. Those algorithms do not guarantee to find the optimal solution in bounded time, but they are generally efficient in finding suboptimal solutions.

Approximate algorithms often employ *Local Search* (LS) as a very powerful tool for solution quality and speed improvement. LS algorithms start with an initial solution and they try to improve it by local changes; those *moves* change a coded solution into a neighboring solution. For example, if a solution is coded as set of binary variables, a possible local move is changing the value of one variable. The drawback is that LS gets trapped in *local optima* which can be very poor with respect to *global optima*; therefore, LS requires a strategy which helps to

* Corresponding author. This work was partially developed during a visiting period at IRIDIA – Université Libre de Bruxelles.

escape from poor local optima and guides the search toward good ones. General strategies for local search are extensively used in *metaheuristics*. Among these we mention Tabu Search [7], Simulated Annealing [12], Evolutionary Computation and Memetics Algorithms [2, 17], Ant Colony Optimization [5], Iterated Local Search [20] and Variable Neighborhood Search [8]. Surveys and current research results on metaheuristics can be found in [21, 3].

Beside general strategies, an effective yet simple way to improve LS is the *parallelization* of search, introduced and discussed in [13, 4, 11, 10]. The common idea among those approaches is to perform more than one local move in parallel[1]. This local search variant can be also embedded in metaheuristic algorithms.

In [11, 10] the set of binary variables of the problem is decomposed in subsets (called *patches*) and for each subset the value of a variable is changed if this improves the objective function restricted to that subset. A local move can be improving for a subset, but since relations among variables remain unchanged, it can be non-improving for other subsets. Nevertheless, it has been shown that when the system is not divided in patches, LS often leads to poor local optima; but as parallelism increases, LS is able to find better and better solutions, up to a point where the performance starts to decrease again. The conclusion is that there exists an optimum degree of parallelism, which allows to get the best average solution quality. In [13] the authors apply a LS move to each variable with probability p. This means that in each iteration pn moves are performed on average, where n is the number of variables. Hence, the parallelism of the algorithm is pn. They show that there is a critical value p_c such that the average solution value reached is the best for $p = p_c$.

The aim of the present work is to probe whether similar effects can be found also with other local search algorithms and in different kinds of problems. In this paper we investigate the effects of parallelizing a local search algorithm applied to MAX-SAT problems. MAX-SAT is an optimization version of the Satisfiability Problem [6] and it is defined by a set of clauses, which are disjunctions of literals (binary variables negated or not): the objective is to minimize the number of unsatisfied clauses (or, equivalently, maximize the number of satisfied clauses). We applied a parallel version of a local search procedure [19] to MAX-SAT and we discovered the existence of an optimal degree of parallelism which allows LS to reach the best performance. This result supports the general conjecture that the optimization of a system composed of conflicting elements can be improved by dividing the system into an optimal number of components and optimize each of them as if it was the whole system. Additionally, parallel local moves improve algorithm effectiveness, especially for large instances.

This paper is organized as follows: in Sect. 2 we introduce the basic concepts and principles of parallel local search, while in Sect. 3 we describe their application to MAX-SAT problems. In Sect. 4 we show experimental results, which are discussed in Sect. 5. Finally, in Sect. 6 we present conclusions and further work.

[1] Moves are performed *synchronously* and they can be implemented either in sequential or parallel algorithms.

2 Local Search and Parallelism

A Constraint Optimization Problem (COP) can be defined as follows:

- a set of variables $X = \{x_1, \ldots, x_n\}$;
- variable domains $D_1, \ldots, D_n$;
- constraints among variables;
- an *objective function* to be minimized $F : D_1 \times \ldots \times D_n \to \mathbb{R}^+$;

Local search (LS) algorithms require the definition of *states*, which are (feasible) configurations of variables and of a *neighborhood function*, which assigns to each state s a set of feasible states $\mathcal{N}(s)$. The simplest LS constructs a path in the state space by moving from a state s to another state in $\mathcal{N}(s)$. The transition from a state to the successor is the result of the application of a *move operator* M (implicitly defined by the neighborhood function) on s. The search stops either when it reaches a local optimum, or when the maximum number of iterations or non-improving moves has been reached.

In [13, 4, 10, 11] some studies on the parallelization of local search algorithms are described. In [13], the authors apply a parallel version of Simulated Annealing [12] to the optimization of NK-landscapes [9, 10]. Suppose to have a minimization problem on N boolean variables; the search space can be represented as an *energy landscape*: the goal is to find a minimum in this landscape. Every variable x_i is associated to an energy value e_i, which is a function of x_i and other K variables. The objective function of the system (total energy) is $E = \frac{1}{N}\sum_{i=1}^{N} e_i$. A move from state s_1 to state s_2 results in an energy difference $\Delta E = E(s_2) - E(s_1)$. The application of the move operator is, in this case, simply a *flip* of a variable (i.e., $x_i \leftarrow \sim x_i$). The basic algorithm behaves as follows: it randomly selects a variable and flips it; it accepts this move with probability 1 if $\Delta E \leq 0$ and with probability $\exp(-\Delta E/T)$ if $\Delta E > 0$. T is a *temperature* parameter, which controls the annealing schedule: the higher T, the higher the probability to choose a non-improving move. In the parallel version, every variable $x_i\,(i = 1, 2, \ldots, n)$ has probability p of being selected, that is, at each iteration pn parallel variable flips are tested on average. Hence, the degree of parallelism of search is $\tau = pn$. The authors discover that there is a p_{opt} for which the algorithm finds the solution with the lowest E: higher or lower values of p on average produce higher total energy values.

Since the effect of a variable flip on the objective function value is evaluated as if it was the only one to change, parallel (i.e., simultaneous) flips introduce a kind of noise in the energy evaluation. As observed in [14], the introduction of noise increases the effectiveness of local search, since it helps to escape from local optima. It is worth to note that [14] shows that the quality of solutions found increases as noise increases, up to a critical value above which the performance decreases again. However, differences and similarities between parallel LS and LS with noise have still to be completely discovered and explained.

Analogous results are reached in [10, 11], where yet a different approach is chosen. The COP is, in this case, the optimization of a NK-landscape with variables arranged in a bidimensional lattice; every variable corresponds to a cell in the lattice and K indicates the number of neighboring cells linked to it. The lattice is divided in P nonoverlapping patches and a simple local search is applied in parallel to each patch. A variable is flipped if it decreases the energy of the patch to which it belongs. The authors find an optimal number of patches which allows the search to reach the lowest total energy value.

The underlying principle of the last approach is that, in order to optimize systems composed of conflicting elements, it is generally useful dividing the system in subsystems and optimize each of them independently. One of the effects of simultaneous changes is to help the search to avoid local optima, as they introduce a kind of *noise* due to the fact that each subset performs a local move supposing the other subsets do not change. Moreover, the authors claim that the optimal subdivision drives the system in a state such that subsystems coordinate themselves for a global optimization goal.

These works on parallelization of search propose very useful ideas for the improvement of local search for COPs and suggest new directions to understand local search behavior. In the following, we present an application of LS parallelization applied to MAX-SAT and we show empirical evidence for the existence of a critical level of parallelism.

We would like to stress that the algorithm of the above-cited and present works are implemented sequentially. With "parallel moves" we mean "synchronous moves". Anyhow, these results could benefit also implementations on parallel architectures.

3 Parallel Local Search for MAX-SAT

MAX-SAT is an optimization version of the Satisfiability Problem (SAT). SAT belongs to the class of NP-complete problems [6] and can be stated as follows: given a set of clauses, each of which is the logical disjunction of $k > 2$ *literals* (a literal is a variable negated or not), we ask whether an assignment to the variables exists that satisfies all the clauses. It is possible to define a representation of MAX-SAT according to the definition of COP given in Sect. 2:

- a set of binary variables $X = \{x_1, \ldots, x_n\}$;
- variable domains $D_1 = \ldots = D_n = \{0, 1\}$;
- no constraints among variables;
- an *objective function* to be minimized: E = number of unsatisfied clauses;

In the literature, effective local search algorithms to solve MAX-SAT have been proposed [15, 1, 22]. In this paper we investigate the parallelization of a simple, yet effective, local search algorithm: MAX-GSAT. MAX-GSAT is a MAX-SAT version of GSAT, a well-known local search procedure for SAT [19]. The high-level description of MAX-GSAT is reported in Fig.1. MAX-GSAT works as follows: at each iteration, it flips the variable which decreases the number of unsatisfied clauses the most. It is not a pure hill climbing algorithm, as it accepts also moves which either produce the same objective function value or increase

```
procedure MAX-GSAT
Input: a set of clauses α, MAXMOVES
Output: a truth assignment of α
begin
    T := initial truth assignment;
    MOVES := 0; besterror := Eval(α, T); bestsolution := T;
    while MOVES < MAXMOVES do
        if T satisfies α then return T;
        p := a propositional variable such that a change in its truth assignment
            gives the largest decrease in number of clauses of α
            that are not satisfied by T;
        T := T with p reversed;
        MOVES := MOVES + 1;
        error := Eval(α, T);
        if error < besterror then
        begin
            MOVES := 0; bestsolution := T; besterror := error;
        end
    end while
return bestsolution;
end
```

Fig. 1. High-level description of MAX-GSAT algorithm

it. The search stops when the maximum number of non-improving moves is reached (or a satisfying assignment is found, i.e., $E = 0$). In our implementation this number is equal to the number of variables.

Our parallelization of MAX-GSAT (thereafter referred to as MAX-PGSAT) is straightforward: variables are divided in τ subsets of equal cardinality and for each subset the variable which fulfills the GSAT criterion is flipped. This results in the application of τ simultaneous variable flips. It is important to stress that the evaluation of a variable flip is done as if all the other variables did not change. Relations between variables are still valid, as variable flips are evaluated by taking into account all the clauses a variable is in. The basic MAX-PGSAT algorithm is described in Fig.2.

The composition of subsets can be either decided before starting the algorithm and kept fixed, or it can be reconfigured at each iteration. In our experiments we adopted the second choice, to make results independent of a particularly (un)favourable subdivision of variables.

4 Experimental Results

In this section we describe the experimental analysis on the performance of MAX-PGSAT as a function of parallelism. We tested MAX-PGSAT on random generated unsatisfiable instances with three literals per clause (3-SAT), retrieved

procedure MAX-PGSAT
Input: a set of clauses α, τ, $MAXMOVES$
Output: a truth assignment of α
begin
  $T :=$ initial truth assignment;
  $MOVES := 0$; $besterror := Eval(\alpha, T)$; $bestsolution := T$;
  while $MOVES < MAXMOVES$ **do**
    if T satisfies α **then return** T;
    Divide the set of variables in τ subsets according to a chosen criterion;
    parfor each subset X_k of variables $(k = 1, 2, \ldots, \tau)$
      $p_k :=$ a propositional variable in X_k such that a change in its truth assignment gives the largest decrease in the number of clauses of α that are not satisfied by T;
    end parfor
    $T := T$ with $p_1, p_2, \ldots, p_\tau$ reversed;
    $MOVES := MOVES + 1$;
    $error := Eval(\alpha, T)$;
    if $error < besterror$ **then**
    begin
      $MOVES := 0$; $bestsolution := T$; $besterror := error$;
    end
  end while
return $bestsolution$;
end

Fig. 2. High-level description of MAX-PGSAT, the parallelized version of MAX-GSAT

from SATLIB[2]. We considered nine sets, from 50 to 250 variables and each set is composed of ten instances with a ratio between clauses (m) and variables (n) approximately equal to 4.3 (the *critical region* [16]). The number of subsets τ varies from 1 (only the best flip among all the variables is performed) to $n/2$; at the beginning of each iteration, a number of τ subsets is randomly generated[3].

The graph in Fig.3 reports the average error (number of unsatisfied clauses) as a function of τ for an instance with 100 variables and 430 clauses. This graph shows the typical behavior of the algorithm. Observe that the original algorithm ($\tau = 1$) reaches an average error of 4 and, as τ increases, the error decreases until a minimum at $\tau = \tau_{opt} = 6$; above that value the average error starts to increase.

For all sets of instances, a similar behavior has been noticed.

[2] `http://www.intellektik.informatik.tu-darmstadt.de/SATLIB.`
[3] All subsets have equal cardinality, except for one which contains $\tau + n \ mod \ \tau$ variables.

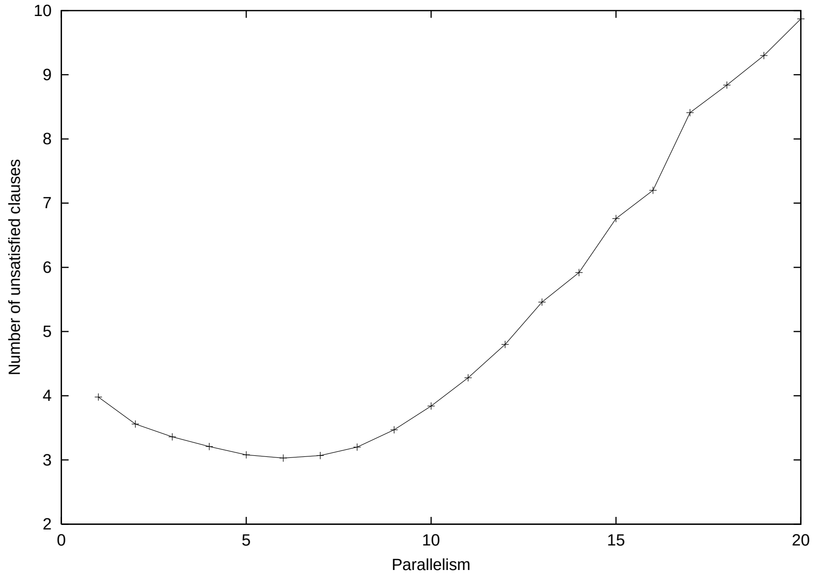

Fig. 3. Average error vs. τ. Instance with 100 variables and 430 clauses. Results are averaged over 10000 runs.

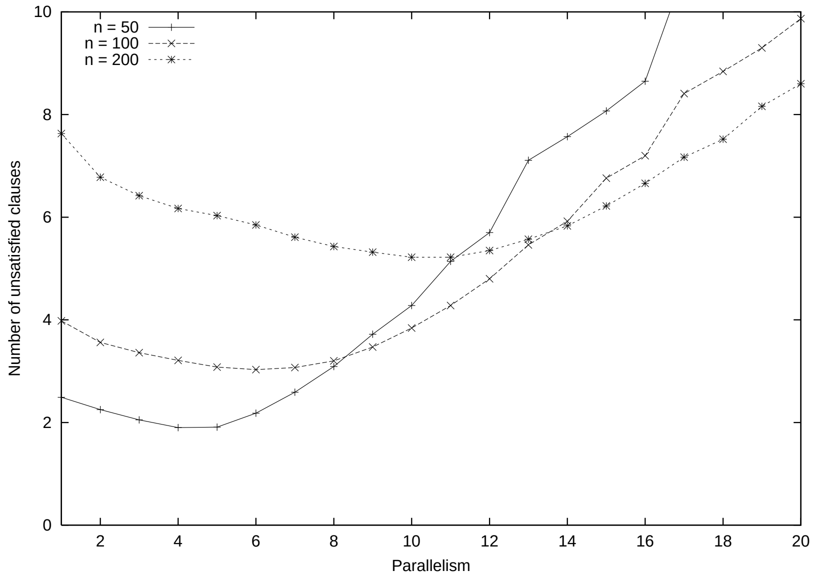

Fig. 4. Average error vs. τ. Instances with 50,100,200 variables and 218,430,860 clauses respectively. Results are averaged over 10000 runs.

Table 1. Median and mean of optimal values of τ for each set of instances.

number of variables	Median τ_{opt}	Mean τ_{opt}
50	4	4.18
75	6	5.9
100	6	6.3
125	7	7.5
150	8	9.4
175	10	10.33
200	11	10.7
225	12	12.36
250	12	12.6

Table 2. Average connectivity and its normalized value for typical 3-SAT random generated instances.

number of variables	q	$\overline{q}$
50	19.88	0.3976
75	21.76	0.2901
100	22.88	0.2288
125	23.33	0.1866
150	23.99	0.1599
175	24.00	0.1371
200	24.11	0.1206
225	24.21	0.1076
250	24.34	0.0973

Moreover, as can be observed in Fig.4, the higher the number of variables, the higher τ_{opt}. This result is summarized in Table 1, where for each set median and mean of τ_{opt} are reported[4].

We conjecture that the fact that τ_{opt} increases as n increases, while m/n remains constant, is due to the decrease of the average amount of *connections* between variables. We define *average connectivity* q as the average number of variables connected with any variable (two variables are connected if they are in a same clause). More formally, let us represent the structure of a MAX-SAT instance as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where each $v \in \mathcal{V}$ corresponds to a variable and arc $(v_i, v_j) \in \mathcal{A}$ $(i \neq j)$ if and only if variables v_i and v_j are in a same clause. q is then the average number of arcs per node. The minimum value for q is 0 (all variables are disconnected) and the maximum is $n-1$ (i.e., $\mathcal{G}$ is fully connected). We will consider the normalization of q, defined as $\overline{q} = q/n$. Table 2 shows the typical values of $\overline{q}$ for the instances considered in this analysis. Fig.5 shows the graphic combination of Table 1 and Table 2. We notice that the curve representing τ_{opt} vs. $\overline{q}$ is monotonically non increasing. As explanation of this phenomenon we argue that when the relations between variables are loose (i.e., $\overline{q}$ is low), a variable flip affects a relatively small number of other variables, thus the system can be subdivided into several nearly independent subsets. On the contrary, tight relations produce a large network of dependencies among the variables and thus they are more sensitive to single flips.

A further analysis of the statistics presented earlier in this section, allows to discover an interesting phenomenon. For example, the graph of Fig.6 shows the average error and the error variance as a function of τ for an instance with 100 variables. We observe that the minimum error is obtained for a value of τ slightly smaller than the value for which the variance has a minimum. We suppose that near the critical value τ_{opt} the algorithm achieves the highest effectiveness and thus it converges toward a smaller region around the best value it can find.

[4] The average error has been evaluated over 100 runs and then median and mean of τ_{opt} over the 10 instances of each set have been considered.

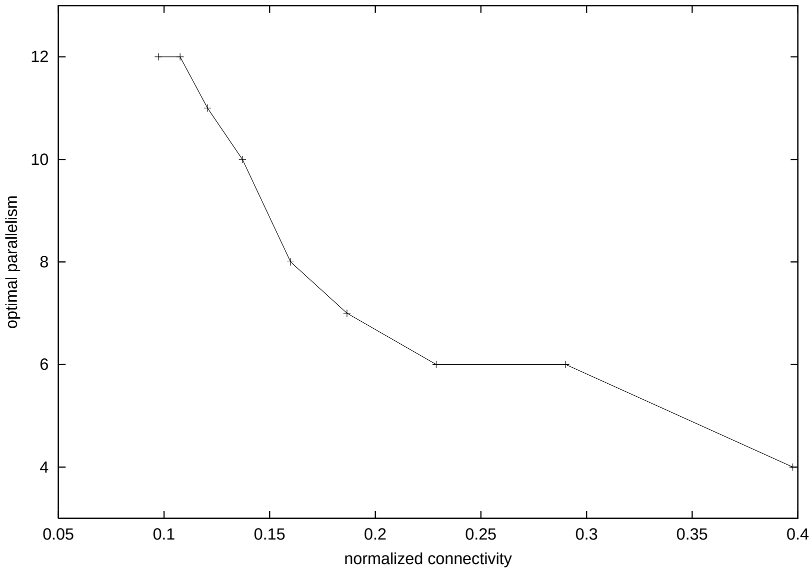

Fig. 5. Optimal parallelism τ_{opt} vs. normalized average connectivity $\overline{q}$.

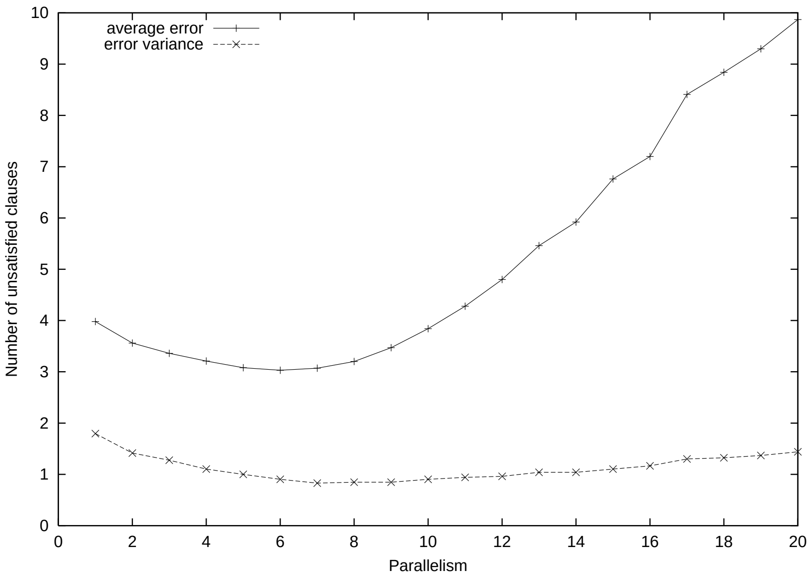

Fig. 6. Average error and variance vs. τ. Instance with 100 variables. Results are averaged over 10000 runs.

5 Discussion

We think that an explanation for the existence of an optimum level of parallelism can be given in terms of optimal trade-off between *exploration* and *exploitation*, as explained in the following. Exploration is achieved by long jumps in the search space, thus high number of simultaneous flips. Exploitation is obtained by an intensified search in a promising region of the search space. This is achieved with very few parallel variable flips.

It is worth mentioning that analogous results have been discovered in the parallelization of GSAT applied to satisfiable instances of SAT [18]. In this case, τ_{opt} corresponds to the best overall performance, not only as number of solved instances, but also iterations and computational time.

Finally, some considerations about the computational time[5]. The algorithm presented has been implemented sequentially and the complexity introduced by the application of τ simultaneous flips is low. The evaluation of the effect of a flip in the objective function requires a constant computational time, thanks to an efficient data structure implementation[6]. Thus, the evaluation phase for τ subsets is proportional to $\tau \cdot n/\tau = n$. However, additional computational time is required for updating data structures after τ flips; this time is proportional to τm, because for each variable flip the data structures of all the connected variables have to be updated. Anyway, on the average, this does not introduce a significant overhead; for example, for the largest instances (250 variables, 1065 clauses) the average time ranges from $70ms$ for $\tau = 1$ to $110ms$ for $\tau_{opt} = 12$.

6 Conclusion and Future Work

We have shown that implicit parallelization in local search applied to MAX-SAT leads to a better average performance of search. The parallelization is controlled by a parameter for which there exists a critical value which determines the optimal algorithm performance. These results are adding to previous ones on criticality and parallelism in combinatorial optimization problems, which show that parallelism, system connectivity and quality of solutions are strictly related. Further analyses are required to know whether the phase transition discovered in [13, 10, 11] is present also in the case of MAX-PGSAT on MAX-SAT instances.

We are currently extending the experiments to other MAX-SAT benchmark sets, including structured instances. Moreover, we are testing MAX-PGSAT on very large instances and results are very promising, especially if the parallel phase is subject to a *schedule*, whereby τ is progressively decreased. In this case, the balance between exploration and exploitation is varied during the search process. In a first phase, a high level of parallelism allows to reach a promising

[5] The algorithm has been implemented in C, compiled with *gcc* (Linux) and ran on a Pentium II 400MHz with 512 MB of RAM and 512 KB of cache memory.

[6] Data structures are similar to those used in the WalkSAT code retrieved from SATLIB.

region of the search space faster. Then, a progressive decrease of parallelism leads the search to converge toward a good local optimum in that region.

Future work is focused on the study of problem parameters which are linked to the behavior of parallel local search and to the formal analysis of local search algorithms in the context of dynamic systems. In particular, the analysis of the properties of the graph defined by variables relations is very promising. Parameters and properties such like *average connectivity*, *diameter*, *stability* and *percolation flows* can be used to divide the system in clusters of correlated variables and thus find the optimal *ad hoc* subdivision. They also could be useful to extract information about the problem itself.

Acknowledgments

We would like to thank Michela Milano for her suggestions and useful discussions; we also thank Thomas Stützle for having provided part of the code. This work was partially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. Andrea Roli acknowledges support from the CEC through a "Marie Curie Training Site" (contract HPMT-CT-2000-00032) fellowship. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] Roberto Battiti and Marco Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 1997.

[2] Patrice Calegari, Giovanni Coray, Alain Hertz, Daniel Kobler, and Pierre Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5:145–158, 1999.

[3] David Corne, Marco Dorigo, and Fred Glover, editors. *New Ideas in Optimization*. Advanced topics in computer science series. McGraw-Hill, 1999.

[4] Frank-Michael Dittes. Optimization on rugged landscapes: A new general purpose Monte Carlo approach. *Physical Review Letters*, 76(25):4651–4655, June 1996.

[5] Marco Dorigo and Gianni Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999. Also available as Technical Report IRIDIA/99-1, Université Libre de Bruxelles, Belgium.

[6] Michael R. Garey and David S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[7] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publichers, 1997.

[8] Pierre Hansen and Nenad Mladenovic. An introduction to Variable Neighborhood Search. In Stefan Voss, Silvano Martello, Ibrahim Osman, and Catherine Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*, chapter 30, pages 433–458. Kluwer Academic Publishers, 1999.

[9] Stuart A. Kauffman. *The origins of order.* Oxford University Press, New York, 1993.

[10] Stuart A. Kauffman. *At home in the universe.* Oxford Press, 1995.

[11] Stuart A. Kauffman and William Macready. Technological evolution and adaptive organizations. *Complexity*, 26(2):26–43, March 1995.

[12] S. Kirkpartick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, 13 May 1983*, 220(4598):671–680, 1983.

[13] William G. Macready, Athanassios G. Siapas, and Stuart A. Kauffman. Criticality and parallelism in combinatorial optimization. *Science*, 271:56–59, January 1996.

[14] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 321–326, Menlo Park, July 27–31 1997. AAAI Press.

[15] Patrick Mills and Edward Tsang. Guided Local Search for solving SAT and weighted MAX-SAT Problems. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000*, pages 89–106. IOS Press, 2000.

[16] David G. Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of sat problems. In *Proceedings, Tenth National Conference on Artificial Intelligence*, pages 459–465. AAAI Press/MIT Press, July 1992.

[17] P. Moscato. Memetic algorithms: A short introduction. In F. Glover D. Corne and M. Dorigo, editors, *New Ideas in Optimization.* McGraw-Hill, 1999.

[18] Andrea Roli. Criticality and parallelism in GSAT. In Henry Kautz and Bart Selman, editors, *Electronic Notes in Discrete Mathematics*, volume 9. Elsevier Science Publishers, 2001.

[19] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. American Association for Artificial Intelligence, AAAI Press.

[20] Thomas Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications.* DISKI - Dissertationen zur Künstliken Intelligenz. infix, 1999.

[21] Stefan Voss, Silvano Martello, Ibrahim H. Osman, and Catherine Roucairol, editors. *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, 1999.

[22] Benjamin W. Wah and Yi Shang. Discrete Lagrangian-Based Search for Solving MAX-SAT Problems. In *Proc. 15th Int'l Joint Conf. on Artificial Intelligence, IJCAI*, pages 378–383, Aug. 1997.