Using Cross-Entropy for Satisfiability *

Hana Chockler
IBM Research
Haifa, Israel
hanac@il.ibm.com

Alexander Ivrii
IBM Research
Haifa, Israel
alexi@il.ibm.com

Arie Matsliah
IBM Research
Haifa, Israel
ariem@il.ibm.com

Simone Fulvio Rollini
University of Lugano
Lugano, Switzerland
rollinis@usi.ch

Natasha Sharygina
University of Lugano
Lugano, Switzerland
natasha.sharygina@usi.ch

ABSTRACT

This paper proposes a novel approach to SAT solving by using the cross-entropy method for optimization. It introduces an extension of the Boolean satisfiability setting to a multi-valued framework, where a probability space is induced over the set of all possible assignments. For a given formula, a cross-entropy-based algorithm (implemented in a tool named CROiSSANT) is used to find a satisfying assignment by applying an iterative procedure that optimizes an objective function correlated with the likelihood of satisfaction.

We investigate a hybrid approach by employing cross-entropy as a preprocessing step to SAT solving. First CROiSSANT is run to identify the areas of the search space that are more likely to contain a satisfying assignment; this information is then given to a DPLL-based SAT solver as a partial or a complete assignment that is used to suggest variables assignments in the search.

We tested our approach on a set of benchmarks, in different configurations of tunable parameters of the cross-entropy algorithm; as experimental results show, it represents a sound basis for the development of a cross-entropy-based SAT solver.

Categories and Subject Descriptors

F.5.3 [**Mathematical Optimization**]: Discrete Optimization; G.1.1 [**Combinatorics**]: Combinatorial Optimization—*Cross-entropy*; F.4.1 [**Logic**]: Logic and Verification—*Satisfiability*

Keywords

SAT, Cross-entropy, Optimization, Pre-processing, Multi-valued Boolean logic

1. INTRODUCTION

The practical significance of the Boolean satisfiability problem has led to major research efforts both among theoreticians and practitioners in this area, which resulted in powerful tools. Nowadays, SAT solvers are routinely used to solve real-life problems represented by formulae with millions of variables in a reasonable time. Overall, however, there is no known procedure (as depicted by the P versus NP problem) that can solve efficiently all instances of SAT. Often these highly engineered tools cannot solve seemingly simple problems. In general, it is virtually impossible to predict which instances can be solved easily and which can not without trying to find the actual solution.

Several techniques have been developed in the satisfiability framework, both incomplete (unable to prove unsatisfiability of a formula) and complete. Most of the successful SAT tools fall into two main classes: 1) the ones (complete) which implement the Davis-Putnam-Loveland-Logemann (DPLL) algorithm [9, 10], where the tools systematically traverse and backtrack a binary tree whose nodes represent partial assignments; and 2) the others (incomplete) based on stochastic local search (SLS) where solvers guess a full assignment and, if it is not a satisfying one, use different heuristics to reassign the values to variables (see [20, 42] for comprehensive surveys). Various heuristics are employed by the stochastic methods to escape local minima and to ensure that the previously tried combinations of values are not assigned in the subsequent tries. Based on the annual competitions of SAT solvers, the DPLL-based tools demonstrate better performance when applied to real-life problems as they take advantage of (and learn from) the structure of the solved instances (see, for example, [12, 32]). On the other hand, stochastic-based approaches tend to exhibit better performance on random instances, especially in conjunction with other techniques (for example, [26] proposes to use the power of GPUs to speed-up SAT computations).

In the last years, many authors have dedicated themselves to combining DPLL and stochastic search-based algorithms in an effective manner, trying to exploit the benefits of both worlds to obtain a competitive complete hybrid solver [40].

These kinds of hybrid approaches have been proved particularly successful; still, they offer many challenges in terms of finding optimal couplings of deterministic and stochastic techniques, global and local strategies of exploration. In this paper, we propose a novel approach to the satisfiability problem built on a model-based stochastic technique known as the *cross-entropy method* and on its adaptation to combinatorial optimization.

The cross-entropy method is a generic approach to rare event simulation and to combinatorial optimization [36]. It derives its name from the cross-entropy or the Kullback-Leibler divergence, which is a fundamental concept of modern information theory [24]. It is an iterative approach based on minimizing the cross-entropy or the Kullback-Leibler divergence between probability distributions. The cross-entropy method was motivated by an adaptive algorithm

*This work is partially supported by the European Community under the call FP7-ICT-2009-5 – project PINCETTE 257647.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

for estimating probabilities of rare events in complex stochastic networks [34]. Then, it was realized that a simple modification allows the use of this method also for solving hard optimization problems, in which there is a performance function associated with the inputs. The cross-entropy method in optimization problems is used to find a set of inputs on which the performance function reaches its global maximum (or minimum), where the input space is assumed to be too large to allow exhaustive exploration. The method is based on an iterative sampling of the input space according to a given probability distribution over this space. Each iteration consists of the following phases:

1. Generate a set of random samples according to a specified mechanism.
2. Update the parameters of the random mechanism based on the data to produce "better" samples in the next iteration, where "better" depends on the chosen performance function.

The initial probability distribution is assumed to be a part of the input. Samples are evaluated according to the performance function. The procedure terminates when a sample with the maximal (or the minimal) value of the performance function is generated, or, if the global extremum is unknown in advance, when an extremum is detected.

We propose an algorithm based on the cross-entropy method and on an extension of the Boolean satisfiability problem to a multi-valued setting. Given a Boolean formula φ in CNF over a set $\mathcal{V}$ of propositional variables, we turn the Boolean range $\{0, 1\}$ into a discrete set $\mathcal{D}$ such that every element of $\mathcal{D}$ is between 0 and 1. Intuitively, these values provide a measure of the "level of truth" of a variable. The *probability space* is derived from all possible assignments to $\mathcal{V}$ over $\mathcal{D}$; a probability distribution $\mathcal{P}$ assigns a probability to each variable and value. We define the semantics of the Boolean connectives over $\mathcal{D}$ and examine some performance functions $\mathbb{S}$ that correlate with the evaluation of φ, in such a way that, if φ is satisfiable, then a global minimum of $\mathbb{S}$ is reached under a satisfying assignment. Starting with a uniform distribution, we execute an algorithm based on the cross-entropy method with SAT-specific optimizations to minimize the value of the chosen performance function.

We carry out an investigation towards the development of a new hybrid DPLL-stochastic technique by letting our cross-entropy algorithm perform an initial fast exploration of the probability space. Assuming a satisfying assignment has not been found, from the values of the best-performing sample (following the intuition about "level of truth") a Boolean partial assignment is derived, which in turn is used by the SAT solver to decide the polarities to be given to variables during the search.

We implemented our variant of the cross-entropy method in a tool named *CROiSSANT* and tested it in combination with state-of-the art SAT solvers on different sets of benchmarks; as the experimental results show, it is a sound basis for the development of a cross-entropy-based SAT solver.

In summary, the contributions of this paper are:

- A new theoretical framework that extends the standard Boolean satisfiability problem to a multi-valued model, suitable for optimization techniques like the cross-entropy method.
- The first systematic application of the cross-entropy method to satisfiability (to the best of our knowledge), and the development of a cross-entropy-based algorithm with SAT-specific optimizations.
- A new hybrid DPLL-stochastic technique that consists in a simple form of interaction between CROiSSANT and a SAT solver, where the information provided by the former (in terms of an optimal sample) is used by the latter to drive variable polarities at decision time.

2. RELATED WORK

It is natural to consider satisfiability as an optimization problem on the space of complete variables assignments, with the performance function evaluating, for example, the number of satisfied clauses. As such, there is considerable ongoing research on applying optimization techniques to satisfiability. A number of powerful stochastic solvers is based on various extensions of the random walk model, see for example GSAT [41], WalkSAT [39], HSAT [16], SDF [38], Novelty and R-Novelty [30], Novelty+ and R-Novelty+ [19], UnitWalk [18].

In contrast to these techniques, our approach is to iteratively adjust the probability distribution, according to which assignments are generated. Also, while nearly all known stochastic approaches for Boolean satisfiability use a binary range for the variables, we use smoother ranges, which allows a more gradual improvement. Surprisingly there is no prior work on the application of the cross-entropy method to satisfiability, to the best of our knowledge, apart from a brief mention in [36]. At the same time, the cross-entropy method is used in many other areas, including buffer allocation [1], neural computation [11], DNA sequence alignment [22], scheduling [28], graph problems [35], and, more recently, testing [5] and replay [6] of concurrent programs. It was shown to be very effective in discovering or approximating solutions to very hard problems for which it is possible to define a "good" (smooth and gradually improving) performance function.

In the last years there has been extensive research on combining stochastic local search (SLS) and conflict driven clause learning. Three main research branches have been followed: using DPLL as an aid for SLS or vice versa SLS as a helper for DPLL, either before or at solving time [8, 13, 15, 17, 21, 29]; letting the two techniques interact in a synergistic way [2–4, 14, 25].

In [29] it is suggested, for an unsatisfiable formula, to use a stochastic solver to try to identify its minimal unsatisfiable cores, and then to use a complete solver on the smaller resulting subformula. Moreover, the scores delivered by the stochastic solver could be used to guide the branching strategy of the complete solver.

Using a stochastic solver to suggest polarities for branching variables is probably even more natural, however to our knowledge it has not been explicitly suggested in the literature. There is also a strong connection between setting the initial branching polarities and the progress saving heuristic, suggested in [33] and used in all state of the art CDCL solvers.

The rest of the paper is organized as follows. We start with the necessary definitions in Section 3. The extended satisfiability framework, as well as the CROiSSANT algorithm, are presented in Section 4, while experimental results are described in Section 5. We conclude the paper and discuss future work in Section 6.

3. PRELIMINARIES

3.1 SAT definitions

$\mathcal{V}$ is a set of propositional variables $v_1, \ldots, v_n$; a literal l is a variable, either with positive (v) or negative ($\overline{v}$) polarity. A clause C is a finite disjunction of literals $l_1 \vee \ldots \vee l_{|C|}$; a formula φ in conjunctive normal form (CNF) is a finite conjunction of clauses

$C_1 \wedge \ldots \wedge C_m$. A Boolean assignment $e : \mathcal{V} \to \{true, false\}$ is a function that assigns either value $true$ or $false$ to some variables $v \in \mathcal{V}$; the assignment is complete if every variable is assigned a value, otherwise it is partial. An assignment e is said to satisfy a formula φ if φ evaluates to $true$ under e; conversely, it falsifies φ if φ evaluates to $false$.

3.2 Cross-entropy for optimization

In this section, we present a brief overview of the cross-entropy method for combinatorial optimization. The cross-entropy method was originally developed in order to efficiently estimate probabilities of rare events and was later shown to be effective in solving combinatorial optimization problems [36].

In this setting, we are given a very large probability space $\mathcal{X}$, a probability distribution f defined on $\mathcal{X}$, and a function $\mathbb{S}$ from this space to $\mathbb{R}^+$, and we are searching for an element X of $\mathcal{X}$ on which $\mathbb{S}$ reaches its global maximum. The function $\mathbb{S}$ is called the *performance function*, and intuitively, it measures "fitness" of the elements of $\mathcal{X}$. The space is assumed to be very large and infeasible to search exhaustively, and the global maximum is rare, so that the probability of finding it by random sampling according to the given probability distribution f is very low. In addition, in many problems we do not know the value of the global maximum in advance. The problem would be solved if we had another distribution g that increases the probability of finding the global maximum. The ideal distribution here would be $g_{\mathbb{S}}$, which gives probability 1 to inputs on which $\mathbb{S}$ reaches its global maximum and 0 to all other inputs. The cross-entropy method attempts to iteratively approximate $g_{\mathbb{S}}$ by changing the probability distribution on $\mathcal{X}$ so that the distance between the current distribution and the ideal distribution decreases in each iteration. The notion of distance used in this approximation is the Kullback-Leibler divergence or distance (also called cross-entropy). The Kullback-Leibler divergence between g and h is defined as:

$$\mathcal{D}(g,h) = E_g \ln \frac{g(X)}{h(X)} = \int g(x) \ln g(x) dx - \int g(x) \ln h(x) dx$$

Note that this is not a distance in the formal sense, since in general it is not symmetric. Since $g_{\mathbb{S}}$ is unknown, the approximation is done iteratively from the input distribution $f = f_0$, where in iteration t we draw a random sample according to the current distribution f_t and compute the (approximate) cross-entropy between the f_t and $g_{\mathbb{S}}$ based on this sample. The probability distribution f_{t+1} for iteration $t+1$ is computed from f_t and the sample in iteration t so that $\mathcal{D}(g_{\mathbb{S}}, f_{t+1}) < \mathcal{D}(g_{\mathbb{S}}, f_t)$. The exact update formula depends on the formulation of the problem, but, roughly speaking, the probability distribution is modified based on the best quantile of the current sample, so that the elements close to the best quantile get a higher probability in the next iteration.

The process can be halted once the global maximum is reached, if it is known in advance. When this is not the case, a common criterion, which we adopt, is to stop when the best quantile does not significantly change for a number of subsequent iterations. The convergence of our cross-entropy algorithm is discussed in §4.2.

The reader is referred to the book on cross-entropy for a complete description of the method [36].

4. THE CROISSANT APPROACH

In this section, we present the extension of the Boolean model to the multi-valued one, together with the semantics of the Boolean connectives; we also describe a probability space induced over the multi-valued space and discuss the choice of an appropriate performance function. We define the variant of the cross-entropy setting adopted and illustrate the core algorithm of CROiSSANT in detail. Then, we set out the mechanics of the interaction between CROiSSANT and a DPLL-based SAT solver and the intuitions behind that. Finally, we discuss the relationships between the parameters of the CROiSSANT algorithm and their tuning in our experiments.

4.1 The multi-valued model

Intuitively, in order to guarantee convergence to a satisfying assignment of a given CNF formula φ, we need a probability space and a performance function that allow an "evolution" of the assignments towards the satisfying assignment. In particular, this means that the performance function reaches its global maximum on a satisfying assignment, and that the value of the performance function increases when the sample is getting closer to a satisfying assignment. Moreover, the sample space should be dense enough to allow gradual changes of the probability distribution. In the standard Boolean logic every variable can have only two values, and thus its value cannot change gradually. Our approach introduces a multi-valued framework, inspired by *fuzzy logic* [43] and its use of *t-norms*.

Assuming a domain $\mathcal{D} \subseteq [0,1]$, an assignment function $e : \mathcal{V} \to \mathcal{D}$ extends a standard Boolean assignment by associating each variable of $\mathcal{V}$ with a value in $\mathcal{D}$. The function e^* evaluates an arbitrary Boolean formula φ to a number in $[0,1]$, given an assignment e of values to its variables:

- $e^*(v) = d$ if $e(v) = d$
- $e^*(\neg\psi) = 1 - e^*(\psi)$.
- $e^*(\psi \vee \eta) = e^*(\psi) \times e^*(\eta)$.
- $e^*(\psi \wedge \eta) = \max(e^*(\psi), e^*(\eta))$.

In our framework, 0 stands for *true* and 1 stands for *false*, contrary to the traditional way of defining their meaning – this allows a more efficient computation of the value of Boolean formulae.

Remark 1. While the above definition of e^* does not correspond to standard algebraic operations (in particular, there is no distributivity), it is easy to see that e^* applied to $\wedge$ is a t-norm, that is, generalizes conjunction, and that e^* applied to $\vee$ is a t-conorm, that is, generalizes disjunction. In particular, when $\mathcal{D} = \{0,1\}$, e^* matches the standard definitions of $\wedge$ and $\vee$ in De Morgan algebra. Indeed, we have for $d \in \mathcal{D}$:

- $e^*(0 \vee d) = 0$, that is, *true* $\vee\, d =$ *true*;
- $e^*(1 \vee d) = d$, that is, *false* $\vee\, d = d$;
- $e^*(1 \wedge d) = 1$, that is, *false* $\wedge\, d =$ *false*;
- $e^*(0 \wedge d) = d$, that is, *true* $\wedge\, d = d$.

The domain $\mathcal{D}$ could in theory be the interval $[0,1]$. Practically, working in a continuous environment is unfeasible from a computational point of view for a combinatorial problem like satisfiability; thus we adopt a framework that *approximates* the continuous domain by a discrete one, allowing a more efficient computation of the performance function and a simpler mechanism to update probabilities in the cross-entropy algorithm. For $K \in \mathbb{N}, K \geq 2$, the range $\mathcal{D}_K$ of all variables in $\mathcal{V}$ is the set of values $\{0, 1/(K-1), 2/(K-1), \ldots, 1\}$. A larger K provides finer granularity and thus a denser space, while a smaller K leads to more efficient computations.

In what follows, we assume an input formula φ in CNF over a set of variables $\mathcal{V}$, with $|\mathcal{V}| = n$. A probability space is defined over the set $\mathcal{E}$ of all possible complete assignments e over

$\mathcal{V}$, where $e(v) \in \mathcal{D}_K$ for each $v \in \mathcal{V}$. A probability distribution $\mathcal{P} : \mathcal{E} \to [0,1]$ gives to each assignment its probability in the following way. Probabilities for individual variables v and values d are directly specified as $\mathcal{P}(v = d)$. In the cross-entropy framework, variables are treated independently; thus, the probability of an assignment $e(v_1) = d_1, \ldots, e(v_n) = d_n$ is computed as $\mathcal{P}((v_1 = d_1), \ldots, (v_n = d_n)) = \prod_i \mathcal{P}(v_i = d_i)$. Due to this independence condition, the probability distribution is completely represented by means of a matrix $\mathbf{p}$ of size $n \times K$, whose generic element $\mathbf{p}_{i,j}$ is exactly $\mathcal{P}(v_i = d_j)$.

A performance function $\mathbb{S}_\varphi : \mathcal{E} \to [0,1]$ assigns to each $e \in \mathcal{E}$ a value that denotes the "truth level" of φ, that is how close φ is to being satisfied under the assignment e. Dealing with formulae in CNF, the most straightforward choice for $\mathbb{S}$ is the $\max$ function based on e^* as defined above: the value of a clause c is $e^*(c)$, the product of its literals values, and the value of a formula is given by the maximum value among its clauses. This way, a single clause having value 1 (falsified) results in the whole formula having value 1, and assignments that satisfy all but one clause (a common situation in structured SAT instances) are very far from the global minimum. However, based on our experiments, the $\max$ function appears to not be smooth enough for our purposes, and it does not allow for a sufficiently accurate discrimination among assignments: indeed, all assignments which falsify at least one clause have the same value 1 and are thus indistinguishable. A better choice is the function $\mathbb{S}$ which we define as follows.

Definitionfl, Performance function $\mathbb{S}$. Given a formula in CNF $\varphi = C_1 \wedge \ldots \wedge C_m$ and an assignment e, the performance function $\mathbb{S}$ is defined for a clause C_i as $\mathbb{S}_{C_i}(e) = \prod_{l \in C_i} e(l)$, and for φ as $\mathbb{S}_\varphi(e) = (\sum_{i=1}^{m} \mathbb{S}_{C_i}(e)/m)$.

Note that if φ is satisfiable, then the global minimum of $\mathbb{S}$ is 0 and it is reached on a satisfying assignment to φ. In other words, we replace the $\max$ function by the arithmetic average of the clauses values; as we demonstrate in §5, this definition allows us to achieve satisfactory convergence towards a global minimum in a reasonable time for satisfiable instances.

4.2 The parameter setting

A generic cross-entropy algorithm for optimization involves two main steps:

- Generate a number R of independent and identically distributed samples in the search space according to a specified probability distribution $\mathbf{p}_t$.
- Update the parameters of $\mathbf{p}_t$, based on a certain amount R_b of best performing samples (the *elite samples set*), by means of cross-entropy minimization.

The cross-entropy method creates a sequence of *levels* $\gamma_0, \ldots$ and *parameter matrices* $\mathbf{p}_0, \ldots$ such that the first sequence converges to an optimal performance, while the second one converges to the corresponding optimal parameter matrix. Usually a smoothed updating rule is used, in which the parameter matrix is taken as a linear combination (with *smoothing parameter* α) of the previous matrix and the new one obtained from the elite samples set.

Several variants of the cross-entropy method ([7, 23, 27, 36]) are described in the literature: with or without smoothing, with or without memory (the elite samples set is entirely computed from the current iteration rather than accounting for the previous generations), with fixed or adaptive α, R and R_b. We adopt a memoryless approach, since it is more efficient in this framework. For the same reason, we keep both R and R_b fixed during the algorithm execution. As commonly done for combinatorial optimization problems, we choose a smoothed updating procedure to avoid the degeneration of the sequence of parameter matrices to a $\{0,1\}$-matrix.

Different proofs can be found certifying the theoretical convergence of the algorithms both in the continuous and in the combinatorial setting; given the approach chosen and dealing with a discrete problem, we mainly refer to [23] and [7]. In particular, [7] compares the effects of adopting a constant or adaptive smoothing parameter: an appropriately decreasing sequence of α is *guaranteed to reach an optimal solution*; using a constant but sufficiently small α, the algorithm converges to an optimal solution with probability arbitrarily close to 1. The price is in terms of speed of convergence; this is why, following the considerations in the literature [36] and our own experience, we use a constant smoothing parameter, even though it might lead to non global optima.

4.3 The CROiSSANT algorithm

Algorithm 1 illustrates our application of the cross-entropy method to the satisfiability problem. The algorithm takes as input a CNF formula φ, a coarseness K of the discretization $\mathcal{D}_K$, a number of samples to generate in each iteration R, the performance function $\mathbb{S}$ introduced in §4.1, a size of the elite samples set R_b, a smoothing parameter α, a time limit.

begin
  repeat
    Initialize $\mathbf{p}_0$;
    $t := 1$;
    repeat
      Generate R assignments $e_1, \ldots, e_R$ according to $\mathbf{p}_{t-1}$;
      Evaluate $\mathbb{S}(e_i)$, sort from smallest to largest $\mathbb{S}_{(1)} \leq \ldots \leq \mathbb{S}_{(R)}$;
      Extract the elite samples set $e_{(R-R_b+1)}, \ldots, e_{(R)}$;
      Set $\gamma_t := \mathbb{S}_{(R-R_b+1)}$;
      Calculate $\mathbf{q}_t$ from the elite set using $(*)$;
      Update $\mathbf{p}_t := \alpha\, \mathbf{q}_t + (1-\alpha)\, \mathbf{p}_{t-1}$;
      $t := t + 1$;
    until *a minimum is detected or the time limit is reached* ;
  until *the time limit is reached* ;
  Return information (see §4.4)
end

Algorithm 1: CROiSSANT algorithm

The algorithm follows these steps. It starts by initializing the probability matrix to the uniform distribution over the variables $\mathcal{V}$ and the values $\mathcal{D}_K$, moving then to the main loop. In each iteration t, a number R of i.i.d. sample assignments from $\mathcal{D}_K^n$ is drawn according to the probability matrix $\mathbf{p}_{t-1}$ of the previous iteration. The performance function $\mathbb{S}$ (which gives a measure of the "truth level" of the formula φ) is evaluated on the samples, and the results are sorted in ascending order; the R_b best samples are taken to form the new elite set and the new level γ_t (corresponding to the performance of the worst sample in the elite set) is determined. From this set, for each variable v and value d a probability $\mathbf{q}_{t,v,d}$ is derived according to the following equation:

$$\mathbf{q}_{t,v,d} = \frac{\sum_{z=1}^{R} \mathcal{I}_{\{\mathbb{S}(e_z) \geq \gamma_t\}} \mathcal{I}_{\{e_z(v)=d\}}}{\sum_{z=1}^{R} \mathcal{I}_{\{\mathbb{S}(e_z) \geq \gamma_t\}}} \qquad (*),$$

where the function $\mathcal{I}$ is an *indicator function* (or a *characteristic function*): for a Boolean expression b the value of $\mathcal{I}_{\{b\}}$ is 1 if b is true, 0 otherwise. In other words, the probability $\mathbf{q}_{t,v,d}$ is the

proportion of samples in the elite set for which v has value d (recall definitions from 4.1). Finally, the probability matrix $\mathbf{p}_{t-1}$ is updated by "shifting" it (based on the smoothing parameter α) towards the matrix $\mathbf{q}_t$ generated from the elite set, thus making it biased towards the current most successful samples.

We use a simple criterion in order to detect whether a minimum has been found: the sequence of γ_t steadily decreases (while the probability matrix converges to an optimal one) although it exhibits oscillations due to the stochastic nature of the algorithm. To account for this phaenomenon, we keep track of the current lowest value among the γ_t: if after a certain number of iterations such value has not improved, we determine that a minimum has been reached.

Since, as discussed, we are not guaranteed to converge to a global minimum, the whole process is repeated for a series of runs until the time limit is met.

In the end, the algorithm returns some information, for example the overall best sample obtained throughout the various runs (see §4.4).

4.4 Interaction with SAT solving

As a first attempt to create a new hybrid framework that exploits the features of DPLL learning and stochastic optimization, we have carried out a *feasibility study* by employing a cross-entropy algorithm as a preprocessing step before SAT solving. CROiSSANT performs a preliminary exploration of the search space, storing the best assignment found; this is used to generate a partial Boolean assignment which is given as input to the DPLL solver. Then, at solving time, when branching on a variable v, the search is driven by choosing its polarity according to that partial Boolean assignment (of course if the variable is affected by the assignment).

The reason behind this approach lies in the relationship between the granularity of the search space and the way the Boolean assignments are produced. In §4.1 we introduced the multi-valued model and discussed the extension of the Boolean domain to an arbitrary discrete domain $\mathcal{D}_K$ with K values between 0 and 1. Intuitively, the association of a value with a variable measures how that variable is close to being either true or false: 2 values can distinguish between true and false, 3 values allow "unknown" variables, and so on. An initial exploration of the search space by CROiSSANT can provide the SAT solver with useful information on the likelihood of variables to be either true or false. By analyzing the best sample found, we extract the subset of variables which have value 0 or 1 and use these values to guide the decision phase at solving time.

Notice that we tested one among several ways of interaction between CROiSSANT and a SAT solver. Other approaches, still having cross-entropy minimization as a preprocessing step, might include: employing a partial Boolean assignment also taking into account how many and which variables are involved; deriving from the best sample a vector of probabilities based on which to assign polarities; storing and using the probability matrix from which the best sample was generated; exploiting structural dependencies among clauses.

More complex approaches could be based on an integrated framework, where a cross-entropy algorithm and a SAT solver are run alternately while exchanging information; it would be interesting to test learning techniques based on resolution, in line with [13], where new clauses are added to reshape the space to remove current local minima.

Cross-entropy can be combined with other stochastic optimization methods, such as simulated annealing, swarm and evolutionary algorithms. A promising direction is also to enhance CROiSSANT with local techniques, for example stochastic local search: intuitively, a cross-entropy-based algorithm can perform an initial wide-scale narrowing of the search space, while local search will be applied in a second phase for finding a global minimum in a much smaller area.

We leave exploration of the aforementioned approaches as part of the future work.

4.5 Tuning of the CROiSSANT algorithm

The input of the algorithm consists of a CNF formula φ, a coarseness K of the discretization $\mathcal{D}_K$, a number R of samples to generate in each iteration, a performance function $\mathbb{S}$, a size R_b of the elite samples set, a smoothing parameter α and a time limit.

Since our goal is to use cross-entropy as a fast preprocessing technique, we give CROiSSANT a short timeout compared to the total time dedicated to preprocessing plus solving. The reasons for the choice of a specific $\mathbb{S}$ have been presented in §4.1; in order to set a framework for experimentation we performed some preliminary benchmarking, analyzing the relationships among R, R_b, α, with respect to the timeout and to the size of the search space, dependent on φ and $\mathcal{D}_K$.

Larger values of R allow for deeper exploration at each iteration; on the other hand, the generation of more samples comes at the price of a smaller amount of iterations that can be carried out within a given time limit. For our experimentation we chose a fixed R, but we plan to investigate how to make the parameter adaptive to the complexity of the instance at hand. A similar tradeoff exists for the size of the elite set: values of R_b close to R increase the preciseness of the estimates (almost all samples are considered to update the distribution), while smaller values allow for faster convergence; we use a fixed R_b of 10% of R.

The adoption of a constant α entails an additional tradeoff between accuracy and speed of convergence: a smaller α increases the probability of finding an optimal solution, but at the expense of a greater amount of iterations.

Another parameter was added, to account for the possibility of running into non global minima, as discussed in §4.1. Stopping condition for the inner loop of the algorithm is the detection of a minimum, in terms of a lack of improvement of the current lowest value in the sequence of γ_t for a certain number of iterations; this amount, to which we will refer as *"no improvement threshold"* (NIT), is also given as input. Notice that a good choice for the threshold depends both on R and R_b (more exploration increases the chances of improvement), and on the size and characteristics of the search space.

With regard to the coarseness of the discretization, a greater K entails a more dense space, with advantages in terms of graduality of convergence of the cross-entropy technique (§4.2) and accuracy of the information supplied to the SAT solver (§4.4); nevertheless, a larger space might make convergence slower and contain more non global minima. We tested this tradeoff using different values of K.

An interesting question that arose, while developing our approach, was how likely it would be to come across the same minimum in different runs, and, in that case, how to avoid the issue. We experimented with manipulating the probability matrix in order to direct the search away from the minima already found. Unfortunately, one of the major points of the cross-entropy algorithm, that is being able to treat variables individually, turned out to be a hindrance to our idea: in fact, by modifying the probabilities of the variables, we would end up reducing the probability of generating not only the minima themselves, but also many other unrelated sample assignments. In practice, we achieved the best performance by simply restarting from the uniform distribution: for a given timeout,

taking as input formulae of sufficiently high complexity (so as to have a sufficiently large search space), we noticed that the algorithm tended to visit different areas of the space in each run, leading to different minima.

5. EXPERIMENTAL RESULTS

We implemented our variant of the cross-entropy-based optimization algorithm in a tool named CROiSSANT written in C++. In this section we present the results of executing CROiSSANT as a preprocessor on a collection of benchmarks with different settings of tunable parameters, following the considerations of §4.5.

Since our approach employs the cross-entropy method with the aim of identifying the areas of the search space that are more likely to contain a satisfying assignment, we considered it natural to focus on *satisfiable* benchmarks. We collected satisfiable instances from the SAT Competition 2011 and the SAT Race 2010 and compared the running time of the state-of-the-art DPLL-based SAT solver MiniSat [31] alone with the running time of MiniSat after a preprocessing by CROiSSANT. Given the stochastic nature of our technique, we took into account not only the default deterministic strategy of MiniSat (to always assign negative polarity), but also the randomized strategy that assigns polarity positive or negative at each branching step with equal probability.

Although not part of the initial aims of the research, at a later stage additional experimentation was conducted on *unsatisfiable* instances, to examine the behaviour of CROiSSANT on search spaces where no satisfying assignment could be found. In this case, the set of IBM hardware designs that are used at IBM for internal benchmarking was chosen. As SAT solver, we opted for Mage [37], an IBM internal industrial tool aimed at verifying hardware designs; its particular tuning allows Mage to constantly outperform MiniSat on instances generated from hardware designs. The results[1] show that improvement exists also for unsatisfiable instances, suggesting, perhaps, that if an instance is unsatisfiable, CROiSSANT can "focus" on the subset of variables that led to conflict, enabling the SAT solver to prune the space faster.

5.1 Experimental Data

A set of 305 + 22 satisfiable benchmarks was initially extracted by running MiniSat on the 900 new instances of SAT Competition 2011 and on the 100 instances of SAT Race 2010, setting a time limit of 3h. The benchmarks were run with a total (preprocessing plus solving) time limit of 1h on machines equipped with a Quad-Core AMD Opteron(tm) processor 2344 HE 1000 MHz and 3Gb of RAM memory.

Table 1 presents the results of executing CROiSSANT as a preprocessor to MiniSat in different configurations compared to the results of MiniSat alone; we show the average values obtained over three runs using different seeds[1]. The first and the second configurations in the table consist in running just MiniSat respectively with default and random polarities. The other configurations include preprocessing using 50 as number of samples per iteration R, 0.1 as smoothing factor α and 50 as no improvement threshold NIT, whereas preprocessing time and coarseness of the discretization K (as number of values) are shown; the best overall (partial) assignment, not taking into account variables with values different from 0 and 1, was given to MiniSat to suggest variable polarities during branching. In the "Solved" column, the numbers represent the amount of instances for which MiniSat could find a satisfiable assignment. The "Average" column shows the average time in seconds spent (after the preprocessing phase) by MiniSat on the different families of instances, both solved and non solved; preprocessing time is not included, apart from the few cases when CROiSSANT was able to find a satisfiable assignment itself, and clearly MiniSat was not run.

[1]Data available on request

5.2 Analysis

The experiments give some promising results. We see that in the first configuration (2 minutes preprocessing, $K = 2$) the use of CROiSSANT indeed improves MiniSat performance on all the three classes of the SAT Competition instances. CRoiSSANT is also noticeably superior (both in number of solved benchmarks and average running times) on the crafted class: this mainly contains problems of combinatorial origin, such as graph isomorphism, Green Tao and Van der Waerden numbers, automata synchronization. Instances of this kind tend to show regularities in the solution space: global and non global minima might be close, so that the information provided by CROiSSANT in terms of the assignment with the best performance (that is, closest to a satisfying assignment) would be particularly useful.

The adoption of a random in opposition to a non-random approach in assigning polarities has a major effect on the results over structured and random instances; in particular, the random strategy performs better than all the other techniques on random problems, but systematically worse on structured problems.

Remark 2. In this set of experiments we limited the running time of CROiSSANT, being focused on the total SAT solving time. This implies that more complex instances were executed for longer but fewer iterations, with the risk of less exploration and less accurate information supplied to the SAT solver; the full data[1] shows that the amount of iterations ranges from tens of thousands to just a few (we need also to take into account the fact that a tool more engineered and fine-tuned than CROiSSANT would have achieved a greater level of exploration of the search space in the same preprocessing time). A more appropriate choice from a probabilistic point of view, and which would have likely allowed to obtain even better overall performance, would have been to set a limit on the amount of iterations instead; we leave this investigation as future work.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we described a novel approach to the satisfiability problem based on the cross-entropy method. Our framework extends the binary Boolean model to a multi-valued setting, where variables can be assigned values in a discrete set; we induce a probability space over the enlarged search space and define a performance function that correlates with the likelihood of the input formula to be satisfiable. We implemented our variant of the cross-entropy algorithm in a tool named CROiSSANT, written in C++. As a first step towards the development of a hybrid approach we employ CROiSSANT as a preprocessor to a DPLL-based solver; CROiSSANT is executed with a time limit and the best assignment found is exploited to suggest variables polarities at solving time. We conducted experiments on different sets of benchmarks, using CROiSSANT in combination with the state-of-the-art solver MiniSat; the results show an improvement both in running times and number of solved instances, providing evidence that our framework is a sound basis for the development of a cross-entropy-based SAT solver.

As future work, we plan on one side to improve the performance of CROiSSANT and to develop a deeper integration with DPLL solving; on the other side, we will try to characterize the classes of instances (as well as the structure of the corresponding search spaces) and the areas, such as combinatorics, where the

Table 1: MiniSat on SAT Competition 2011 and SAT Race 2010 benchmarks (average values)

		(1) MiniSat only		(2) MiniSat random pol.		
	#bms	Solved	Average	Solved	Average	
SAT Comp.2011	305	140.0	2126.56	145.7	2074.63	
application	47	41.0	638.70	37.7	778.08	
crafted	58	45.0	971.98	41.7	1226.05	
random	200	54.0	2811.05	66.3	2625.40	
SAT Race 2010	22	20.0	627.17	19.3	756.42	
cryptography	11	11.0	556.39	10.3	704.38	
hardware-verif.	1	1.0	106.45	1.0	12.71	
mixed	7	5.0	1070.14	5.0	1144.97	
software-verif.	3	3.0	26.72	3.0	288.56	

		(3) 2 mins, $K = 2$		(4) 2 mins, $K = 3$		(5) 2 mins, $K = 10$	
	#bms	Solved	Average	Solved	Average	Solved	Average
SAT Comp.2011	305	**143.7**	**2017.71**	139.7	2039.56	143.3	2026.79
application	47	**42.0**	**619.41**	39.7	654.00	40.0	687.45
crafted	58	**47.7**	**798.19**	**48.3**	**794.50**	**50.7**	**769.28**
random	200	**54.0**	**2699.97**	51.7	2726.23	52.7	2725.45
SAT Race 2010	22	19.3	736.80	20.3	624.69	19.3	668.95
cryptography	11	10.3	791.53	10.7	623.13	10.0	687.31
hardware-verif.	1	1.0	107.62	1.0	172.85	1.0	40.50
mixed	7	5.0	1035.19	5.7	942.05	5.3	1000.07
software-verif.	3	3.0	49.57	3.0	40.50	3.0	38.52

cross-entropy method is most effective. A second promising direction is to combine CROiSSANT with local techniques, such as stochastic local search: intuitively, the cross-entropy-based algorithm can perform an initial narrowing of the search space, while local search will be helpful in a later stage for finding a global minimum (a satisfying assignment) in a much smaller space.

7. REFERENCES

[1] G. Alon, D. Kroese, T. Raviv, and R. Rubinstein. Application of the cross-entropy method to buffer allocation problem in simulation-based environment. *Annals of Operations Research*, 2004.

[2] G. Audemard, J. Lagniez, B. Mazure, and L. Sais. Integrating conflict driven clause learning to local search. In *LSCS*, pages 55–68, 2009.

[3] G. Audemard, J. Lagniez, B. Mazure, and L. Sais. Learning in local search. In *ICTAI*, pages 417–424, 2009.

[4] A. Balint, M. Henn, and O. Gableske. A novel approach to combine a SLS- and a DPLL-solver for the satisfiability problem. In *SAT*, pages 284–297, 2009.

[5] H. Chockler, E. Farchi, B. Godlin, and S. Novikov. Cross-entropy based testing. In *FMCAD*, pages 101–108, 2007.

[6] H. Chockler, E. Farchi, B. Godlin, and S. Novikov. Cross-entropy-based replay of concurrent programs. In *FASE*, pages 201–215, 2009.

[7] A. Costa, O. Jones, and D. Kroese. Convergence properties of the cross-entropy method for discrete optimization. *Op. Res. Lett.*, 35(5):573–580, 2007.

[8] J. Crawford. Solving satisfiability problems using a combination of systematic and local search. In *Rutgers University*, 1996.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[10] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of ACM*, 7(3):201–215, 1960.

[11] U. Dubin. The cross-entropy method for combinatorial optimization with applications. Master Thesis, The Technion, 2002.

[12] N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT*, pages 502–518, 2003.

[13] H. Fang and W. Ruml. Complete local search for propositional satisfiability. In *Association for the Advancement of Artificial Intelligence*, pages 161–166, 2004.

[14] L. Fang and M. Hsiao. A new hybrid solution to boost SAT solver performance. In *DATE*, pages 1307–1313, 2007.

[15] B. Ferris and J. Froehlich. Walksat as an informed heuristic to DPLL in SAT solving. Technical report, 2004.

[16] I. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *AAAI*, pages 28–33, 1993.

[17] D. Habet, C. Li, L. Devendeville, and M. Vasquez. A hybrid approach for SAT. In *CP*, pages 172–184, 2002.

[18] E. Hirsch and A. Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Ann. Math. Artif. Intell.*, 43(1):91–111, 2005.

[19] H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *AAAI/IAAI*, pages 661–666, 1999.

[20] H. Hoos and T. Stützle. *Stochastic Local Search. Foundations and Applications*. Morgan Kaufmann/Elsevier, 2004.

[21] N. Jussien and L. Olivier. Local search with constraint propagation and conflict-based heuristics. *Artif. Intell.*, 139(1):21–45, 2002.

[22] J. Keith and D. Kroese. Rare event simulation and combinatorial optimization using cross entropy: sequence alignment by rare event simulation. In *Proc. of the 34th Winter Sim. Conf.: Exploring New Frontiers*, pages 320–327. ACM, 2002.

[23] D. Kroese, T.Taimre, and Z.I.Botev. *Handbook of Monte Carlo Methods*. Probability and Statistics. Wiley, 2011.

[24] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[25] F. Letombe and J. Marques-Silva. Improvements to hybrid incremental SAT algorithms. In *SAT*, pages 168–181, 2008.

[26] P. Manolios and Y. Zhang. Implementing survey propagation on graphics processing units. In *SAT*, pages 311–324, 2006.

[27] L. Margolin. On the convergence of the cross-entropy method. *Annals of Operations Research*, 134(1):215–238, 1975.

[28] L. Margolin. Cross-entropy method for combinatorial optimization. Master Thesis, The Technion, 2002.

[29] B. Mazure, L. Sais, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Ann. Math. Artif. Intell.*, 22(3-4):319–331, 1998.

[30] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *AAAI/IAAI*, pages 321–326, 1997.

[31] MiniSAT Web page. http://minisat.se/.

[32] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535, 2001.

[33] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.

[34] R. Rubinstein. Optimization of computer simulation models with rare events. *European Journal on Operations Research*, 99:89–112, 1997.

[35] R. Rubinstein. The cross-entropy method and rare-events for maximal cut and bipartition problems. *ACM Trans. on Mod. and Comp. Sim.*, 12(1):27–53, 2002.

[36] R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Information Science and Statistics. Springer-Verlag, 2004.

[37] RuleBase PE. www.haifa.il.ibm.com/projects/verification/RB_Homepage.

[38] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. *Artif. Intell.*, 132(2):121–150, 2001.

[39] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *AAAI*, pages 337–343, 1994.

[40] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *IJCAI (1)*, pages 50–54, 1997.

[41] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI*, pages 440–446, 1992.

[42] J. P. M. Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *EPIA*, pages 62–74, 1999.

[43] L. A. Zadeh. Fuzzy logic. *IEEE Computer*, 21(4):83–93, 1988.