# Stochastic Local Search Methods for Dynamic SAT — an Initial Investigation

**Holger H. Hoos** and **Kevin O'Neill** [1]

**Abstract.** We introduce the dynamic SAT problem, a generalisation of the satisfiability problem in propositional logic which allows changes of a problem over time. DynSAT can be seen as a particular form of a dynamic CSP, but considering results and recent success in solving conventional SAT problems, we believe that the conceptual simplicity of SAT will allow us to more easily devise and investigate high-performing algorithms for DynSAT than for dynamic CSPs. In this article, we motivate the DynSAT problem, discuss various formalisations of it, and investigate stochastic local search (SLS) algorithms for solving it. In particular, we apply SLS algorithms which perform well on conventional SAT problems to dynamic problems and we analyse and characterise their performance empirically; this initial investigation indicates that the performance differences between various algorithms of the well-known WalkSAT family of SAT algorithms generally carry over when applied to DynSAT. We also study different generic approaches of solving DynSAT problems using SLS algorithms and investigate their performance differences when applied to different types of DynSAT problems.

## 1 Introduction

An important method for solving hard combinatorial search problems is heuristic repair, whereby a solver generates a complete but suboptimal solution, and then applies local repair techniques to find an optimal solution. This method has been used to solve constraint satisfaction problems as well as hard satisfiability problems. (See [9] and [11] for introductions.)

One reason for the initial excitement surrounding local search and heuristic repair methods was the potential for solving optimization problems which face changes over time. Scheduling problems, for example, face unexpected events which may require schedule revision, and the efficiency of dynamic rescheduling is important for time-critical applications. Minton et. al., in their 1992 paper introducing heuristic repair methods for constraint satisfaction and scheduling problems [9], note that repair-based methods can be used for dynamic rescheduling in a natural manner, while complete backtracking methods are required to throw away any current solution and incrementally build a new solution. This observation leads naturally to the question of whether local search is indeed effective at repairing solutions when problems undergo small changes which invalidate the current solution. In this paper, we attempt to examine this question in more detail in the context of a particularly simple dynamic constraint satisfaction problem, dynamic propositional satisfiability.

Since the introduction of local search methods for general constraint satisfaction problems, methods for solving hard satisfiability (SAT) problems using local search have improved greatly, in part because SAT is a simplified CSP (with only two possible values per variable) which allows the use of highly efficient heuristics (exemplified by the WalkSAT family of algorithms). Such algorithms have successfully handled classical AI problems such as planning many times faster than other algorithms [7]—the existence of efficient polynomial encodings for such problems allows advances in fundamental SAT-solving algorithms to apply to other kinds of problems as well.

Considering these recent successes in solving conventional SAT problems, we believe that the simplicity of SAT may facilitate the design and investigation of high-performing algorithms for dynamic search problems. Some of the problems tackled by fast SAT algorithms, like planning, are often dynamic in nature, and may benefit from the use of highly efficient algorithms for dynamic satisfiability problems. Furthermore, we believe that fundamental principles learned from studying dynamic SAT problems should carry over nicely into other search problems like generalized constraint satisfaction problems.

In this paper we formally introduce the *dynamic satisfiability* problem and perform some simple experiments to see how state-of-the-art SAT-solving algorithms handle dynamic SAT problems. We also consider the fundamental question of whether, when solving series of related SAT problems, it is better to start search with the best solution found for the previous problem or to start search with a random initial assignment.

## 2 The Dynamic Satisfiability Problem

The conventional SAT problem can be generalised in different ways to allow for dynamic changes in the problem over time. One possibility is to start from a given SAT instance and allow clauses to be dynamically added to or retracted from this instance. The motivation behind this definition is that of modelling a system which is subject to different constraints at differents points in time; these constraints could reflect the state of the environment or of a subsystem, or the input by a user who controls the system interactively. This notion of a dynamic SAT problem is captured by the following formal definition:

**Definition 1:** An *instance of the dynamic SAT problem (DynSAT) over a set $V$ of propositional variables* is given by a function $\Phi : N \mapsto CNF(V)$, where $N$ is the set of nonnegative integers, and $CNF(V)$ is the set of all propositional formula in conjunctive normal form which use only the variables in $V$. For technical reasons, we will consider only cases for which the set of all clauses over all time steps is finite, i.e., $\Phi$ mentions only a finite number of clauses.

If a DynSAT instance does not change after a a finite number of time steps, i.e., if $\exists n : \forall m > n : \Phi(m) = \Phi(n)$, we call this instance an *$n$-stage DynSAT instance*. A DynSAT instance is *cyclic with period $\Delta$* if $\forall n : \Phi(n + \Delta) = \Phi(n)$.

The *decision variant of the DynSAT problem* is to determine for a given DynSAT instance $\Phi$ whether $\Phi(n)$ is satisfiable for each time

[1] Computer Science Department, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada, email: {hoos,oneill}@cs.ubc.ca

$n$, i.e., it has a model $M(n)$. If this is the case, $\Phi$ is called satisfiable, otherwise, $\Phi$ is called unsatisfiable. The problem of determining a sequence of models is called the *model tracking variant of the DynSAT problem*.

Another way of defining DynSAT is to use a fixed set of clauses but allow certain propositional variables to be set to true or false at different points in time. The intuition behind this generalisation of conventional SAT is that of certain distinguished propositional variables representing sensor information or user input, while the remaining variables correspond to aspects of the system which are controlled by the DynSAT solver (e.g., actions, in the context of SAT-encoded planning problems). This can be easily formalised in the following way:

**Definition 2:** An *instance of the dynamic SAT problem (DynSAT) over a set $V$ of propositional variables* is given by a CNF formula $F$ over $V$ and a second-order function $\Psi : N \mapsto (V \mapsto \{true, false, free\})$, where $N$ is the set of positive integers. For each time $n$, $\Psi(n)$ determines for each variable appearing in $F$ whether it is fixed to true, fixed to false, or not fixed. The notion of *$n$-stage* and *cyclic DynSAT instances* can be defined exactly analogously as in Definition 1.

The *decision variant of the DynSAT problem* is to determine for a given DynSAT instance $(F, \Psi)$ whether it is satisfiable, i.e., to determine whether for each time $n$, $F$ has a model $M(n)$ such that $M(n)$ assigns true to each variable $v$ for which $\Psi(n)(v) = true$ and false to each variable $v$ for which $\Psi(n)(v) = false$. Analogously to Definition 1, the problem of determining the sequence of models is called the *model tracking variant of the DynSAT problem*.

It is not hard to see that Definitions 1 and 2 are equivalent in the sense that each DynSAT instance according to Definition 1 can always be transformed into an equivalent DynSAT instance according to Definition 2 and vice versa. The proof of this proposition is based on the following two observations:

Given a DynSAT instance $(F, \Psi)$, for each variable $v$ which is fixed at time $n$ we add the unit clause $v$ to $F$ if $\Psi(n)(v) = true$, and we add the unit clause $\neg v$ to $F$ if $\Psi(n)(v) = false$. (If $\Psi(n)(v) = free$, then $v$ is not fixed at time $n$ and no unit constraints need to be added.) This gives a sequence $\Phi$ where $\Phi(n)$ consists of $F$ with necessary unit clauses added. Clearly $\Phi$ is satisfiable if $(F, \Psi)$ is satisfiable.

Conversely, given a DynSAT instance $\Phi$, we let $F$ be the CNF formula consisting of all the clauses mentioned by $\Phi$ (according to the definition, this is a finite structure); we then extend the set of propositional variables by adding an indicator variable $v_i$ for each clause $c_i$ in $F$. Now, another CNF formula $F'$ is obtained by replacing each clause $c_i$ in $F$ by $c_i \vee \neg v_i$. Finally, $\Psi(n)$ is defined such that for the indicator variables, $\Psi(n)(v_i) = true$ if $c_i$ appears in $\Phi(n)$, and $\Psi(n)(v_i) = false$ if $c_i$ does not appear in $\Phi(n)$; for all original problem variables $v$, $\Psi(n)(v) = free$. $(F', \Psi)$ is satisfiable exactly if $\Phi$ is satisfiable.

Both definitions have advantages. Definition 1 is conceptually simpler and a slightly more obvious generalisation from conventional SAT from a theoretical point of view; this makes it slightly better suited for theoretical considerations. Definition 2, on the other hand, reflects actual dynamic systems with sensory information in a more direct way and, as we will see later, facilitates the development of generalisations of conventional local search algorithms for SAT to DynSAT. For the remainder of this paper, we therefore focus on DynSAT problems formalised according to Definition 2.

It should be noted that for practical applications, both types of changes—adding/retracting clauses and fixing/releasing variables—can occur. This situation can either be handled by a formulation allowing for both changes (which can be obtained by combining Definitions 1 and 2) or by encoding one type of changes into the other one based on the observations above. Furthermore, both definitions are slightly more general than suggested by the informal motivation given before: in practical applications (i.e., DynSAT-encoded dynamic problems), we would expect that typically only a distinguished set of clauses or variables would be subject to the dynamic changes, while other clauses or variables represent static properties of the given problem. For the sake of generality and conceptual simplicity, we did not reflect this intuition in our definitions.

DynSAT can be generalised to a dynamic version of MAX-SAT in a straightforward way: Dyn-MAX-SAT instances are DynSAT instances where the objective is to maximise the number of satisfied clauses in each stage of the problem. Thus, Dyn-MAX-SAT instances model problems where assignments which do not satisfy all clauses in a given stage are still of value. Typically, this situation is given if optimisation problems are modelled where some of the clauses represent conditions which are not essential to a solution, but which, when satisfied, increase the value of a solution.

## 3 SLS Algorithms for DynSAT

Stochastic local search is a particularly promising method for solving dynamic satisfiability problems: intuitively, the underlying local search paradigm seems to be well suited for recovering solutions after local changes of the problem occurr. Furthermore, state-of-the-art SLS algorithms show an impressive performance in solving a broad range of conventional, static SAT problems [5] and these algorithms can be easily extended to solve dynamic SAT and dynamic MAX-SAT. One drawback of SLS algorithms compared to systematic search methods is the fact that they are typically incomplete, i.e., they cannot prove the unsatisfiability of a problem instance. However, in practice this is often not problematic, since in many cases, the problem is to find a model (or, in dynamic SAT, a sequence of models), and SLS algorithms have been shown to be competitive with or superior to complete systematic search procedures for a wide range of SAT problems. Furthermore, due to the size and hardness of the problem instances, or tight time-constraints, systematic search procedures often cannot be run to completion, which severely limits the practical relevance of their theoretic completeness. Finally, it has recently been shown that some of the best-performing SLS algorithms currently known are probabilistically approximately complete, i.e., they find an existing model with arbitrarily high probability when given sufficient search time [3]. It is also known that in practice, these algorithms can be easily parallelised with optimal speedup [6].

For solving dynamic SAT problems using SLS algorithms, there are several basic approaches:

1. Solving a DynSAT instance as a series of conventional SAT instances. This method is very generic and allows arbitrary SAT algorithms to be used. However, it does not exploit the fact that the changes from one stage of the dynamic problem to the next are typically rather small and local in nature and thus might require only relatively small repairs to the solution from the last stage. For SLS algorithms with random search initialisation (such as the GSAT [11, 10] or WalkSAT algorithms [8]), this method is equivalent to restarting the search at the beginning of each stage. We therefore refer to this method as *random restart*.
2. Using SLS algorithms for conventional SAT, but after each change, continuing the search from the point in the search trajectory where the change occurred. We call this method *trajectory continuation*. Intuitively, it should be able to recover a solution quickly if only a few search steps are required to repair the clauses which are unsatisfied after the change.

3. Devising specialised SLS algorithms for DynSAT which try to identify promising starting points for recovering a solution after a change has occurred. This approach is a generalisation of trajectory continuation and, in principle, allows us to exploit knowledge on the probability or frequency of changes.
4. Devising specialised SLS algorithms for DynSAT which exploit given or learned knowledge about the dynamics of the problems, but are not just generalisations of trajectory continuation. Such algorithms might, for example, steer the search towards solution which are more robust to change, using statistical information on the probability of specific changes learned during previous runs of the algorithm.

Approaches 1 and 2 have the advantage that existing, highly optimised implementations of state-of-the-art SLS algorithms for SAT, such as various variants of GSAT [10] and WalkSAT [8] algorithms can be used directly or with very minor modifications. This is particularly attractive when assuming an application scenario where the changes occur on the same time-scale as is required for finding or recovering a solution and no information regarding the frequency or probability of certain changes is available. This situation would be given, for instance, in an embedded real-time control system, where the changes reflect sensor information in a highly unpredictable physical or virtual environment. Approaches 3 and 4 require more substantial modifications of existing algorithms or even newly designed algorithms. They seem to be more appropriate for situations where the time constraints are more relaxed, such that additional computation time is available for collecting the information required for reaching more robust solutions. Approach 4 would also be the most promising method if the environment is such that vital aspects of the problem dynamics are either known or can be learned reasonably efficiently [13].

For this initial investigation, we focus on approaches 1 and 2, since as we have seen, approaches 3 and (even to a greater extent) 4, require additional assumptions which complicate the empirical evaluation and restrict its scope. For now, we thus restrict ourselves to the investigation of the following questions:

- Does trajectory continuation (approach 2) work significantly better than random restart (approach 1)?
- Which SLS algorithms for conventional SAT lend themselves best to solving DynSAT problems, using random restart or trajectory continuation?

When using the trajectory continuation variant – as opposed to the random restart variant – of an SLS algorithm to find a model after a change in the problem has occurred, the search cost could be affected in different ways. Ideally, the model of the previous stage is also a model of the new stage, making search unnecessary. (Since our goal here is to test performance of various search strategies in tracking models rather than our good fortune in choosing models that satisfy two adjacent DynSAT stages, all DynSAT instances used in our study have been constructed in a way that this situation never occurs.) A second possibility is that the old model is close to a model of the new stage such that the expected search cost is small compared to solving the new stage from scratch. A third possibility is that the old model could be in an area of the new search space from which finding a solution of the new stage is relatively difficult. In this case, trajectory continuation would show an increased search cost over random restart. A priori, it is not clear which of these effects would dominate in practice. Furthermore, different SLS algorithms for conventional SAT might be differently affected by the position the search is started from after a change occurrs. In the following section, we describe experiments which investigate these questions.

# 4 Empirical Results

For our empirical investigation, we focus on members of the prominent WalkSAT algorithm family. In particular, we use WalkSAT/SKC (the original WalkSAT algorithm) [10], WalkSAT/TABU [8], Novelty$^+$ and and R-Novelty$^+$ [3]. These are among the best-performing SLS algorithms for various types of conventional SAT problems [5]. These algorithms start the search at a randomly chosen truth assignment and in each step select a currently unsatisfied clause which is then forced to become satisfied by selecting one of its literals and flipping the truth value of the corresponding variable (for details on the variable selection heuristics, see [8, 3]).

The DynSAT instances used for our experiments are derived from sets of Random-3-SAT and SAT-encoded Graph Colouring instances taken from the SATLIB Benchmark Suite.[2] These problem types were selected because they have been intensely studied in the SAT community and allow us to study potential differences in algorithmic behaviour for random and more structured problems.

## 4.1 Random-3-SAT Instances

For Random-3-SAT, we developed a series of 10-stage DynSAT instances based on SATLIB test-set uf125-538 (a set of 100 cnf formulae with 125 variables and 538 clauses each). For each conventional SAT instance, a DynSAT instance was obtained by fixing 6 variables at each stage. For the first stage $\Psi(0)$, we specified 6 randomly selected variables $v$, where $v$ was set randomly to $true$ or $false$. For the subsequent stages, $\Psi(i+1)$ included one negated variable from $\Psi(i)$ as well as 5 other randomly chosen variables. (Negating one variable from the previous stage ensured that stage $i$ and stage $i+1$ would not share any models, making some search necessary for every stage.) Using a systematic SAT solver, we checked the satisfiability of each stage thus generated; if a given stage was not satisfiable, more stages meeting the same requirements were generated until a satisfiable one was found.

To evaluate the efficacy of a local search algorithm for DynSAT, we use the RLD-based approach of [4]: for each stage of a DynSAT instance, we measure the run-length (i.e., the number of variable flips needed to reach a satisfying assignment), then sum up the run-lengths over all stages and measure these total run-lengths over multiple tries on the same problem instance in order to obtain run-length distributions (RLDs). Here, each RLD is based on 250 tries per instance; each try had a high cutoff parameter setting of $10^7$ for each stage to ensure a maximal number of successful tries. From these RLDs, we extracted the median search cost per instance and analysed the distribution of this measure over the test-set. We tested the performance of three algorithms: WalkSAT/SKC with noise 0.5, WalkSAT/TABU with tabu-list length 5, and R-Novelty$^+$ with noise 0.7 and walk probability 0.01.[3]

Table 1 shows basic descriptive statistics of the distribution of median search cost over our Random-3-SAT test-set. It shows that the search cost for the variants using trajectory continuation is approximately a factor of 2 lower than for those using random restart. This performance gain can be observed for all three algorithms, indicating that the effectiveness of trajectory continuation does not depend on the search heuristic. Furthermore, it may be noticed that the performance differences between the three WalkSAT variants are qualitatively analogous to those observed for the underlying test-set of conventional SAT instances [5]

[2] SATLIB is a widely used resource for SAT-related research available on the WWW at www.informatik.tu-darmstadt.de/AI/SATLIB.

[3] The parameter settings are the ones which are approximately optimal for the underlying conventional SAT problems [5].

| test-set | mean | v.c. | median | $q_{90}$ | $q_{90}/q_{10}$ |
|---|---|---|---|---|---|
| wsat t.c. | 20145.67 | 1.59 | 10077 | 49475 | 33.18 |
| wsat r.r | 42333.78 | 1.07 | 28311 | 76392 | 5.04 |
| wsat+tabu t.c. | 14894.20 | 1.44 | 7265 | 32353 | 29.20 |
| wsat+tabu r.r | 28456.89 | 0.84 | 19407 | 58160 | 6.50 |
| rnov+ t.c. | 6975.99 | 1.15 | 3800 | 16168 | 28.97 |
| rnov+ r.r | 14252.82 | 0.65 | 11241 | 27494 | 5.09 |

**Table 1.** DynSAT instances based on SATLIB test-set uf125-538, basic descriptive statistics of median search cost per instance for different algorithms using random restart (r.r.) and trajectory continuation (t.c.); v.c. denotes the variation coefficient (stddev/mean) and $q_x$ the $x\%$ percentile.

Interestingly, for trajectory continuation, a much larger variation in search cost across the test-set can be observed than for random restart. A closer look at the underlying data reveals that the effect of trajectory continuation varies significantly between different stages of a problem, such that for the 10-stage problems used here, these differences add up such that for some instances, many of the stages are likely to be solved almost instantly, while for others, all stages require substantial search for t.c. as well as for r.r.

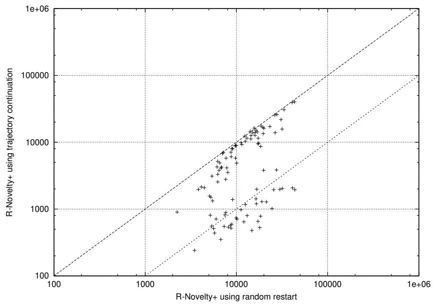


**Figure 1.** Correlation of median search cost for R-Novelty$^{+}$ with t.c. and r.r. across the DynSAT test-set derived from uf125-538; the correlation coefficient is $r = .45$.

This variation can also be seen from Figure 1, showing the search cost per instance for R-Novelty$^{+}$ with trajectory continuation vs. random restart. Note that there are only 2 problem instances for which t.c. performs (minimally) worse than r.r., while for about 1/3 of the instances tested, the search cost for t.c. is more than one order of magnitude lower. The correlation between the search cost for t.c. and r.r. across the test-set is typically rather weak (0.45 for R-Novelty$^{+}$, similar for the other algorithms), while when comparing different algorithms using t.c. or r.r. each, a much stronger correlation is observed (correlation coefficients between 0.8 and 0.95). This confirms that the improvement achieved by using t.c. is orthogonal to the effect of the underlying algorithm and largely independent from the relative hardness of the instance within this test-set.

Overall, these results suggest that trajectory continuation is generally more efficient than random restart when solving DynSAT instances based on Random-3-SAT problems, and that for the underlying SLS algorithm, high performance for static SAT instances translates to good performance on related DynSAT problems.

## 4.2 SAT-encoded Graph Colouring Instances

The previous results naturally lead to the question of whether the results presented thus far depend on properties of the underlying Random-3-SAT instances, or extended to DynSAT instances based on other kinds of SAT problems. Therefore, we conducted a second series of experiments using more structured DynSAT instances derived from SATLIB test-set flat100-239 comprising 100 SAT-encoded graph colouring instances, each with 100 vertices, 239 edges, and a chromatic number of 3. These SAT instances have been studied in [5] and contain 300 propositional variables each. From each of these SAT instances, we derived a 10-stage DynSAT instance, where in each stage $\Psi(i)$ the colour of three vertices is fixed. For the first stage $\Psi(0)$, we randomly chose three vertices, and for each vertex we picked a random colour. For the following stages, $\Psi(i+1)$ includes one vertex from the previous stage, $\Psi(i)$, fixed to a different colour, one vertex from $\Psi(i)$ fixed at the same colour, and one new randomly chosen vertex/colour pair. As for the Random-3-SAT instances, this ensures that a new model be found for the resulting SAT encoding; here, it also guarantees that the stage $\Psi(i+1)$ cannot be solved by simply permuting the colours entailed by of $M(i)$. As before, we made sure that each stage is satsifiable in order to obtain a test-set of 100 satisfiable DynSAT instances.

For this test-set, we measured the performance of two algorithms: WalkSAT/SKC with noise 0.5, and Novelty$^{+}$ with noise 0.6 and walk probability 0.01.[4] For each instance, we measured RLD data from 100 tries, where a high cutoff parameter setting of $10^7$ variable flips (per stage) ensured a maximal number of successful tries.

| test-set | mean | v.c. | median | $q_{90}$ | $q_{90}/q_{10}$ |
|---|---|---|---|---|---|
| wsat t.c. | 1172919.30 | 0.60 | 1053738 | 1968591 | 4.17 |
| wsat r.r | 1187833.22 | 0.62 | 957819 | 2146433 | 4.24 |
| wsat t.c.s.r. | 670571.11 | 0.58 | 566165 | 1112616 | 4.01 |
| nov+ t.c. | 289056.55 | 0.92 | 205854 | 510034 | 5.13 |
| nov+ r.r | 301332.71 | 0.86 | 212009 | 548829 | 5.37 |
| nov+ t.c.s.r | 226391.21 | 0.73 | 168413 | 422093 | 5.60 |

**Table 2.** DynSAT instances based on SATLIB test-set flat100-239, basic descriptive statistics of median search cost per instance for different algorithms using random restart (r.r.), trajectory continuation (t.c.), and trajectory continuation with soft restart (t.c.s.r.).

From the performance data shown in Table 2, it is immediately apparent that for the graph colouring problems, trajectory continuation is not as effective as for the Random-3-SAT instances. This suggests that for these more structured problems, it is much more difficult to reach a model starting from an old model after a change has occurred.

Upon closer examination of the underlying performance data from individual stages of selected problem instances, it becomes clear that for these problems, two contrary effects can be observed: Sometimes, trajectory continuation is as efficient in recovering solutions quickly as for the Random-3-SAT instances. There are other cases, however, for which the model found during the preceeding stage seems to provide an especially bad starting point for the next phase of search, and trajectory continuation gives significantly worse performance than random restart.

This observation leads us to introduce a slight modification of the trajectory continuation approach: the *soft restart* strategy triggers a random restart when for a given number of flips (in our experiments set to 10 times the number of variables in the given problem instance) no improvement in the objective function (i.e., the number of unsatisfied clauses) over the best value encountered since the last restart has been achieved. Intuitively, when combined with trajectory continuation, this strategy should enable the underlying SLS algorithms to recover quickly from bad initial assignments, as well as exploit the full benefit of good initial assignments. The data in Table 2 confirms this intuition and shows that trajectory continuation with soft restart is significantly more efficient than random restart for the graph

[4] As for Random-3-SAT, these parameter settings were selected according to approximately optimal parameter settings for the underlying conventional SAT test-sets, see [5].

colouring instances considered here. However, the performance difference is still much smaller than for the Random-3-SAT instances, suggesting that for more structured DynSAT problems, t.c. is less effective than it is for unstructured, random problem instances.

When analysing the correlation of search cost for the same algorithm using random restart and t.c. with soft restart, resp., we find that the correlation is quite strong ($r = 0.94$ for Novelty$^+$). This reflects the smaller impact of t.c. with soft restart vs. random restart, but also the more uniform structure of the instances imposed by the SAT-encoding of the underlying graph colouring instances, which makes the variation in observed performance difference between the two approaches less extreme than for the Random-3-SAT case.

Overall, these results confirm that trajectory continuation (especially when combined with soft restart) is more efficient than random restart.

## 5 Related Work

The concept of dynamic combinatorial search problems is not a new one. Dechter and Dechter [1] introduce the dynamic constraint satisfaction problem, and describe methods for finding new solutions when CSPs undergo minor changes like the addition of unit constraints to exisiting variables. Their work focuses on the propogation of constraints in constraint networks, and doesn't consider heuristic repair or local search at all.

Verfaillie and Schiex [12] examine ways to reuse solutions to dynamic random CSPs, and introduce an algorithm which combines features of backtracking and heuristic repair techniques. Their algorithm performs well for a range of random problem instances, but it is not clear how its performance would compare to state-of-the-art SLS algorithms for CSPs. Furthermore, they don't consider more structured problems.

Ginsberg, Parkes, and Roy [2] attempt to quantify the notion of a robust SAT model by introducing *supermodels*. Their distinction between the robustness of solutions and algorithms is a useful one, but finding reasonable supermodels which are robust to changes of even several variables is orders of magnitude more difficult than finding a simple model, and thus not useful for the DynSAT algorithms we propose here. We hope future work will be able to further quantify the robustness of SAT models in a way that is easier to approximate, and which is easily amenable to local search methods.

Wallace and Freuder [13] also explore dynamic CSPs and solution stability, considering the case where problem changes are temporary and recurring. In this case, it is possible to learn what the next problem change is likely to be, so Wallace and Freuder define a stable solution as one where variables are assigned values which are not likely to be made unavailable in the next problem change. This idea is useful when problems changes are recurring, but doesn't help us decide when a solution is stable in a more fundamental way, i.e., when a solution is as robust as possible to future problem changes, even if they are are unexpected.

## 6 Conclusions

In this paper, we introduced the dynamic SAT problem (DynSAT), gave two different, but equivalent, definitions for this problem, and presented an initial investigation of stochastic local search algorithms for DynSAT. We characterised several approaches for solving DynSAT problems using stochastic local search, two of which allow existing, powerful SLS algorithms for SAT to be used with little or no modification. We investigated these two approaches, based on random restart and trajectory continuation after each change of the problem, by empirically analysing the performance of several variants of the well-known WalkSAT algorithm for SAT when applied to different types of DynSAT instances derived from established benchmark problems for conventional SAT. Our results indicate that trajectory continuation is considerably more efficient than random restart, particularly for hard, unstructured problems problem instances. We also found that heuristics which improve SLS performance on static instances also help to solve the corresponding DynSAT instances more efficiently.

This work presents only an initial investigation of the DynSAT problem and methods for solving it. Many interesting research issues remain to be explored. One of the most fruitful areas appears to be the study of algorithms for DynSAT which utilise the time between finding an initial solution of the current stage of a problem and the occurrence of the next change to search for solutions which are more robust with respect to problem changes. Moreover, when considering scenarios where the changes can be expected to be of a regular nature, some of the lessons learned for dynamic CSPs can obviously be applied to DynSAT, and combined with heuristic strategies which work well for conventional SAT. Other directions for future research on dynamic SAT include the adaption of systematic, Davis-Putnam like SAT procedures to the dynamic case and the combination of SLS algorithms for DynSAT with polynomial preprocessing techniques which are known to be crucial for solving large and complex conventional SAT problems.

Overall, we believe that DynSAT is an interesting problem which will allow us to extend the knowledge gained from studying algorithms for conventional SAT to dynamic problems. Sharing the same motivation as the more general dynamic CSP problem, dynamic SAT could benefit from the same conceptual simplicity which has facilitated recent successes in SAT-related research.

## REFERENCES


[1] R. Dechter and A. Dechter, 'Belief maintenance in dynamic constraint networks', in *Proc. AAAI-88*, pp. 37–42. MIT Press, (1988).

[2] Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy, 'Supermodels and robustness', in *Proc. AAAI-98*, pp. 334–339, (1998).

[3] H.H. Hoos, 'On the run-time behaviour of stochastic local search algorithms for SAT', in *Proc. AAAI-99*, pp. 661–666. MIT Press, (1999).

[4] H.H. Hoos and T. Stützle, 'Evaluating Las Vegas Algorithms — Pitfalls and Remedies', in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pp. 238–245. Morgan Kaufmann Publishers, San Francisco, CA, (1998).

[5] H.H. Hoos and T. Stützle, 'Local search algorithms for SAT: An empirical evaluation', *to appear: J. Automated Reasoning, special Issue "SAT 2000"*, (1999).

[6] H.H. Hoos and T. Stützle, 'Towards a characterisation of the behaviour of stochastic local search algorithms for sat', *Artificial Intelligence*, **112**, 213–232, (1999).

[7] Henry Kautz and Bart Selman, 'Pushing the envelope: Planning, propositional logic, and stochastic search', in *Proc. AAAI-96*, Portland, Oregon, (1996).

[8] David McAllester, Bart Selman, , and Henry Kautz, 'Evidence for invariants in local search', in *Proceedings of IJCAI-97*, (1997).

[9] Steven Minton et al., 'Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems', *Artificial Intelligence*, **58**, 161–205, (1992).

[10] Bart Selman, Henry A. Kautz, and Bram Cohen, 'Noise Strategies for Improving Local Search', in *Proceedings of AAAI'94*, pp. 337–343. MIT Press, (1994).

[11] Bart Selman, Hector J. Levesque, and David G. Mitchell, 'A new method for solving hard satisfiability problems', in *Proc. AAAI-92*, pp. 440–446, San Jose, California, (1992).

[12] G. Verfaillie and T. Schiex, 'Solution reuse in dynamic constraint satisfaction problems', in *Proc. AAAI-94*, pp. 307–312. MIT Press, (1994).

[13] Richard J. Wallace and Eugene C. Freuder, 'Stable solutions for dynamic constraint satisfaction problems', in *Principles and Practice of Constraint Programming*, pp. 447–461, Pisa, Italy, (1998).