algorithms

MDPI

Article

A Dynamic Clause Specific Initial Weight Assignment for Solving Satisfiability Problems Using Local Search

Abdelraouf Ishtaiwi [1,*], Feda Alshahwan [2], Naser Jamal [3], Wael Hadi [3] and Muhammad AbuArqoub [4]

[1] Information and Communication Technology, University of Petra, Amman 1196, Jordan
[2] The Public Authority for Applied Education and Training, Shuwaikh 70654, Kuwait; fa.alshahwan@paaet.edu.kw
[3] Department of Computer Information Systems, University of Petra, Amman 1196, Jordan; njamal@uop.edu.jo (N.J.); whadi@uop.edu.jo (W.H.)
[4] Computer Science, University of Petra, Amman 1196, Jordan; Abu-Arqoub@uop.edu.jo
* Correspondence: aishtaiwi@uop.edu.jo; Tel.: +962-796405480

Abstract: For decades, the use of weights has proven its superior ability to improve dynamic local search weighting algorithms' overall performance. This paper proposes a new mechanism where the initial clause's weights are dynamically allocated based on the problem's structure. The new mechanism starts by examining each clause in terms of its size and the extent of its link, and its proximity to other clauses. Based on our examination, we categorized the clauses into four categories: (1) clauses small in size and linked with a small neighborhood, (2) clauses small in size and linked with a large neighborhood, (3) clauses large in size and linked with a small neighborhood, and (4) clauses large in size and linked with a large neighborhood. Then, the initial weights are dynamically allocated according to each clause category. To examine the efficacy of the dynamic initial weight assignment, we conducted an extensive study of our new technique on many problems. The study concluded that the dynamic allocation of initial weights contributes significantly to improving the search process's performance and quality. To further investigate the new mechanism's effect, we compared the new mechanism with the state-of-the-art algorithms belonging to the same family in terms of using weights, and it was clear that the new mechanism outperformed the state-of-the-art clause weighting algorithms. We also show that the new mechanism could be generalized with minor changes to be utilized within the general-purpose stochastic local search state-of-the-art weighting algorithms.

Keywords: artificial intelligence; dynamic local search algorithms; optimization; planning

Citation: Ishtaiwi, A.; Alshahwan, F.; Jamal, N.; Hadi, W.; AbuArqoub, M. A Dynamic Clause Specific Initial Weight Assignment for Solving Satisfiability Problems Using Local Search. *Algorithms* **2021**, *14*, 12. https://doi.org/10.3390/a14010012

Received: 7 December 2020
Accepted: 26 December 2020
Published: 5 January 2021

1. Introduction

Many real-world problems are of NP-complete (nondeterministic polynomial) class. In contrast, systematic approaches (based on explicit rules) require exponential time rather than a polynomial-time to reach a solution. A problem is NP-complete if there is an NP (np) problem, and all NP problems are reduced to (np) in polynomial time. Therefore, NP-complete problems are of great interest in computer science because finding an algorithm that could solve any NP-complete problem implies that there is a general-purpose algorithm that could solve all NP-complete problems of the same class.

Metaheuristics methods are widely employed to handle NP-complete problems with attributes such as uncertainty and vast search spaces. The metaheuristic approach's main idea is to find a solution with an acceptable quality rather than finding an optimal one. Another characteristic of metaheuristics methods is that they are incomplete such that they do not guarantee to find a solution if one exists. In literature, the metaheuristics methods are classified into global-based approaches (also known as population-based approaches) and local-based approaches. Nature designs generally inspire the global-based metaheuristics to solve optimization problems [1–3], such as the Ant Colony Optimization [4], which

was mainly inspired by ants' actions and behaviors searching for food. Moreover, the Genetic algorithms (GA metaheuristics) [5] are other examples that were primarily derived from the natural selection process that is based on past knowledge that is stored in many chromosomes (mainly derived from Charles Darwin theory [6]) and used to make future decisions on the selection process. In recent years, Swarm Intelligence (SI) global-based metaheuristics have shown significant improvements in the overall process of handling optimization problems, mainly when applied for solving real-world industrial optimization problems such as scheduling [7,8], vehicle routing [9]. The underlying general framework in swarm intelligence depends on a decision-making process involving a broad range of information gathered from multiple dimensions and sub-domains of the search space. For instance, when searching for a solution using a SI technique, two factors are considered to evaluate the current search stage progress: (1) the improvement of the current search stage cost and (2) the impact of the current search stage on the overall improvement of the objective cost function of the search process.

The other metaheuristic category proposed almost four decades ago and received a high level of researcher's attention is local search-based metaheuristics (LS). LS techniques share most of the characteristics of the global-based metaheuristics such as incompleteness, and the ability to handle NP-complete optimization problems when covering all possible solution within the search space is not feasible due to a limitation in computational resources. Our focus on this research is local search metaheuristics, specifically the general-purpose weighting dynamic local search techniques for SAT.

In 1971, the Boolean satisfiability problem (SAT) (also known as propositional satisfiability problem, or SAT for short) was proven to be an NP-complete as in Reference [10], which motivated researchers around the world to come up with general mechanisms for solving SAT. However, no such mechanisms exist and engineering a general mechanism to tackle the SAT problem still an open area for research and experimentation. Stochastic local search (SLS) techniques efficiently handle problems encoded into SAT. Therefore, researchers over the decades are motivated to enhance the performance of SLS techniques.

SAT problems are of great interest to researchers as a wide range of real-world problems are encoded into SAT in areas like artificial intelligence such as planning, scheduling, and other areas such as cryptography or bioinformatics, computer theory, engineering, operation research, physics, to mention a few. However, the nature of SLS techniques is that they are incomplete (no guarantee to explore the search space [11] fully); therefore, they can not prove satisfiability. Nevertheless, almost four decades' effort pays off as there are many algorithms capable of handling the SAT problem with hundreds of thousands of variables, which was satisfying when formulating and handling the real-world problem into SAT. However, when converted into the SAT problem, the modern real-world problems require solving the SAT problem that is much larger, in much less time than the size of the SAT problems and the solving speed a decade ago. These requirements of the modern real-world problems imposed even more challenges into handling the SAT problem. Therefore, researchers in both tracks: the exact search area (also known as complete or systematic search), and the stochastic local search (also known as optimization search, or incomplete search [12]), are trying to investigate more efficient algorithms to handle the size (where the hardware is limited) and the speed in which the solution to the SAT problem is satisfying w.r.t both the quality of the solution and the speed of reaching it.

SLS techniques, no matter how sophisticated they are, are either descending directly from or based indirectly on the work of Lin and Kernighan on Local Search (LS), that is, hill-climbing [13]. The hill-climbing procedure iteratively improves the objective function, where all the violated constraints (clauses in SAT encoded problems in CNF (Conjunctive normal form)) are measured by searching for moves that reduce the total number of violated constraints. For example, while searching for a solution, any local modification (moves) that make a major reduction of the objective function is taken.

The work of Lin and Kernighan showed that hill-climbing successfully reaches solutions when handling problems such as travelling salesman problem and the n-queen

problem [13] However, the hill-climbing algorithm is faced with encountering what is known as local optima, in which there are no moves in the current solution stage that improve the objective function cost, that is, getting stuck in local minima. In Reference [14], another algorithm that could improve the hill-climbing algorithm was proposed that employs a random walk known as WalkSAT ([15]).

The random walk addition reduces the level of the hill-climbing algorithm greediness by increasing the level of the randomness through explicitly picking at complete random order any unsatisfied clause (violated constraint) within the current solution stage and flip one of its literals (assign a different value from the variable domain in constraints satisfaction problem (CSP) [16]). Another algorithm based on the hill-climbing local search algorithm is the greedy local search (GSAT) [17] algorithm, which generally follows the hill-climbing procedure such that the moves must improve the current objective function cost, with the addition that equal-cost moves are permitted with a probability parameter p (also known as noise as in simulated annealing [18–21]).

The GSAT algorithm could prevent the search process from getting stuck in plateaus, which is another drawback of the hill-climbing algorithm. Thus, the GSAT algorithm could continue searching even though the objective function cost is not improving. In other words, GSAT improved the performance of the hill-climbing algorithm and could successfully cope with the hill-climbing level of greediness through the sideway utilization. However, adding the noise to allow the sideway moves does not prevent GSAT from getting stuck in local minima; therefore, GSAT+Restart (GSAT reset), GSAT+Random Walk, and GSAT+Weights were proposed in the year 1993 [21]. The three variants' experimental results show that major improvements are gained such that larger problems could be handled, and the speed of reaching feasible solutions was improved. In 1993 also, Paul Morris [22] proposed a similar clause weighting technique known as Breakout algorithm The GSAT+Weight and the Breakout algorithm became a building block for many clause weighting-based algorithms in SLS due to their efficacy in handling enormous problems and their simplicity w.r.t implementation. Therefore, a large number of weighting SLS algorithms were proposed in the last three decades.

The breakout algorithm and the GSAT+Weight both assign weights (penalties) to all the clauses (constraints) in the formula $\mathcal{F}$. Initially, the weights are set to 1, then searching for cost improving moves begins such that the objective function (which measures the total number of unsatisfied clauses or violated constraints) is improving, that is, the number of violations is reducing. When the search could not find any cost improving moves, the weights of all violated constraints (unsatisfied clauses) are increased additively as in Reference [23] or multiplicatively as in Reference [24]. This paradigm of using weights while searching for a solution is a milestone in local search and artificial intelligence in general due to its implications of handling real-world problems.

In recent years, handling real-world problems through stochastic local search (SLS) techniques has evolved dramatically, which has lead to the production of more sophisticated general-purpose stochastic local search algorithms. The evolution of SLS varies from proposing simple yet efficient approaches such as tabu search [25,26], and simulated annealing [18–20]), to more sophisticated approaches such as scatter search [27], evolutionary algorithms [28], dynamic local search [23,24,29,30], and hybrid SLS [31,32].

In this paper, we are interested in dynamic local search weighting algorithms, as this family of algorithms forms one of the four general-purpose mechanisms for solving SAT [33]. More specifically, we are interested in investigating the setting of the initial weights, as, to the best of our knowledge, the first and only try in this direction was made in Reference [34] where a new algorithm is proposed by Ishtaiwi et al., known as DDFW(Divide and distribute fixed weights), in which the initial weights were set to 8, instead of 1. Therefore, a crucial question appears here: is there a way to allocate the initial value of the weights so that it optimizes the search process's performance and reduces the time it takes to reach a solution with higher efficiency than the currently existing algorithms.

In the next Section 2, we discuss the preliminaries that are used within the paper. In Section 3, we discuss a general overview of the similarities and differences among the state-of-the-art clause weighting SLS solvers. In Section 3.1, we broadly investigate the initial weight setting impact on the state-of-the-art clause weighting SLS. Then, in Section 4, we propose a new novel mechanism to assign the clauses' weights dynamically. We show that changing the current method of weight initialization is dramatically improved by our novel method. Then, in Section 5, we illustrate the improvements experimentally gains by our approach, and then we analyze the results, and then, the conclusion is drawn in Section 6.

2. Preliminaries

For the purpose of conducting our current research we consider the SAT problem encoded in Conjunctive Normal Form (CNF). CNF formulas ($\mathcal{F}$) consist of a conjunction $\bigwedge$ of *clauses*, and each clause is a disjunct $\bigvee$ of literals where each literal is a propositional variable or its negations. The task is to find boolean values (true, or false) assignment for all the literals that can evaluate the formula ($\mathcal{F}$) to true. Applying systematic search techniques to this task is computationally costly as they require exponential run time because they cover every possible assignment of literals. Therefore, SLS is the method of choice when handling the large SAT problems [35], as they consider only a subset of the search space (known as candidate solution), which is evaluated in polynomial time. A candidate solution to an SAT encoded problem in CNF formula ($\mathcal{F}$) is an assignment of all literals in CNF such that the objective cost function $\neq 0$ (or incomplete solution). A solution to an SAT problem is reached when the objective function of the given CNF ($\mathcal{F}$) is $= 0$.

Given a CNF ($\mathcal{F}$) formula, a clause (cl) $\in$ ($\mathcal{F}$) can be either *satisfied* or *unsatisfied*. A clause is satisfied if $\exists$ a true literal $\in cl$, otherwise the clause cl is unsatisfied. The objective function of the CNF formula ($\mathcal{F}$) is $= 0$ (where all the clauses are satisfied), iff there is an assignment of all literals $\in$ ($\mathcal{F}$), that is, all the clauses are satisfied. We denote the size of a clause sCl as the number of all literals $\in cl$. Furthermore, if a same signed literal l occur in cl_i and clause cl_j where $i \neq j$ then the clauses cl_i and cl_j are neighbors. $sNcl$ denotes the number of all neighboring clauses of the clause cl.

3. Clause Weight in SLS

As we discussed previously, the two main building blocks for almost all weighting local search algorithms are the GSAT+Weight and the Breakout algorithm. However, the GSAT+Weight algorithm was designed to handle the boolean satisfiability (SAT) problems, which is the subject of this research as we only consider the SAT-based problems. Therefore, our focus is on the state-of-the-art clause weighting algorithms driven directly as an improvement of the GSAT+Weight algorithm, or the algorithms that conceptually employ the underlying technique of the GSAT+Weight mechanism.

The stat-of-the-art solvers that use different variants of the weighting strategy of the GSAT+Weight mechanism are known as Dynamic Local Search weighting solvers. There are many examples of the state-of-the-art dynamic SLS solvers that utilize weights, for example, mulLWD+WR [36], DCCAlm [37], CSCCSat [38], BalancedZ [39], $Score_2SAT$ [40], CCAnrSim [41], CryptoMiniSAT [42], Sparrow [43], MapleCOMSPS_LRB_VISDIS [42], and previously DLM [30], SAPS [24], PAWS [23] and DDFW [34].

The underlying technology designed in GSAT is that in each step of the search, a move (m) is taking if m reduces the total number of unsatisfied clauses. As we previously discussed in Section 2, a clause cl consists of the disjunct of some literals or their negation (propositional variable); if cl is (unsatisfied), then none of its literals is true. A move is to change the value of a literal within cl (known as the flip) from true to false or vice versa. A decision of which literal within cl must be taken based on an evaluation function. Some algorithms consider how many clauses will become unsatisfied by flipping a literal as

in (e.g., probSAT algorithm [44]). However, in general, most of the state-of-the-art SLS algorithms evaluate a the cost of move m as follows:

$$score(m) = Makes(m) - Breaks(m), \quad (1)$$

where the ($Makes(m)$) is the number of currently unsatisfied clauses that will become satisfied by the move m, and ($Breaks(m)$) is the number of the clauses that are currently satisfied and would become false by the move m. The Score is the change of the objective function during the current stage of the search process if the move m is performed.

A score(m) is either a positive number, which means the move (m) will satisfy a maximum number of clauses if taken. A negative score of the move (m) means (m) will make more unsatisfied clauses than satisfied ones if taken. In some cases, the score(m) is equal to zero, which means the move (m) will cause the search to move entirely in a plateau (also known as sideway moves). The best case scenario is that the score(m) is always a positive value, but it is not always the case, and the search will face a situation where the score(m) is either zero or a negative number. Handling the three possible scenarios of the value of the score(m) is one of the imperative characteristics that differentiate solvers from each other, for example. The breakout algorithm would not permit moves with score zero or negative scored moves (highest level of greediness). Another example is the WalkSAT, where moves are taken, regardless of their score, with probability p, when the score(m) is equal to zero or a negative value, for all possible moves in the current position during the search. However, sideways moves, where $score() = 0$ are taken with probability p (currently known as the noise probability) in most modern general-purpose state-of-the-art SLS solvers.

SLS techniques follow the same basic general process when applied for solving a CNF formula $\mathcal{F}$. They start by randomly assigning random binary values to all the literals in $\mathcal{F}$. Then, iteratively, search for ways to improve the current objective function by finding cost improving moves through the employment of different techniques and heuristics. One of which is the clause weighting technique. A standard fundamental technology shared among the dynamic local search weighting solvers is the use of weights (penalties) such that the search can take non-improving moves. Taking non-improving moves play a significant role in the diversification of the search process. Moreover, it helps in either escaping from or preventing the search from encountering local minima (where the objective function $\neq 0$, while there is no cost improving moves could be taken).

The clause weighting SLS algorithms initially assign a unit one of weight (also known as penalties) for each clause in $\mathcal{F}$ (e.g., References [23,24,29,30,43]). Then, weights are maintained during the search such that when there is no cost improving moves in the current search stage, weights are modified additively as in Reference [23] or multiplicatively as in Reference [24]. Thus, the score(m) is calculated as follows:

$$score(m) = WeightMakes(m) - WeightBreaks(m) \quad (2)$$

where the ($WeightMakes(m)$) is the sum of weights of the currently unsatisfied clauses that will become satisfied, and ($WeightBreaks(m)$) is the sum of weights of the clauses that are currently satisfied and would become false by the move m.

3.1. Assessment of the Initial Weight Distribution and Its Impact

To accurately assess the efficacy and the impact of the assignment of the initial weight on the overall performance of a given SLS clause weighting algorithm, we conducted experiments on a wide range of the SAT-based CNF problems obtained from both the SATLIB (dimacs cnf) [45] and the SAT competition benchmarks [46] random track benchmarks. We selected the problems set based on two conditions: first, all the problems are hard to solve, where the ratio of variables-to-clauses is 4.2 (aka, the phase transition region). Second, the problems are of three different sizes, small (e.g., BMS-499), medium (e.g., 4blocks), and large (e.g., bw_large.d). To conduct our experiment, we used PAWS as the solving

mechanism to gather the weights' statistics at the beginning and during the search. PAWS uses a simple yet efficient additive weighting mechanism for adjusting the weights. It starts by assigning a unit one of weight for each clause in the problem, then iteratively increases the weights of the unsatisfied clauses by one and then, periodically, smooths the weights to prevent unlimited weights growth. We designed the experimental investigation as follow: firstly, we recorded, for each problem, the total number of clauses nCl, and the neighborhood information, (as previously discussed in Section 2, a neighboring area of clause cl consist of all the clauses that share at least one literal $\in$ clause cl), such that, we recorded the minimum neighborhood size $MinN$, the maximum neighborhood size $MaxN$, the average neighborhood size $AvgN$. Also, we calculated the total number of smallest clauses cl (the size of a clause sCl is the number of literals within the clause cl) that have the smallest neighborhood size $sCL\&sN$, and the smallest clauses with the largest neighborhood size $sCL\&lN$, and the largest clauses with the smallest neighborhood size $lCl\&sN$ and finally, the largest clauses with the largest neighborhood size $lCl\&lN$. Our initial observation from the data that are showed in Table 1 reveals that the highest percentage of the clauses are of small size, more specifically 93%, and 99% of which have small neighborhood size, except for the logistics problem where the percentage of small clauses is 91%.

Table 1. General structural statistics of each problem in our problem set.

Problems	nCl	MinN	MaxN	AvgN	sCl&sN	sCl&lN	lCl&sN	lCl&lN
4blocks	47,820	3	740	433	26,670	21,137	0	13
ais12	5666	12	143	95	197	3672	11	12
BMS-499	278	7	23	14	88	190	0	0
bw_large.c	50,457	5	465	69	38,780	11,636	23	18
bw_large.d	131,973	5	741	86	111,741	20,178	32	22
CBS	403	8	33	21	171	23	0	0
f800-hard	3440	9	37	22	1511	1929	0	0
f1600-hard	6880	8	41	22	3067	3813	0	0
f2000	8500	7	41	22	3797	4703	0	0
fla-barthel	1720	10	39	22	724	996	0	0
flat200-har	2237	3	20	13	641	1596	0	0
g125.17	66,272	17	149	125	31,042	35,105	0	125
g125.18	70,163	18	149	125	32,868	37,170	0	125
huge	7054	5	171	63	3000	4037	7	10
logistics.d	21,991	5	235	26	11,317	8824	1325	525
uf400-hard	1700	9	38	22	768	932	0	0
unif-k5-v200	4223	214	321	269	2081	2142	0	0
unif-k7-v98	8603	1957	2327	2158	4179	4424	0	0
unif-k7-v102	8955	2000	2328	2158	4466	4489	0	0

Based on the above observations, the SAT-based CNF clauses are categorized into the following four categories:

1. a small clause in a small neighborhood
2. a small clause in a large neighborhood
3. a large clause in a small neighborhood
4. a large clause in a large neighborhood

The above four categories are general to any SAT-based CNF problems. However, the fact that some problems clauses are of fixed size, like the 3-CNF problem, therefore, we have came to a conjecture that the first two categories are imperative factors while engineering a general-purpose SLS clause weighting solver. Thus, we ran a further investigation of our problem sets, focusing on the number of clauses weights gain while searching for solutions, focusing on the first two clauses categories' statistics. Out of no surprise, the results, as shown in Figure 1, support our conjecture such that the small clauses, regardless of the size of their neighborhood, were more prone to the fluctuation between satisfied and unsatisfied status.

On the other hand, clauses of considerable neighborhood size were harder to become unsatisfied for two reasons: first, the possibility of a clause to contain a true literal is high. Second, the score of the move (m), where m is to flip a literal within a large unsatisfied clause, is high. Furthermore, Figure 2 illustrates the weights of the small clauses compared to their neighborhood size. It is clearly shown that clauses with smaller neighborhood gained more weight until the solver reached the solution. Also, Figure 2 show that small clauses, even if they have a large neighborhood, generally gained more weights compared to large clauses, which gained a relatively smaller amount of weight until the solver flipped one of their literals.

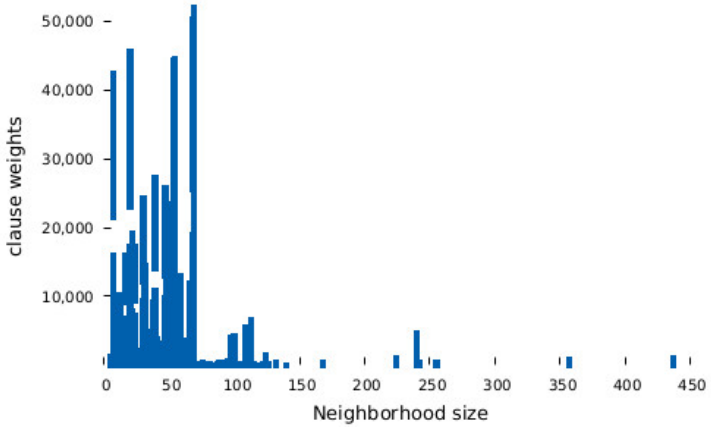

Figure 1. Weight distribution of the clauses of the bw_large problem until a solution is reached.

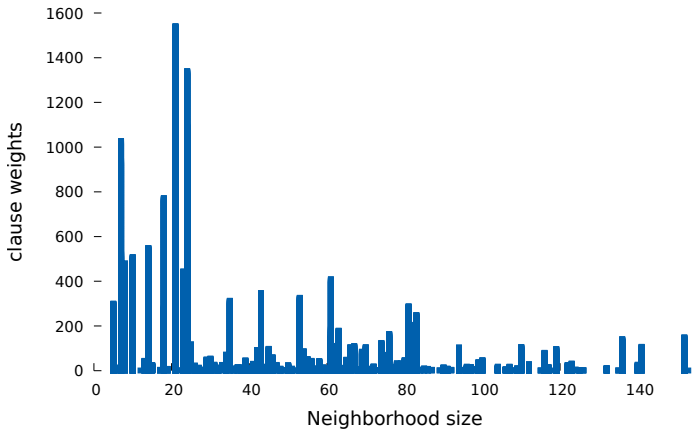

Figure 2. The weight distribution of the small sized clauses with small neighborhood size of the bw_large problem.

The above clause weight assessment indicates that one of the reasons for the fluctuation status of the small clauses (from satisfied to unsatisfied and vice versa) is because of the initial weight assignment where the clause weighting SLS solvers initially treats the clauses equally (assigning a weight of 1 to all clauses), regardless of their size or the size of their neighborhood. Thus, the gap between the weights of the clauses in the weights' initial assignment is considerable.

To overcome this challenge, the clause weighting SLS solvers adjusts the weights of the clauses multiplicatively (e.g., ESG [47], SDF [48], SAPS) or, in most solvers, additively (e.g., PAWS, Sparrow, BalancedZ, YalSAT...etc.). In the next section, we discuss our novel mechanism in assigning the initial clauses weights dynamically, contributing to filling the clauses' initial gap, which appears due to the initial equal assignment of clauses weights.

4. The Dynamic Initial Weights Assignment in SLS

In the previous section, we discussed the negative effect of the equal assignment of the clauses' initial weights on SLS solvers' overall performance. Our experiments show that the small size clauses take a long time to build up weights to become attractive for the search to satisfy them, especially if their neighborhood's size is large. Secondly, the clauses' size plays a vital role when calculating the cost function, such that the large clauses are more likely to break more clauses than small clauses if they become unsatisfied. Thus, small clauses remain in their condition (whether satisfied or not satisfied) for longer than necessary time. These factors contribute to prolonging the search period in general. Therefore, we started thinking about developing a mechanism that would fill the gap between weights, which is often considerable, so that the time spent until a small clause build weights is much less.

Our newly innovative algorithm is divided into two main parts. The first is to analyze a set of information from the problem structure that relies on collecting data before starting the search. The other part is to dynamically allocate initial weights for the clauses, based on the outcome from part one. Table 2 illustrates the problem-specific structural information, as discussed previously. Our novel mechanism starts with the information-gathering based on observations one and two in Section 3.1. More specifically, we count the total number of clauses (#MinClsS) of the minimum-sized clause (MinClS) and the total number of clauses with maximum-sized (MaxClsS) of the maximum-sized clause (MaxClS). Moreover, we calculate the average clauses size (AvgS) and the total number of average-sized clauses (#AvgS). Using these measures, we designed our mechanism to assign the initial weights as in Algorithm 1. The lines from 2 to 16 of Algorithm 1 show the initial weights allocation based on the clause (Cl) size. If the size of (Cl) is less than or equal to the arithmetic average of the sizes of the clauses $(AvgClS)$ in the problem (which means that the (Cl) is of small size), if it is, we check whether (Cl) belongs to a small neighborhood $(size(Neighbor(Cl) <= AvgN)$, if so, we assign to the clause (Cl) an initial weight equal to the neighborhood size $size(Neighbor(Cl))$ of the clause Cl. If the size of the clause's neighborhood is large $(size(Neighbor(Cl) > AvgN)$, we allocate (Cl) initial weight with the equivalent of the arithmetic average clauses' sizes $(AvgClS)$. For the clauses of large size $(size(Cl) > Avg(ClS)$, we check whether the neighborhood size associated with (Cl) is small $(Neighbor(Cl) <= AvgN)$; in this case, we allocate the (Cl) weight to equal the arithmetic average of the clauses' sizes $(AvgClS)$. Lastly, if the clause is large $(size(Cl) > AvgClS)$ and located in a large neighborhood $(Neighbor(Cl) > AvgN)$, we assign the initial weight value of (Cl) equal to the clause's size $size(Cl)$.

Table 2. Clause specific structural statistics.

Problems	nCls	MinClS	#MinClsS	MaxClS	#MaxClsS	AvgS	#AvgS
4blocks	47,820	2	9898	33	12	2	9896
ais12	5666	2	4191	12	12	2	4189
BMS-499	278	3	278	3	0	3	278
bw-large.c	50,457	2	37,493	16	17	2	37,491
bw-large.d	131,973	2	102,580	20	25	2	102,578
CBS	403	3	403	3	0	3	403
f800-hard	3440	3	3440	3	0	3	3440
f1600-hard	6880	3	6880	3	6880	3	6880
f2000	8500	3	8500	3	8500	3	8500
fla-barthel	1720	3	1720	3	1720	3	1720
flat200-har	2237	2	2037	3	200	2	2037
g125.17	66,272	2	66,147	17	125	2	66,145
g125.18	70,163	2	70,038	18	125	2	70,036
huge	7054	2	5682	10	5	2	5681
logistics.d	21,991	2	19,647	16	7	2	19,645
uf400-hard	1700	3	1700	3	1700	3	1700
unif-k5-v200	4223	5	4223	5	4223	5	4223
unif-k7-v98	8603	7	8603	7	8603	7	8603
unif-k7-v102	8955	7	8955	7	8955	7	8955

Algorithm 1 DDFW($\mathcal{F}$) + DynamicInitWeight.

randomly instantiate each literal in $\mathcal{F}$;
if $size(Cl) <= AvgClS$ **then**
  if $size(Neighbor(Cl)) <= AvgN$ **then**
    set the $initWeight(Cl)$ to $size(Neighbor(Cl))$;
  else
    set the $initWeight(Cl)$ to $AvgClS$;
  end if
else
  if $size(Cl) > AvgClS$ **then**
    if $size(Neighbor(Cl)) <= AvgN$ **then**
      set the $initWeight(Cl)$ to $AvgClsS$;
    else
      set the $initWeight(Cl)$ to $size(Cl)$;
    end if
  end if
end if
for each clause $c_a \in \mathcal{F}$ **do**
  let the weight w_a of each clause $c_a = InitWeight(c_a)$;
end for
set the minimum m to the number of false clauses $c_f \in F$;
set counter i to zero and boolean b to false;
while solution is not found and not timeout **do**
  calculate the list $\mathcal{L}$ of literals causing the greatest reduction in weighted cost Δw when flipped;
  if ($\Delta w < 0$) or ($\Delta w = 0$ and probability $\leq 15\%$) **then**
    randomly flip a literal in $\mathcal{L}$;
  else
    for each false clause $c_f \in \mathcal{F}$ **do**
      select a satisfied same sign neighboring clause c_n with maximum weight w_n;
      if $w_n < 2$ **then**
        randomly select a clause c_n with weight $w_n \geq 2$;
      end if
      if $w_n > 2$ **then**
        transfer a weight of two from c_n to c_f;
      else
        transfer a weight of one from c_n to c_f;
      end if
    end for
  end if
end while

5. Experimental Results and Analyses

To investigate the new mechanism, we designed the experiments to cover a broad range of data sets within the domain of SAT CNF-encoded problems. Thus, we categorized the problem sets according to the following factors: First, all the problem sets have been proven to be solvable, which were obtained from the DIMACS problems [45]. Secondly, according to their hardness level, we classified the problems sets into two sub-categories, easy to solve problems and hard to solve problems (according to the phase transition [49]).

Then we obtained the problem sets accordingly, based on their size. The resulting problem data sets include four sub-categories as follows:

1. small problems that are hard to solve, i.e., flat200-hard
2. small problems that are easy to solve, i.e., BMS-499

3. large problems that are hard to solve, i.e., bw_large.d
4. large problems that are easy to solve, i.e., g125.18

Table 3 shows the results for the most studied SAT-encoded problems (e.g., the blocks world problem set (bw) [24], logistics planning instances, flat graph coloring (flat), uniform random 3-SAT instance (uf) and all-interval-series problem (ais)). These problems are relatively small in size, so we have also included large problem sets such as bw_large.d and the graph coloring (g125.17), where both problems are very hard to solve. Furthermore, we have included uniformed 5-SAT, and uniform 7-SAT problem sets to widen our experimental investigation. For testing the novel dynamic initial weight assignment mechanism, we directly built the mechanism into DDFW (as shown in Algorithm 1) to gain the benefits of adaptive parameters; hence, we spent a shorter time on the implementation and directly ran the experiments without needing to tune the DDFW process. DDFW was one of the first algorithms to classify clauses into two groups, satisfied clauses, and unsatisfied clauses. DDFW algorithm relies on transferring weights between the two groups of clauses by searching for a neighboring satisfied clause and transferring weight from it to an unsatisfied clause. For these properties, we built our mechanism in the DDFW algorithm to take advantage of the process of dividing the clauses into two groups, as it is compatible with our innovative mechanism in terms of relying on searching in neighboring clauses to reach the satisfied clauses.

Table 3. Results for the DIMACS problems set.

Problems	ddfw	Paws	Yalsat	probSAT	ddfw+dynIniWeight	Score2SAT	Ratio
4blocks.cn	8.71	2.67	1.67	0.28	0.45	0.51	19
ais12.cnf	0.65	0.10	1.66	7.10	0.09	**0.02**	7.2
BMS-499	0.17	0.04	**0.00**	0.06	**0.00**	**0.00**	≈20
bw_large.c	0.00	0.63	5.81	1.99	**0.00**	0.25	n/a
bw_large.d	0.19	1.22	9.22	7.71	**0.09**	1.7	2.1
CBS	0.04	0.10	0.04	**0.03**	0.00	0.00	≈4
f800-hard	0.56	0.46	1.47	0.47	**0.13**	2.94	4.3
f1600-hard	6.15	5.02	5.18	2.74	**0.52**	3.79	11.8
f2000	3.92	87.80	0.56	**0.42**	0.95	6.45	4.1
fla-barthe	0.23	0.12	0.41	0.10	**0.07**	0.09	3.3
flat200-har	0.30	0.83	8.72	1.82	**0.03**	timed out	10
g125.17.cn	0.55	**0.02**	8.75	timed out	0.09	3.57	6.1
g125.18.cn	0.02	0.02	0.02	11.67	**0.01**	0.45	2
huge.cnf	0.17	0.02	0.01	0.00	0.00	0.00	≈17
logistics.	0.00	7.25	0.52	0.07	**0.00**	0.03	n/a
uf400-hard	3.13	0.72	9.01	2.12	0.73	**0.41**	4.3
unif-k5-v200	3.34	21.30	4.37	2.20	**0.65**	1.28	5.1
unif-k7-v98	20.30	26.50	32.55	18.66	**4.56**	6.76	4.5
unif-k7-v102	90.25	81.30	110.51	85.46	**42.15**	63.51	2.2

We began by testing the performance of the dynamic initial weight assignment mechanism. To do so, we compared the original DDFW + dynamic initial weight assignment with other state-of-the-art solvers, namely, paws, probSAT, yalSAT, and Score2SAT. Figure 3 shows the algorithms' CPU runtime when applied for solving the problem sets. It was clear that DDFW + the dynamic initial weight assignment performed significantly better than DDFW as the ratio indicates (where the ratio in Table 3 is the performance of ddfw+dynInitWeight gains over ddfw, for example if the ratio is 5 then ddfw+dynInitWeight performed five times better than ddfw). We conducted all the experiments using Ubuntu 18.04 operating systems, an 8th generation I5 with four cores, each of which speed is 2.5 GHz, and 16 GB of memory.

The results show that DDFW+Dynamic initial weight assignment is significantly enhanced when compared to DDFW and PAWS as it performed better in all the problem sets. Especially on the problems bw_large.d, f800-hard, f1600-hard, flat200-hard, the g125.17, and the

uf400-hard, where the performance was an order of magnitude better. Also, DDFW+Dynamic initial weight assignment performed four times better than DDFW on the unified k5-v200 and k7-v98 and twice better on the k7-v102 problem. Moreover, DDFW+Dynamic initial weight assignment, when compared to other solvers, performed better in 50% of the overall problem sets and performed similarly with the other solvers on the other problem sets. Our dynamic mechanism's significant performance enhancement could be noticed on large problems such as the g125.17 and problems known to be hardest to solve, such as the f1600-hard, f800-hard, the uf400-hard, the bw_large.d and the uniform problems k5 and k7. Which support our preliminary assessment such that small clauses with either, small or large neighborhood are hard to be satisfied, and take a longer time to build weights. So when they are dynamically assigned the initial weight as per our mechanism, the performance is enhanced.

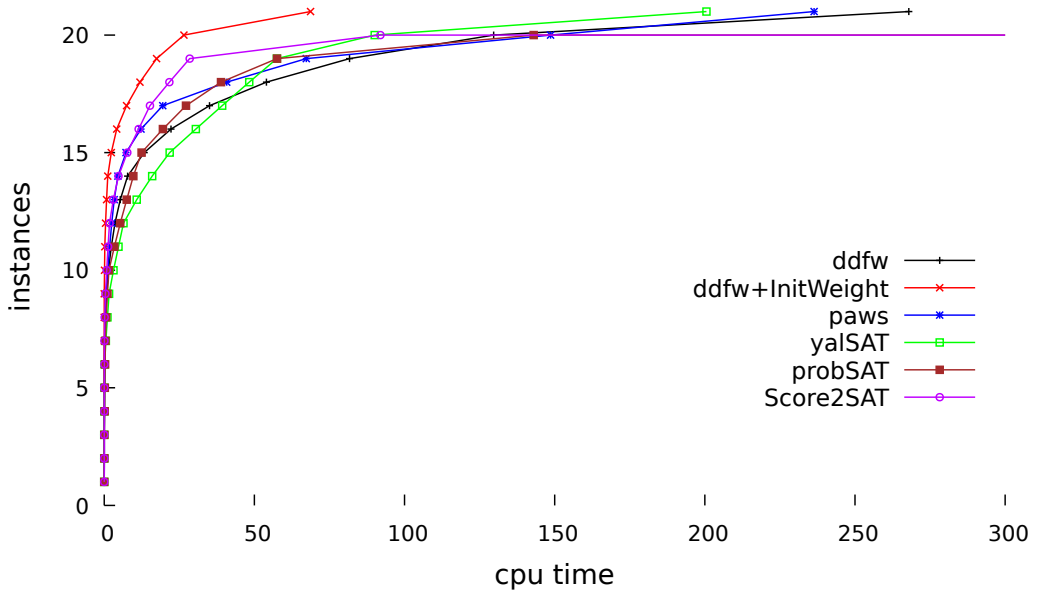

Figure 3. The figure shows the ddfw+initWeight algorithm performance in comparison with other state-of-the-art algorithms.

6. Conclusions

We conclude from the presented research that assigning dynamically the initial weights is significant regardless of the problem's size. Also, the mechanism's effectiveness and impact vary from one problem to another, as the problems in which the number of small clauses is enormous and the size of neighboring neighborhoods is minimal are affected positively by our presented mechanism. Simultaneously, the problems are affected less positively when having in their structure a more significant proportion of clauses of different or large sizes and the sizes of large adjacent neighborhoods. Furthermore, we would like to emphasize that our research was not to develop a new general-purpose clause weighting SLS but rather to look into a dynamic mechanism that could fill the gap between the size of the clauses and the neighborhood sizes within the problem structure. As a result of our research, we conclude that a general-purpose clause weighting SLS is developed, known as DDFW+Dynamic Initial weight assignment, that is very competitive to the state-of-the-art SLS, with a novel mechanism that dynamically assigns the clauses weights.

Our research could be extended to be employed by non-weightings SLS solvers such that the division of the clauses and their neighborhood is utilized during the initialization phase prior to the search process. Furthermore, as the proposed mechanism within the paper is limited to handling problems that have complete solutions (satisfiable), a possible and logical extension of our work is to apply the mechanism to handling MAX SAT problems.

Author Contributions: Conceptualization, A.I.; methodology, A.I.; software, A.I. and W.H.; resources, A.I. and M.A.; data curation, F.A. and W.H.; writing—original draft preparation, A.I.; writing—review and editing, A.I., W.H., M.A., F.A. and N.J.; visualization, A.I. and N.J. All authors have read and agreed to the published version of the manuscript.

Funding: The research has been partially funded by the deanship of the scientific research at petra university.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data.

References

1. Bianchi, L.; Dorigo, M.; Gambardella, L.M.; Gutjahr, W.J. A Survey on Metaheuristics for Stochastic Combinatorial Optimization. *Nat. Comput. Int. J.* **2009**, *8*, 239–287. [CrossRef]
2. Slowik, A.; Kwasnicka, H. Nature Inspired Methods and Their Industry Applications-Swarm Intelligence Algorithms. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1004–1015. [CrossRef]
3. Umamaheswari, H.A.K. A bio-inspired swarm intelligence technique for social aware cognitive radio handovers. *Comput. Electr. Eng.* **2018**, 925–937. [CrossRef]
4. Dorigo, M.; Stützle, T. *Ant Colony Optimization*; Bradford Company: Holland, MI, USA, 2004.
5. Eiben, A.E.; Raué, P.E.; Ruttkay, Z. Genetic Algorithms with Multi-Parent Recombination. In Proceedings of the International Conference on Evolutionary Computation, The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, Erusalem, Israel, 9–14 October 1994; Springer: Berlin/Heidelberg, Germany, 1994; pp. 78–87.
6. Patanella, D. Darwin's Theory of Natural Selection. In *Encyclopedia of Child Behavior and Development*; Goldstein, S., Naglieri, J.A., Eds.; Springer: Boston, MA, USA, 2011; pp. 461–463. [CrossRef]
7. Effatparvar, M.; Aghayi, S.; Asadzadeh, V.; Dashti, Y. Swarm Intelligence Algorithm for Job Scheduling in Computational Grid. In Proceedings of the 2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), Bangkok, Thailand, 25–27 January 2016; pp. 315–317. [CrossRef]
8. Dulebenets, M.A. Application of Evolutionary Computation for Berth Scheduling at Marine Container Terminals: Parameter Tuning Versus Parameter Control. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 25–37. [CrossRef]
9. Pasha, J.; Dulebenets, M.A.; Kavoosi, M.; Abioye, O.F.; Wang, H.; Guo, W. An Optimization Model and Solution Algorithms for the Vehicle Routing Problem With a "Factory-in-a-Box". *IEEE Access* **2020**, *8*, 134743–134763. [CrossRef]
10. Cook, S.A. The Complexity of Theorem-proving Procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; ACM: New York, NY, USA, 1971; pp. 151–158. [CrossRef]
11. Hoos, H.H.; Stützle, T. Local Search Algorithms for SAT: An Empirical Evaluation. *J. Autom. Reason.* **2000**, *24*, 421–481. [CrossRef]
12. Kautz, H.A.; Sabharwal, A.; Selman, B. Incomplete Algorithms. In *Handbook of Satisfiability*; Biere, A., Heule, M., van Maaren, H., Walsh, T., Eds.; Frontiers in Artificial Intelligence and Applications; IOS Press: Amsterdam, The Netherland, 2009; Volume 185, pp. 185–203. [CrossRef]
13. Lin, S.; Kernighan, B.W. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Oper. Res.* **1973**, *21*, 498–516. [CrossRef]
14. Minton, S.; Johnston, M.D.; Philips, A.B.; Laird, P. Solving Large-scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method. In Proceedings of the Eighth National Conference on Artificial Intelligence, Boston, MA, USA, 29 July–3 August 1990; Volume 1, pp. 17–24.
15. Selman, B.; Kautz, H.A.; Cohen, B. Noise Strategies for Improving Local Search. In Proceedings of the Twelfth National Conference on Artificial Intelligence, American Association for Artificial Intelligence, Seattle, WA, USA, 31 July–4 August 1994; Volume 1, pp. 337–343.
16. Dechter, R. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artif. Intell.* **1990**, 273–312. [CrossRef]
17. Selman, B.; Levesque, H.; Mitchell, D. A New Method for Solving Hard Satisfiability Problems. In Proceedings of the 10th AAAI, San Jose, CA, USA, 12–16 July 1992; pp. 440–446.
18. Johnson, D.S.; Aragon, C.R.; McGeoch, L.A.; Schevon, C. Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Oper. Res.* **1991**, *39*, 37–406. [CrossRef]
19. Černý, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Optim. Theory Appl.* **1985**, *45*, 41–51. [CrossRef]
20. Kirkpatrick, S.; Gelatt, C.; Vecchi, M. Optimization by Simulated Annealing. In *Readings in Computer Vision*; Fischler, M.A., Firschein, O., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 1987; pp. 606–615. [CrossRef]
21. Selman, B.; Kautz, H. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In Proceedings of the 13th International Joint Conference on Artificial Intelligence, San Mateo, CA, USA, 1–3 June 1993; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1993; Volume 1, pp. 290–295.

22. Morris, P. The Breakout Method for Escaping from Local Minima. In Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington, DC, USA, 11–15 July 1993; pp. 40–45.
23. Thornton, J.; Pham, D.N.; Bain, S.; Ferreira, V. Additive versus Multiplicative Clause Weighting for SAT. In Proceedings of the 19th National Conference on Artificial Intelligence, San Jose, CA, USA, 25–29 July 2004; pp. 191–196.
24. Hutter, F.; Tompkins, D.A.D.; Hoos, H.H. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. In *Principles and Practice of Constraint Programming—CP 2002*; Van Hentenryck, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 233–248.
25. Glover, F. Tabu Search—Part I. *ORSA J. Comput.* **1989**, *1*, 190–206. [CrossRef]
26. Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **1986**, *13*, 533–549. [CrossRef]
27. Glover, F. Heuristics for Integer Programming Using Surrogate Constraints. *Decis. Sci.* **1977**, *8*, 156–166. [CrossRef]
28. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1989.
29. Amin, S.; Fernandez-Villacanas, J.L. Dynamic local search. In Proceedings of the Second International Conference On Genetic Algorithms in Engineering Systems, Innovations and Applications, Glasgow, UK, 2–4 September 1997; pp. 129–132.
30. Wu, Z.; Wah, B.W. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, Austin, TX, USA, 30 July–3 August 2000; pp. 310–315.
31. Feo, T.A.; Resende, M.G.C. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Oper. Res. Lett.* **1989**, *8*, 67–71. [CrossRef]
32. Lourenço, H.R.; Martin, O.C.; Stützle, T. Iterated Local Search: Framework and Applications. In *Handbook of Metaheuristics*; Springer: Boston, MA, USA, 2010; pp. 363–397. [CrossRef]
33. KhudaBukhsh, A.R.; Xu, L.; Hoos, H.H.; Leyton-Brown, K. SATenstein: Automatically Building Local Search SAT Solvers from Components. In Proceedings of the 21st International Jont Conference on Artifical Intelligence, San Francisco, CA, USA, 11–17 July 2009; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2009; pp. 517–524.
34. Ishtaiwi, A.; Thornton, J.; Sattar, A.; Pham, D.N. Neighbourhood Clause Weight Redistribution in Local Search for SAT. In *Principles and Practice of Constraint Programming—CP 2005*; van Beek, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 772–776.
35. Hoos, H.H.; Stützle, T. Stochastic Local Search Algorithms: An Overview. In *Springer Handbook of Computational Intelligence*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 1085–1105. [CrossRef]
36. Ishtaiwi, A.; Issa, G.; Hadi, W.; Nawaf, A. Weight Resets in Local Search for SAT. *Int. J. Mach. Learn. Comput.* **2019**, *9*, 874–879. [CrossRef]
37. Luo, C.; Cai, S.; Wu, W.; Su, K. Double Configuration Checking in Stochastic Local Search for Satisfiability. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, Quebec City, QC, Canada, 27–31 July 2014; pp. 2703–2709.
38. Luo, C.; Cai, S.; Su, K.; Wu, W. Clause States Based Configuration Checking in Local Search for Satisfiability. *IEEE Trans. Cybern.* **2015**, *45*, 1014–1027. [CrossRef] [PubMed]
39. Sinz, C.; Egly, U. (Eds.) *Theory and Applications of Satisfiability Testing—SAT 2014—17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, 14–17 July 2014, Proceedings*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8561. [CrossRef]
40. Gaspers, S.; Walsh, T. (Eds.) *Theory and Applications of Satisfiability Testing—SAT 2017—20th International Conference, Melbourne, VIC, Australia, 28 August–1 September 2017, Proceedings*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10491. [CrossRef]
41. Janota, M.; Lynce, I. (Eds.) Theory and Applications of Satisfiability Testing—SAT 2019. In Proceedings of the 22nd International Conference, SAT, Lisbon, Portugal, 9–12 July 2019; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11628. [CrossRef]
42. Pulina, L.; Seidl, M. (Eds.) Theory and Applications of Satisfiability Testing—SAT 2020. In Proceedings of the 23rd International Conference, Alghero, Italy, 3–10 July 2020; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2020; Volume 12178. [CrossRef]
43. Balint, A.; Fröhlich, A. Improving Stochastic Local Search for SAT with a New Probability Distribution. In Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing, Edinburgh, UK, 11–14 July 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 10–15. [CrossRef]
44. Balint, A.; Schöning, U. Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break. In Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing, Trento, Italy, 17–20 June 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 1–29. [CrossRef]
45. Hoos, H.H.; Stützle, T. *SATLIB: An Online Resource for Research on SAT*; IOS Press: Amsterdam, The Netherlands, 2000; pp. 283–292.
46. Beyersdorff, O.; Wintersteiger, C.M. (Eds.) Theory and Applications of Satisfiability Testing—SAT 2018. In Proceedings of the 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, 9–12 July 2018; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10929. [CrossRef]
47. Schuurmans, D.; Southey, F.; Holte, R.C. The Exponentiated Subgradient Algorithm for Heuristic Boolean Programming. In Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, WA, USA, 4–10 August 2001; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2001; Volume 1, pp. 334–341.

48. Schuurmans, D.; Southey, F. Local Search Characteristics of Incomplete SAT Procedures. *Artif. Intell.* **2001**, *132*, 121–150. [CrossRef]
49. Gent, I.P.; Walsh, T. The SAT Phase Transition. In Proceedings of the ECAI-94, Amsterdam, The Netherlands, 8–12 August 1994; pp. 105–109.