Cooperative Parallel SAT Local Search with Path Relinking

Padraigh Jarvis[1] and Alejandro Arbelaez[2]

[1] United Technologies Research Centre
JarvisPa@utrc.utc.com
[2] School of Computer Science & Information Technology
Insight Centre for Data Analytics
University College Cork, Ireland
a.arbelaez@cs.ucc.ie

Abstract. In this paper, we propose the use of path relinking to improve the performance of parallel portfolio-based local search solvers for the Boolean Satisfiability problem. In the portfolio-based framework several algorithms explore the search space in parallel, either independently or cooperatively with some communication between the solvers. Path relinking is a method to maintain an appropriate balance between diversification and intensification (and explore paths that aggregate elite solutions) to properly craft a new assignment for the variables to restart from. We present an empirical study that suggest that path relinking outperforms a set of well-known parallel portfolio-based local search algorithms with and without cooperation.

Keywords: SAT · Parallel Local Search

1 Introduction

The propositional satisfiability problem (SAT) is a fundamental problem in computer science with important applications ranging from bioinformatics [19] to planning [23] and scheduling [22]. The SAT problem consists in determining whether a given Boolean formula $\mathcal{F}$ is satisfiable or not. This formula is usually represented using the *Conjunctive Normal Form* (CNF) as follows: $\mathcal{F} = \bigwedge_i \bigvee_j l_{ij}$, where each l_{ij} represents a literal (a propositional variable or its negation) and the disjunctions $\bigvee_j l_{ij}$ are the clauses in $\mathcal{F}$. A k-SAT problem indicates that $\mathcal{F}$ contains k literals per clause, for instance a 3-SAT formula can be represented as follows:

$$\mathcal{F} = (v_{11} \vee v_{12} \vee v_{13}) \wedge (v_{21} \vee v_{22} \vee v_{23}) \ldots (v_{n1} \vee v_{n2} \vee v_{n3})$$

In the weighted MaxSAT problem, clauses are associated with a positive weight and the problem consists in minimizing the cost, i.e., the sum of weights of unsatisfied clauses. The weighted partial MaxSAT problem consists in finding a solution (or an assignment for the variables) that minimizes cost while satisfying a given subset of clauses (i.e., hard clauses).

Complete parallel solvers for the SAT problem have received significant attention recently, these solvers can be divided into two categories the classical divide-and-conquer approach [14] and the parallel portfolio approach [7, 1]. The first one typically divides the search space into several sub-spaces, and the second one lets algorithms compete and cooperate to solve a given problem instance.

In this paper, we focus our attention in cooperative parallel local search solvers for the SAT and Weighted Partial MaxSAT problems. In our settings, each member of the portfolio shares its best assignment for the variables. At each restart point, instead of classically generating a random assignment to start with, the portfolio aggregates the shared knowledge to carefully craft a new starting point.

This paper is organized as follows. Section 2 presents key concepts of local search, including a description of a set of well-known variable selection methods to tackle SAT and MaxSAT problems. Section 3 provides general concepts about parallel portfolios of local search algorithms. Section 4 describes our new cooperative policies using path relinking. Section 5 evaluates our new cooperative policies and Section 6 presents concluding remarks and areas of future work.

2 Local Search for SAT and MaxSAT

Algorithm 1 describes the general schema of the local search procedure for the SAT problem. It starts with a random assignment for the variables in the formula F (*initial-solution* line 2). The key point of the local search procedure is depicted in lines 3-9 where the algorithm flips the most appropriate variable until a certain stopping condition is met, e.g., a given number of flips is reached (Max-Flips) or after a given timeout. After this procedure the algorithm restart itself with a new fresh random assignment for the variables.

Algorithm 1 Local Search (CNF formula F, Max-Flips, Max-Tries)

```
1: for try := 1 to Max-Tries do
2:    A := initial-solution(F)
3:    for flip := 1 to Max-Flips do
4:       if  A satisfies F  then
5:          return A
6:       end if
7:       x := select-variable(A)
8:       A := A with x flipped
9:    end for
10: end for
11: return 'No solution found'
```

As one may expect, a critical part of the algorithm is the variable selection function (line 7 *select-variable*), which indicates the next variable to be flipped in the current iteration of the algorithm. Currently, nearly all variable selection

algorithms are variations of the GSAT [18] and WalkSAT [17] algorithms originally proposed for the SAT problem. These two algorithms attempt to select the variable with the highest *score*.

$$score(x) = make(x) - break(x)$$

Intuitively, *make(x)* indicates the total number of clauses that are currently unsatisfied but become satisfied after flipping x. Similarly, *break(x)* indicates the total number of clauses that are currently satisfied but become unsatisfied after flipping x. Taking this into account, local search algorithms tend to select variables with the minimum *score*, flipping those variables would most likely increase the chances of obtaining the optimal assignment for the variables. In the following, we describe seven well-known variable selection algorithms for the SAT and MaxSAT problems.

- *WalkSAT* [17] uniformly at random selects an unsatisfied clause c. Then, with a probability *wp* selects a random variable from c and with probability *1-wp* identifies the most suitable variable in c.
- *AdaptNovelty+* (*AN+*) [8] uses an adaptive mechanism to properly self-tune the noise parameter (wp) of WalkSAT algorithms (e.g., *Novelty+*). *AdaptNovelty+* introduces a new parameter ϕ to control the value of *wp*. *wp* is initially set to 0 and updated when search stagnation is observed, i.e., $wp = wp + (1 + wp) \times \phi$. Additionally, whenever an improvement is observed *wp* is decreased, i.e., $wp = wp - wp \times \phi/2$. The authors define search stagnation as a stage when no improvement has been observed in the objective function for a given number of iterations.
- *G2WSAT* (G2) [11] introduces the concept of promising decreasing variable. Broadly speaking, a variable is decreasing if flipping it improves the objective function (i.e., total number of (weighted) violated clauses).
- *Adaptive G2WSAT* (AG2) [11] aims to integrate an adaptive noise mechanism into the G2WSAT algorithm.
- *PAWS* (Pure Additive Weighting Scheme) [20] assigns a weight penalty to each clause, with those that go unsolved having their weight penalty changed. This solver includes a chance to make a flip that will result in a lateral movement in satisfiability and a variable to determine how often the weights of clauses are changed.
- *Dist* [5] proposes a variable selection scheme based on hard and soft clauses. *Dist* initially, maintains a list of hard-decreasing variables (i.e., a set of promising decreasing variables of hard clauses), and the algorithm defines a hard and a soft score for the variables in the problem. Furthermore, the authors propose to on-the-fly adjust the weight of hard clauses. This way, *Dist* bias the variable selection process towards improving the score of hard clauses with the set of hard-decreasing variables.
- *CCLS* [13] maintains a list of candidate variables CCMPVars (Configuration Checking and Make Positive), each variable x in the list has a make(x) $>$ 0 and the age of x is smaller than at least one of its neighbour variables

(i.e., a variable sharing at least one clause). In the diversification phase with probability p performs a random walk step; otherwise, with a probability *1-p* in the intensification phase the algorithm selects the variable with the greatest score in CCMPVars. However, if CCMPVars is empty the algorithm performs a random walk. *CCEHC* [12] extends *CCLS* to prioritize the search towards variables involved in violated hard clauses.

3 Parallel Local Search

In this paper, we use the traditional parallel portfolio framework by executing several algorithms in parallel (or different copies of the same one with different random seeds). Therefore, each algorithm independently executes a sequential restart-based local search algorithm and we periodically restart the algorithms to aggregate the common knowledge of the portfolio.

$$M = \begin{pmatrix} X_{11} & X_{12} & \dots & X_{1n} \\ X_{21} & X_{22} & \dots & X_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{c1} & X_{c2} & \dots & X_{cn} \end{pmatrix}$$

Fig. 1. Pool of elite solutions.

In our parallel algorithm we maintain a pool of elite solutions. In this context, each algorithm in the portfolio shares the best solution observed so far in a shared pool M (see Figure 1). Where n indicates the total number of variables of the problem and c indicates the number of local search algorithms in the portfolio. In the following we are associating local search algorithms and processing cores. Each element X_{ji} in the pool denotes the i^{th} variable of the best solution found so far by the j^{th} core.

The initial restarting solution of the algorithms in the portfolio is determined by the cooperation protocol and is a composition of the solutions in the pool. Therefore, maintaining an appropriate balance between diversification and intensification of the solutions in the pool is an important step in the proposed cooperative framework. We remark that we use a random solution for the first start and the cooperative framework afterwards.

Recently, [2] proposed seven cooperative algorithms for parallel SAT solving. These strategies range from a voting mechanism, where each algorithm in the portfolio suggest a value for each variable, to probabilistic constructions. This way, the *variable-initialization* function (line 2, Algorithm 1) uses cooperation (after the second restart) in lieu of random values for the variables.

Prob uses a probability function based in the number of occurrences of variables with positive and negative values. *PNorm* normalizes the probability function with the quality of the solutions (i.e., number of unsatisfied clauses), therefore, values involved in better truth assignments are most likely to be used in

the future. Complete details about these two popular cooperative techniques are available in [2].

Other work in the area includes PGSAT [15], a parallel version of the GSAT algorithm. The entire set of variables is randomly divided into τ subsets and allocated to different processors. In this way at each iteration, if no global solution has been obtained, the i^{th} processor uses the GSAT score function to select and flip the best variable for the i^{th} subset. Another contribution to this parallel architecture is described in [16] where the authors aim to combine PGSAT and random walk. Thus at each iteration, the algorithm performs a random walk step with a certain probability wp, that is, a random variable from an unsatisfied clause is flipped. Otherwise, PGSAT is used to flip τ variables in parallel at a cost of reconciling partial configurations to test if a solution is found.

4 Path Relinking

Path relinking [6] is a popular technique to generate new solutions by exploring paths that connect elite solutions. To generate the new solution (i.e., line 2 in Algorithm 1), an initial solution and a guiding solution are selected from the pool to represent the starting and the ending points of the path.

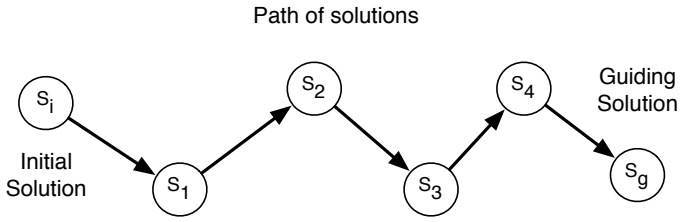


Fig. 2. Path relinking.

In this paper, we use path relinking to generate new starting solutions for the algorithms in the portfolio. Figure 2 depicts the process of generating the path of solutions. We select the initial (s_i) and guiding (s_g) solutions, and then the path relinking algorithm generates the intermediate solutions by replacing values for the variables in s_i with values from s_g. In the context of this paper, the path of solutions regulates the intensification/diversification trade-off. The first neighbour solution of s_i denotes, at least, one change in the initial solution and n-1 chances in the guiding solution.[3] Ideally, in order to balance the diversification/intensification trade-off, the new generated solution should be in middle between the s_i and s_g.

Algorithm 2 shows our path relinking algorithm to generate a new starting solution. Let s_i and s_g denote the initial and guiding solutions and $pd \in [0, 1]$

[3] n denotes the number of variables in the problem.

denotes the probability of using the initial or guiding solution for each variable in the problem. In particular, we explore the following four alternatives to define s_i and s_g:

- *best2rand (b2r)*: s_i represents the best solution in M and s_g is randomly selected from M;
- *cbest2rand (cb2r)*: s_i represents the best solution obtained so far for the processing unit that is currently seeking a new starting solution and s_g is randomly selected from M;
- *best2cbest (b2cb)*: s_i represents the best solution in M and s_g represents the best solution obtained so far for the processing unit that is currently seeking a new starting solution;
- *best2sbest (b2sb)*: s_i represents the best solution in M and s_g represents the second best solution available in M.

The path relinking algorithm uses pd to balance the diversification vs. intensification trade-off dilemma, a value close to 1 (resp. 0) favours s_g (resp. s_i). Therefore, pd=0.5 is a reasonable value for a proper intensification/diversification balance of the solutions. Furthermore, *best2rand* and *cbest2rand* provide further diversification benefits as the method randomizes the selection of the solutions in the pool. Certainly, biasing the search towards s_i or s_g might improve performance for specific problem families. However, without explicit knowledge of the target instances we recommend pd=0.5.

Algorithm 2 Path-relinking(S_i, S_g, pd)

```
1: s := s_i
2: for i := 1 to |s| do
3:    if with probability pd then
4:       s[i] := s_g[i]
5:    end if
6: end for
7: return s
```

5 Experiments

In this section, we present experiments for our cooperative parallel portfolios using path relinking for SAT and Weighted Partial MaxSAT Solving. We decided to build our parallel portfolio on top of UBCSAT [21], a well-known local search library that provides efficient implementations of popular local search algorithms for SAT and MaxSAT.

In our experiments we use the sequential local search algorithms with their default parameters and MaxFlips = 10^6 except for non-cooperative algorithms. Indeed, sequential algorithms are equipped with important diversification techniques and usually perform better without restarts and therefore we use MaxFlips = ∞ for non-cooperative parallel portfolios.

5.1 SAT Experiments

In these experiments we consider all known satisfiable uniform random k-SAT instances from the 2017 and 2018 SAT competitions (for a total collection of 174 instances).[4] and we consider the following algorithms: *AN+*, *G2WSAT*, *PAWS*, *AG2*. We evaluated the impact of our cooperative policies with two versions of the portfolio. The first version analyses the impact of the new policies with multiple copies of the same algorithm and the second one implements a parallel portfolio with four different algorithms.

We conducted this set of experiments on 15 machines running Ubuntu18.04 with 16GB of RAM and a AMD Razen 5 2400g CPU with 4 cores. We ran each solver 5 times on each instance (each time with a different random seed) with a 5-minute time cutoff. For each pair ⟨instance, solver⟩ we compute the median time and the Penalized Average Runtime (*PAR10*), i.e., average runtime, but unsolved instances are considered as 10× the time-limit [9], over all 5 runs .

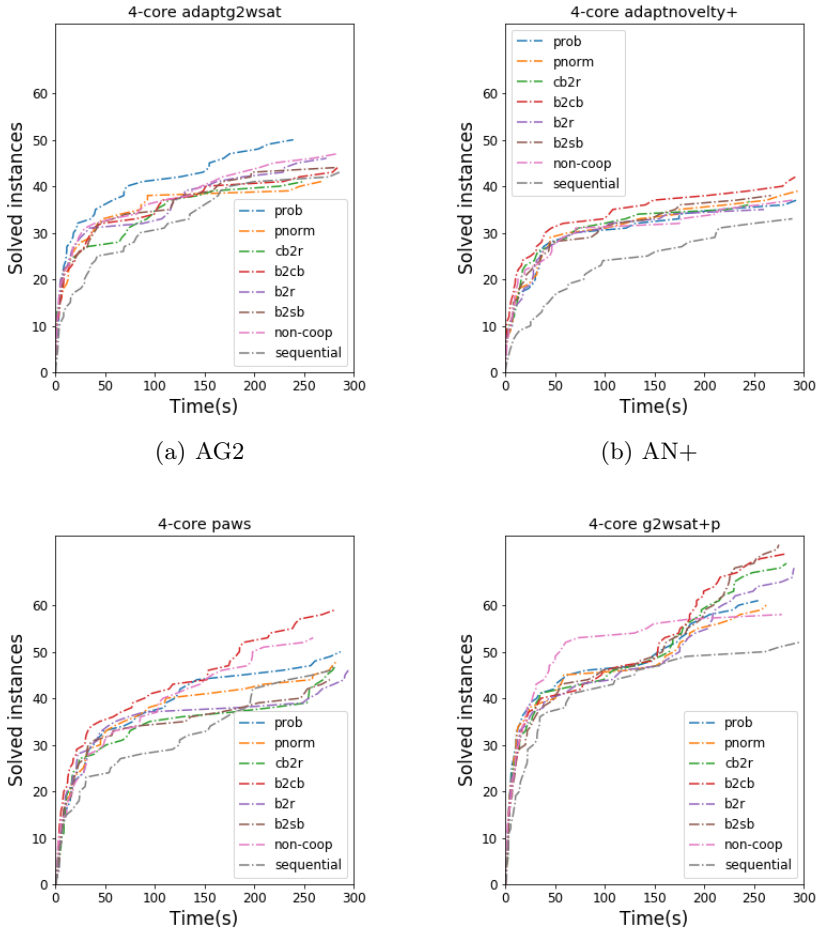
Figure 3 shows the cactus plot of the parallel portfolios with multiple copies of the same algorithm. The y-axis gives the number of solved instances and the x-axis presents the cumulative runtime. In this figure, it can be seen that our new cooperative policies with path relinking outperform (except for *AN+*) existing techniques such as: *PNorm*, *Prob*, and a portfolio without cooperation (*non-coop*).

Figures 3(a) and 3(b) show the performance of the two weakest algorithms, that is, *AG2* with 51 instances in 280 seconds (for *Prob*) and *AN+* with 43 instances in 267 seconds (for *best2cbest*). Figure 3(c) shows that *best2cbest* is the best cooperative policy for *PAWS* solving 60 instances in 276 seconds. On the other hand, Figure 3(d) summarizes the performance of the best algorithm, it can be observed that the non-cooperative framework outperforms the other methods when the time cutoff is up 150 seconds. However, after this point, *best2sbest* and *best2cbest* report outstanding performances with respectively 74 instances in 275 seconds and 72 instances in 281 seconds.

Figure 4 shows the cactus plot of the parallel portfolio with different algorithms, *sequential* reports the performance of the best sequential algorithm (i.e., *G2WSAT*). Similarly to Figure 3(d) the non-cooperative portfolio reports a very good performance up to about 200 seconds. However, after this point our new path relinking policies largely outperform *non-coop*, *Prob* and *PNorm*. In particular, *cbest2rand* (resp. *best2cbest*) solves 7.4% (resp. 12%) more instances than *PNorm* (resp. *Prob*).

Table 1 reports complete details of the performance of a 4-core portfolio with different algorithms. # Solved reports the number of solved instances within the time limit, Time denotes the average time in seconds for solved instances (i.e., average across instances of the median across 5 runs on a given instance), and PAR10 reports the average PAR10 of the parallel portfolio. It can be observed that all our new path relinking policies outperform existing methods (except *best2sbest*), i.e., *cbest2rand* and *best2cbest* solve seven (resp. five) more instances than *PNorm* (resp. Non-cooperation).

[4] https://satcompetition.org/



(a) AG2 (b) AN+

(c) PAWS (d) G2WSAT

Fig. 3. Cactus plot for 4-core portfolios using copies of the the same algorithm.

Table 1. Portfolio full results

Method	# Solved	Time	PAR10
Sequential	53	51	2101
Non-cooperative	63	41	1928
Pnorm	65	58	1901
Prob	62	55	1950
best2cbest	70	73	1822
cbest2rand	**70**	**69**	**1821**
best2rand	68	66	1853
best2sbest	64	61	1919

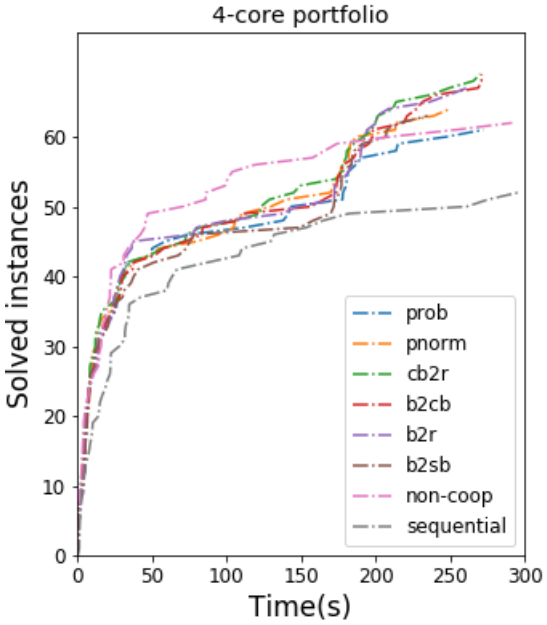


Fig. 4. Cactus for 4-core Portfolio with different algorithms

Table 2 reports the performance of a 4-core parallel portfolio with multiple copies of the best sequential algorithm, i.e., *G2WSAT*. In this experiment, we observe that all our new policies outperform well-known techniques, for instance, the worst path relinking algorithm solves more instances than the best exiting technique for this experiment (i.e., *Prob*). Furthermore, our overall best new policy (i.e., *best2sbest*) solves 21% more instances than the non-cooperative portfolio and 25% more instances than the best existing method. As expected the time of solved instances increases as this parallel algorithm solves more instances.

Finally, Table 3 summaries the overall results of the 4-core portfolios. Algorithm indicates the base local search algorithm. Coop. Policy indicates the cooperative policy, each cell shows the performance of Non-cooperative portfolios, Prob, and the best parallel portfolio for the reference algorithm.[5] As it can be observed the cooperative portfolio always outperform the non-cooperative one, and our new path relinking policies outperform all other policies in 4 out of 5 experiments. Furthermore, *G2* equipped with best2sbest is the overall winner policy with 74 solved instances in 87 seconds.

[5] Please notice that *AG2* only reports two cooperative policies as Prob is the winner strategy.

Table 2. Results for 4-core G2 Parallel Portfolios

Method	# Solved	Time	PAR10
Sequential	53	51	2101
Non-cooperative	59	31	1993
PNorm	61	57	1968
Prob	62	55	1950
best2cbest	72	79	1791
cbest2rand	70	78	1824
best2rand	69	82	1843
best2sbest	**74**	87	**1761**

Table 3. Experiment results

Algorithm	Coop. Policy	# Solved	Time	PAR10
AG2	Non-cooperation	48	54	2187
	Prob	51	45	2134
AN+	Non-cooperation	38	51	2355
	Prob	38	52	2356
	best2cbest	43	54	2272
G2	Non-Cooperation	59	31	1993
	Prob	62	55	1950
	best2sbest	**74**	**87**	**1761**
PAWS	Non-Cooperation	54	67	2090
	Prob	51	63	2139
	best2cbest	60	41	1990
Portfolio	Non-Cooperation	63	41	1928
	Prob	62	55	1950
	cbest2rand	70	69	1821

5.2 Weighted Partial MaxSAT

We conducted experiments using crafted and random instances. The first dataset is a collection of 234 crafted instances used regularly in the annual MaxSAT competitions: staff-scheduling (12), auctions/auc-paths (20), auctions/auc-scheduling (20), min-enc/planning (30), warehouses (18), casual-discovery (35), csg (10), random-net (32), set-covering (45), mip-lib (12).

For the second dataset, we followed a similar approach as [13] and used *makewff* [24] to generate 270 uniform random weighted partial MaxSAT instances around the phase transition, i.e., 90 3-SAT instances, 90 5-SAT instances, and 90 7-SAT instances; and the number of variables per instance ranges from 2000 to 4000 (3-SAT), 1000 to 3000 (5-SAT), and 300 to 500 (7-SAT). For each random instance we randomly split clauses into two disjoint sets with hard and soft clauses. The number of hard clauses varies bewteen 10% - 40% of the total clauses in the problem.

After a preliminary experimentation we decided to use multiple copies of *AdapNovelyt+* to build our parallel portfolio for the random dataset. We would like to remark that *WalkSAT* and *G2WSAT* reported a poor performance and were unable to find feasible solutions for this problem family (i.e., satisfying all hard clauses). Alternatively, we use *AdaptNovelty+*, *WalkSAT*, and *G2WSAT* for crafted instances, so that we build our portfolios for crafted instances as follows:

- 4 Cores: *AdaptNovelty+* (2 cores), *WalkSAT* (1 core), and *G2WSAT* (1 core);
- 8 Cores: *AdaptNovelty+* (3 cores), *WalkSAT* (2 cores), and *G2WSAT* (3 cores).

We compare our cooperative algorithm against the following state-of-the-art local search solvers (with their recommended parameters): *Dist*, *CCEHC*, *CCLS*, *Prob*, and *PNorm*.[6] We remark that we use the same configurations for all our portfolios using the UBCSAT library. Unfortunately, *Dist*, *CCEHC*, *CCLS* do not support parallelism, and therefore, the only feasible parallel option for these solvers is the parallel portfolio without cooperation.

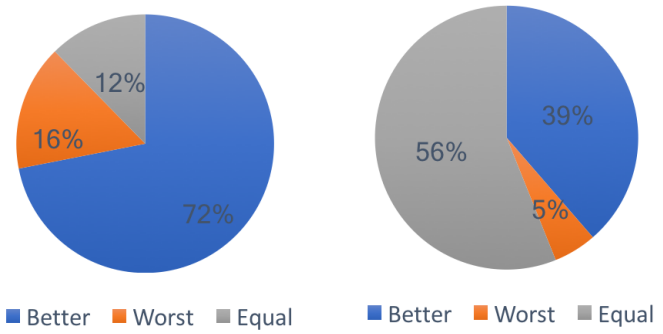
We conducted this set of experiments in the Microsoft Azure Cloud using DS4_v2 virtual machines with 28 GB of RAM and 8 cores at 2.40 Ghz Intel Xeon Processors E5-2673 running ubuntu. We ran each solver 5 times on each instance (each time with a different random seed) with a 5-minute wall-clock timeout (300 seconds) for each experiment. For each pair ⟨instance, solver⟩ we compute the median time and solution quality over all 5 runs. Furthermore, we report the number of instances a given solver finds the best solution among all the solvers. We restart our local search solveres in all cooperative portfolios (i.e., *PNorm*, *best2rand*, and *best2cbest*) every 10^6 iterations or flips.

We start our evaluation with Figure 5, we compare the performance of *best2rand* (cooperative portfolio) vs. *Portfolio* (non-cooperative portfolio) with

[6] In this paper, we use the implementation of *Dist*, *CCEHC*, *CCLS*, *Prob* and *PNorm* available in the SAT competitions and the website of the authors.

eight cores. In both cases we use the same reference algorithms to build the portfolio.

As it can be seen in the figure, the cooperative framework implementing path relinking helps to considerably improve performance. For random instances (Figure 5(a)) *best2rand* outperforms *Portfolio* for 128 instances; for 22 instances both solvers report the same solution cost; and only for 28 instances *Portfolio* outperforms *best2rand*. Alternatively, for crafted instances (Figure 5(b)) *best2rand* outperforms *Portfolio* for 73 instances; *Portfolio* outperforms *best2rand* for 106 instances; and interestingly the non-cooperative portfolio only outperforms the cooperative one for 10 instances. It is also worth mentioning that *best2rand* is faster than *Portfolio* when both parallel solvers report the same solution cost, i.e., 13 and 47 times faster for random and crafted instances.



(a) Random instances. (b) Crafted instances.

Fig. 5. *best2rand* vs. *Portfolio*. Proportion of instances where the *best2rand* is better (resp. worst and equal) than *Portfolio* (counterpart portfolio without cooperation).

Table 4 presents further experimental results with the performance of sequential and parallel algorithms with and without cooperation using 4 and 8 cores for random instances. We recall that for random instances we build our parallel portfolio with and without cooperation using *AdaptNovelty+* (denoted as *Portfolio* in the table), and we omit the performance of *WalkSAT* and *G2WSAT* as these two solvers are unable to find feasible solutions for this problem family. Actually, the sequential version of the solvers in UBCSAT are unable to find the best solution for random instances (i.e., an assignment for the variables that satisfies all hard clauses), while *CCLS* and *CCEHC* solve 2 instances.

These results confirm that the cooperative approach with path relinking outperforms its counterpart portfolio with existing cooperative policies and without cooperation. For instance, *best2rand* solves (with 8 cores) respectively 36 and 32 more instances than the reference portfolio without cooperation (i.e., *Portfolio*) and *PNorm*.

As expected our non-cooperative portfolio is considerably weaker than *CCLS* (best sequential solver). However, adding our suggested cooperative framework

Table 4. Results for random instances.

Algorithm	Sequential		4 Cores		8 Cores	
	Time (s)	Best	Time (s)	Best	Time (s)	Best
best2rand	–	0	**189.6**	**22**	**157.4**	**54**
best2cbest	–	0	176.2	20	185.5	49
PNorm	–	0	149.8	7	162.9	22
Portfolio	–	0	114.1	6	166.3	18
CCLS	**183.9**	**2**	143.3	15	139.6	38
Dist	2.3	1	108.1	7	108.4	14
CCEHC	266.5	2	116.1	7	125.4	16

leads to substantial performance improvements. As a result of that, our cooperative portfolio greatly outperforms the parallel version of *CCLS*, e.g., *best2rand* solves 7 and 16 more instances than *CCLS* with 4 and 8 cores.

Table 5. Results for crafted instances.

Algorithm	Sequential		4 Cores		8 Cores	
	Time (s)	Best	Time (s)	Best	Time (s)	Best
best2rand	49.2	74	56.6	113	57.3	123
best2cbest	49.2	74	51.3	110	**87.0**	**138**
PNorm	49.2	74	38.8	101	38.4	103
Portfolio	49.2	74	35.4	98	26.2	99
AdaptNovelty+	49.2	74	40.8	86	42.9	91
WalkSAT	10.2	41	12.6	44	20.3	48
G2WSAT	25.8	57	16.1	60	12.1	61
CCLS	4.5	45	11.3	50	9.7	50
Dist	14.7	88	11.4	94	27.3	108
CCEHC	**15.1**	**109**	**13.0**	**115**	18.7	121

We now switch our attention to crafted instances (Table 5). In this experiment, we include experimental results for our reference sequential solvers from UBCSAT (i.e., *WalkSAT*, *G2WSAT*, and *AdaptiveNovelt+*) as these solvers report competitive performance against modern local search solvers (i.e., *CCLS*, *Dist*, and *CCEHC*) for the Weighted Partial MaxSAT problem. In this dataset, it can be observed that *CCEHC* is the best sequential solver, reporting 109 instances with the best performance, followed by *Dist* (88 instances), and *AdaptiveNovelty+* (74 instances).

Similarly to random instances, our cooperative solver with path relinking outperforms its counterpart solvers *PNorm* (cooperative solver) and *Portfolio* (parallel portfolio without cooperation). For instance, *bets2rand* solves 9 and 12 more instances than *PNorm* and *Portfolio* with 8 cores. *CCEHC* is the best portfolio with 4 cores solving 115 instances, 2 more than *best2rand*. This performance difference is mainly because *CCEHC* is considerably better (for this dataset) than

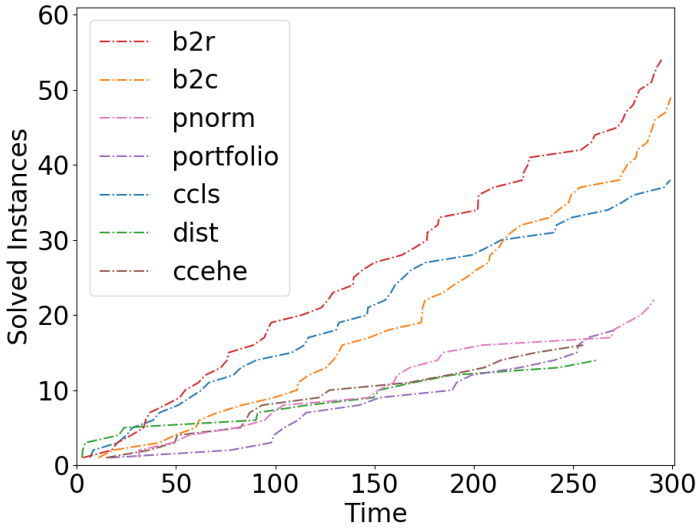


Fig. 6. Cactus plot for 8-core portfolios with random instances.

our sequential algorithms from the UBCSAT library. Finally, *best2cbest* leads the ranking (with 8 cores) by solving 17 more instances than the parallel portfolio with the best sequential solver (i.e., *CCEHC*). Certainly, this performance improvement comes from our path relinking cooperative framework.

Finally, Figures 6 and 7 show the cactus (i.e., number of solved instances with given a time limit) plot for 8-core portfolios for random and crafted instances. *best2rand* (b2r) is the best parallel solver, the second place is for *best2cbest* (b2c), and the third place is for *CCLS*. On the other hand, for crafted instances (Figure 7) *best2cbest* and *best2rand* are the most effective solvers.

6 Conclusions and Future Work

In this paper, we proposed a cooperative framework using path relinking, a well-known technique to combine solutions in meta-heuristic search. The algorithm exploits parallelism by executing multiple local search algorithms in parallel, at each restart point, instead of classically generating a random solution to start with, we propose the use of path relinking to carefully craft new starting solutions.

Extensive experiments on a large number of instances for the SAT and Weighted Partial MaxSAT problems suggest that our new cooperative framework outperforms its counterpart parallel portfolio with and without cooperation. Furthermore, we have seen improvements for parallel portfolios with multiple copies of the same algorithm and parallel portfolios with different algorithms.

In the future, we would like to investigate the use offline and online tuning of the *pd* parameter to balance the diversification vs. intensification trade-off.

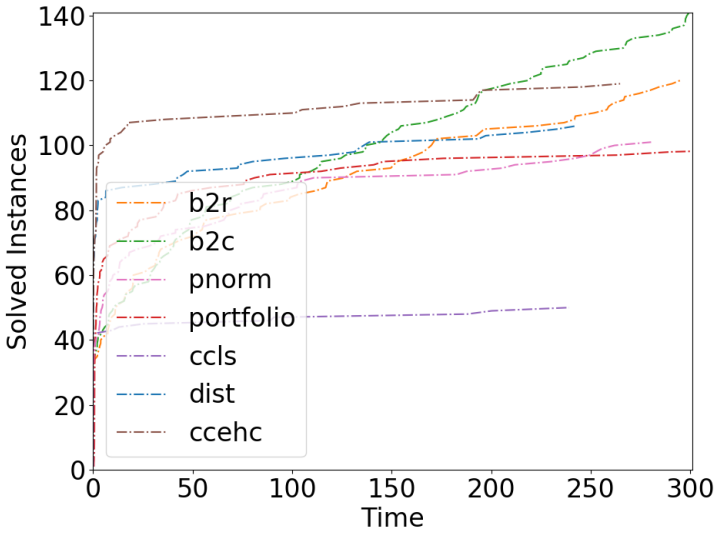


Fig. 7. Cactus plot for 8-core portfolios with crafted instances.

On the one hand, for the offline case, we plan to explore automatic tools such as ParamLS [10] and F-RACE [4]. On the other hand, for the online case, we would like to investigate the use of reinforcement learning for self-adaptive tuning of the *pd* parameter. Finally, we would like to investigate the use of supervised machine learning to identify the best set of algorithms for a given problem instance [3].

References

1. Arbelaez, A., Codognet, P.: From sequential to parallel local search for SAT. In: EvoCOP. pp. 157–168 (2013)
2. Arbelaez, A., Hamadi, Y.: Improving parallel local search for SAT. In: LION 5. pp. 46–60 (2011)
3. Arbelaez, A., Hamadi, Y., Sebag, M.: Continuous search in constraint programming. In: Autonomous Search, pp. 219–243 (2012)
4. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: GECCO. pp. 11–18 (2002)
5. Cai, S., Luo, C., Lin, J., Su, K.: New local search methods for partial maxsat. Artif. Intell. **240**, 1–18 (2016)
6. Glover, F.: Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). Discrete Applied Mathematics **49**(1-3), 231–255 (1994)
7. Hamadi, Y., Jabbour, S., Sais, L.: Manysat: a parallel SAT solver. JSAT **6**(4), 245–262 (2009)
8. Hoos, H.H.: An adaptive noise mechanism for walksat. In: AAAI/IAAI. pp. 655–660 (2002)
9. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Tradeoffs in the empirical evaluation of competing algorithm designs. Ann. Math. Artif. Intell. **60**(1-2), 65–89 (2010)

10. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: Paramils: An automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)
11. Li, C.M., Wei, W., Zhang, H.: Combining adaptive noise and look-ahead in local search for SAT. In: SAT'07. pp. 121–133 (2007)
12. Luo, C., Cai, S., Su, K., Huang, W.: CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artif. Intell. **243**, 26–44 (2017)
13. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans. Computers **64**(7), 1830–1843 (2015)
14. Martins, R., Manquinho, V.M., Lynce, I.: An overview of parallel SAT solving. Constraints **17**(3), 304–347 (2012)
15. Roli, A.: Criticality and parallelism in structured SAT instances. In: Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings. pp. 714–719 (2002)
16. Roli, A., Blesa, M.J., Blum, C.: Random Walk and Parallelism in Local Search. In: Metaheuristic International Conference (MIC'05). Vienna, Austria (2005)
17. Selman, B., Kautz, H.A., Cohen, B.: Noise Strategies for Improving Local Search. In: AAAI. pp. 337–343 (1994)
18. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. In: AAAI'96. pp. 440–446
19. Strickland, D.M., Barnes, E.R., Sokol, J.S.: Optimal protein structure alignment using maximum cliques. Operations Research **53**(3), 389–402 (2005)
20. Thornton, J., Pham, D.N., Bain, S., Jr., V.F.: Additive versus multiplicative clause weighting for SAT. In: AAAI. pp. 191–196 (2004)
21. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In: SAT'04 (2004)
22. Vasquez, M., Hao, J.: A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Comp. Opt. and Appl. **20**(2), 137–157 (2001)
23. Zhang, L., Bacchus, F.: MAXSAT heuristics for cost optimal planning. In: AAAI'12
24. Zhang, W., Rangan, A., Looks, M.: Backbone guided local search for maximum satisfiability. In: IJCAI'03. pp. 1179–1186 (2003)