On the Performance of CDCL-based Message Passing Inspired Decimation using $\rho\sigma$PMP

Oliver Gableske and Sven M"uelich and Daniel Diepold
Ulm University, Theoretical Computer Science, 89081 Ulm, Germany, 
{oliver.gableske,sven.mueelich,daniel.diepold}@uni-ulm.de

Abstract. In [11], an algorithm called genMID is introduced, which is a generic approach for designing Message Passing Inspired Decimation (MID) solvers. Furthermore, [11] derives a flexible Message Passing (MP) heuristic, called $\rho\sigma$PMP, which is able to mimic the behavior of all product-based MP heuristics currently available for SAT. Even though [11] provides a generic approach to derive MID solvers using $\rho\sigma$PMP, no actual MID solver is presented. The first contribution of the paper at hand is to derive such a MID solver. It follows the genMID approach in order to combine a CDCL search with $\rho\sigma$PMP, resulting in the CDCL-based MID solver DimetheusMP. The MP related parameters of this solver are then tuned in order to adapt it to a wide variety of crafted and random formulas from the SAT Challenge 2012. The performance of DimetheusMP is then compared to DimetheusJW, which applies the exact same CDCL search as DimetheusMP but uses Jeroslow-Wang instead of $\rho\sigma$PMP in order to guide the search. The paper shows that the DimetheusMP solver significantly outperforms the DimetheusJW solver. It furthermore shows that the flexibility of the MP heuristic $\rho\sigma$PMP is indeed necessary to achieve this performance; the parameter tuning prefers this MP heuristic above all others. Additionally, the paper shows that the MID approach enables CDCL search to solve satisfiable uniform random formulas, and that Message Passing can be helpful to solve both satisfiable and unsatisfiable crafted formulas.

1 Introduction

SAT is one of the most studied combinatorial problems and the interest in practically applicable algorithms to solve this problem (called SAT solvers) has increased during the past decades. Several classes of SAT solvers are available, like Conflict-Driven Clause Learning (CDCL), Stochastic Local Search (SLS), and look-ahead solvers. A similarity between all of them is their application of variable and value selection heuristics to advance the search. Well known examples for such heuristics are VSIDS and phase-saving [22, 23].

Message Passing (MP) is a comparatively new class of such heuristics [13, 4]. An MP heuristic creates estimations of variable assignments, called biases. An MP based SAT solver then uses these biases for assigning variables and thus can simplify the formula under the assumption that these biases capture a
partial satisfying assignment. We call the simplification using biases \textit{MP Inspired Decimation} (MID). Estimating biases and performing MID is repeated until the formula is solved by the set of given assignments or a conflict occurred. If a conflict occurs, the solver either reports the failure of the MID approach or, in conjunction with a complete search strategy like CDCL, can learn from the conflict, backjump, and repeat the bias computation along with MID. SAT solvers that apply an MP heuristic and MID are henceforth called \textit{MID solvers}.

So far, several MP heuristics have been proposed in the literature. The four basic product-based MP heuristics currently available for SAT are Belief Propagation (BP) \cite{19}, Survey Propagation (SP) \cite{6,8}, Expectation Maximization BP Global (EMBPG) and Expectation Maximization SP Global (EMSPG) \cite{14,15}. All these approaches have their respective strengths and weaknesses. The main advantage of BP and SP is the quality of the biases they report. These biases capture partial satisfying assignments much more precisely than the biases computed by the EM variants \cite{13}. This works comparatively well on random k-SAT formulas. However, these non-EM variants suffer from the fact that they are not guaranteed to converge. This gave rise to the development of EMBPG and EMSPG \cite{13}, which are guaranteed to converge but suffer from the weaker quality of the biases to capture a satisfying assignment. The latter heuristics do, however, seem to provide better performance on crafted formulas \cite{13}.

So far, one always had to choose between these basic MP heuristics in order to realize a MID solver. This necessity to choose in conjunction with the different strengths and weaknesses of the heuristics is a serious drawback and several attempts have been made in the literature to overcome this drawback by trying to interpolate between these heuristics \cite{1,20}. The goal was to derive an MP heuristic with increased flexibility that allows for good performance of a MID solver on a wide variety of CNF formulas by adaption of the MP behavior.

In \cite{11}, a thorough introduction to the different MP heuristics is given. This paper also provides the ISI technique to combine two given MP heuristics into an interpolation. Several interpolations are derived using ISI and are shown to constitute a hierarchy of generality. The most general MP heuristic that is derived in \cite{11} is $\rho\sigma\text{PMP}^i$. Refer to Fig. 1 for an overview of the interpolations and different levels of the hierarchy.

The $\rho\sigma\text{PMP}^i$ heuristic is able to interpolate between all the presented MP heuristics using two interpolation parameters $\rho, \sigma \in [0,1]$. Here, $\rho$ captures the difference between the BP and SP variants basically allowing to control how “careful” the MP heuristic reports biases. Setting $\rho = 0$ results in a BP behavior being least careful, and setting $\rho = 1$ results in an SP behavior being most careful. Furthermore, $\sigma$ captures the difference between the non-EM variants and the EM variants basically allowing to control how much the convergence of the heuristic is enforced. Setting $\sigma = 0$ results in non-EM behavior that does not enforce the convergence at all, and setting $\sigma = 1$ results in the EM behavior that guarantees convergence. The application of $\rho, \sigma \in (0,1)$ results in an intermediate behavior. We suggest reading \cite{11} for a proper introduction to MP, the various MP heuristics, and their interpolations.
Fig. 1. A conceptual overview of the MP heuristics presented in [11]. The arrows indicate which heuristics are interpolated using ISI. The paper at hand focuses on $\rho\sigma\text{PMP}^i$, which is able to interpolate between all the other mentioned heuristics.

All together, using $\rho\sigma\text{PMP}^i$ remedies the need to choose from one of the other heuristics. Its flexibility allows to adapt the MID solver’s MP behavior to mimic the behavior needed to solve a specific formula.

Even though [11] provides a self-contained theoretical derivation of $\rho\sigma\text{PMP}^i$, the empirical support for the claim that the application of $\rho\sigma\text{PMP}^i$ is indeed helpful in practice is quite preliminary. The paper at hand will focus on these practical aspects. First, it provides the information on how to realize a MID solver using $\rho\sigma\text{PMP}^i$ and CDCL. Second, it will provide details regarding a substantial empirical study that was performed on random and crafted formulas of the SAT Challenge 2012 [2].

Three main conclusions can be drawn from this study. First, the CDCL-based MID approach using $\rho\sigma\text{PMP}^i$ is helpful in practice and the flexibility provided by $\rho\sigma\text{PMP}^i$ is crucial to its success. Second, CDCL-based search in conjunction with MID can solve uniform random k-SAT formulas. Third, it can be beneficial to use MP to solve both satisfiable and unsatisfiable crafted formulas.

Before we go into the details of the empirical study, we provide information on how a CDCL-based MID solver can be realized in the next section.

2 Deriving a CDCL-based MID solver using $\rho\sigma\text{PMP}^i$

As stated in the introduction, CDCL solvers use variable and value selection heuristics. Whenever a CDCL solver must make a decision in order to advance the search (select a not yet assigned variable and assign it to either true or false), it will call for these heuristics to provide a decision literal (a variable with a sign). It will then assign the corresponding variable such that the decision literal becomes true. The CDCL will then perform unit propagation which either results in a conflict or not. If no conflict arises, another decision is made until a satisfying assignment has been constructed. If a conflict arises, the CDCL will analyze the conflict and back-jump to advance the search again, or deduce the unsatisfiability of the formula if the conflict arose unconditionally.

In practice, selecting a decision literal in CDCL solvers is commonly done via VSIDS and phase-saving [22, 23]. However, both heuristics must be initialized.
That is, one needs to initialize variable activities for VSIDS, stating an initial ordering of the variables in which they are to be assigned. Furthermore, one needs to initialize the variable phases for phase-saving, stating what assignment is to be checked first for the variables.

As already stated in the introduction, the main goal of MP is to compute biases to all variables $v_i, i \in \{1, \ldots, n\}$, of a given CNF formula $F$. A bias $\beta(v_i) \in [-1, 1]$ can be understood as both activity and phase for variable $v_i$. Here, the abstract of the bias is its activity. The larger the bias abstract, the higher the initial activity. The sign of the bias is understood as the variable phase. A negative bias suggests an assignment to false, and a positive bias suggest an assignment to true. See [11] for further details.

A more common approach for the initialization of VSIDS and phase-saving in CDCL solvers is the Jeroslow-Wang (JW) heuristic [18], that computes a score for each variable. The variable score is the sum of the positive and negative literal scores that JW computes. The literal scores depend on the number of occurrences of the literals and the sizes of the clauses in which they occur. In that sense, JW is able to provide variable activities (based on the total score of the variable) and phases (based on which literal has the higher score). After computing the variable scores to all $n$ variables, one can normalize these values into $[0, 1]$ using the largest value. The sign (positive or negative) is determined by the literal score. All together, JW and MP can both provide biases in $[-1, 1]$.

The idea now is to implement CDCL search in such a way that it can apply either MP or JW in order to initialize VSIDS and phase-saving, but leaves all other aspects of the CDCL search untouched. One can then compare the performance of these two versions of the solver in order to determine how helpful the application of MP is in comparison to JW.

We implemented Dimetheus [10], which is a CDCL-based MID solver that follows the genMID approach from [11]. The bias computations in Dimetheus can be realized by either JW or $\rho\sigma PMP^i$. The two versions, called DimetheusJW and DimetheusMP, perform a similar CDCL search, and differ only in the way they initialize (and possibly re-initialize) VSIDS and phase-saving using either the biases computed by JW or those computed by $\rho\sigma PMP^i$. During the ongoing CDCL search, both versions modify VSIDS activities and variable phases as commonly done in CDCL. The variable activities are modified during FirstUIP conflict analysis, and the variable phases are modified during back-jumping.

The application of $\rho \sigma PMP^i$ in conjunction with the MID approach introduces the MP related parameters into DimetheusMP. These parameters are $\rho$ (which controls the carefulness of the MP heuristic when reporting biases), $\sigma$ (which controls the convergence behavior of the MP heuristic), and $p$ (which constitutes the reliability of the computed MP biases and controls how often biases must be re-computed after assigning a specific number of variables). The settings to these parameters can be optimized with the means of parameter tuning. The application of JW does, however, not provide any parameters that can be tuned. The crucial observation is that DimetheusMP has an increased flexibility in comparison to DimetheusJW, and that parameter tuning allows to adapt
the MP and MID behavior of DimetheusMP to a given set of formulas. The parameter settings for $\rho$ and $\sigma$ influence the Message Passing itself, i.e. the bias computation. The parameter $p$ controls how often these biases are re-computed. The idea here is to compute biases for all variables once, and assign $p \cdot n$ many not yet assigned variables with strongest bias. Therefore, DimetheusMP might re-compute biases and re-initialize VSIDS and phase-saving for the remaining variables during search, while DimetheusJW initializes VSIDS and phase-saving exactly once. In summary, the additional flexibility of DimetheusMP is on one hand related to the flexible MP heuristic $\rho\sigma PMP^i$ in conjunction with its parameters $\rho$ and $\sigma$, and on the other hand, to the generic MID approach that allows to re-compute biases in conjunction with parameter $p$.

The purpose of this paper is to compare the performance of the two Dimetheus versions and show via an empirical study that DimetheusMP can significantly outperform DimetheusJW once its MP related parameters have been tuned. We thereby aim to provide empirical evidence that the MID approach in conjunction with $\rho\sigma PMP^i$ is helpful in practice when combined with CDCL search.

The following section will provide details regarding this empirical study and reports on our findings for optimal parameter settings for different types of formulas from the SAT Challenge 2012.

3 Empirical Study

This section provides details regarding the empirical study. It will explain the hard- and software used, the formulas we chose to conduct the experiments, and present the results. The section is concluded with a discussion that states weaknesses of our study and summarizes the most important results.

All necessary data to repeat this study can be downloaded from [12]. This package contains the CNF formulas, solvers, a description on how to use these solvers, as well as all the raw data we gathered during our experiments. Additionally, the package contains reference [11] which provides additional information on MP and $\rho\sigma PMP^i$.

3.1 Purpose and Idea of the Empirical Study

The purpose of this empirical study is to determine, whether the application of the $\rho\sigma PMP^i$ heuristic in conjunction with MID is helpful in practice when combined with CDCL search. In order to do that, we analyze and compare the performance of DimetheusMP and DimetheusJW. We aim to show, that there is a significant performance benefit of using $\rho\sigma PMP^i$ in comparison to Jeroslow-Wang, for both random k-SAT and crafted formulas.

The formulas chosen for our experiments are the random and crafted formulas from the SAT Challenge 2012 (SC2012, see [2]). We separated the provided formulas into classes by grouping formulas of the same type or source.

We then used the CDCL solvers Lingeling and DimetheusJW (that do not apply MP) to solve the instances once in order to aquire the CDCL-only performance for each of the classes. Here, Lingeling is used to merely provide
a reference to a state-of-the-art (SOTA) CDCL solver. The performance of \texttt{DimetheusJW} is considered to be the base performance that will provide a reference for the MP enabled CDCL \texttt{DimetheusMP}.

We then used \texttt{DimetheusMP} and the means of parameter tuning to identify good settings for the MP related parameters $\rho$, $\sigma$, and $p$ (see [11] for details on those parameters) for each of the formula classes. After these settings have been identified, we compare the performance of \texttt{DimetheusJW} and \texttt{DimetheusMP} to determine if the application of the MP heuristic $\rho\sigma\text{PMP}_i$ in conjunction with the MID approach can be helpful in practice.

3.2 Hard- and Software

The hardware we used to conduct the empirical study is part of the bwGRiD [7]. It provided us with 48 compute nodes for which each node consists of 2 Intel Harpertown quad-core CPUs with 2.83 GHz and 16 GByte RAM.

The operating system that runs on the bwGRiD is Scientific Linux 5.5. The support software we used to carry out the study is EDACC [3] in conjunction with the AAC [9] module. The solvers we used are \texttt{Lingeling} [5], and \texttt{Dimetheus} [10] in the two versions JW and MP.

3.3 Benchmarks

As already stated, the formulas we used are the crafted and uniform random formulas from the SC2012 [2]. We separated the formulas into classes of same type/source (e.g. all satisfiable battleship instances went into the class battleship-sat, all uniform random 3-SAT instances with ratio 4.2 went into k3-r4.2). We also took care that the classes do not become too small (it simply makes no sense to have a single crafted formula in its own class), and ignored the formulas that could not be added to any class in a meaningful way. This gives us a total of 376 crafted formulas in 29 classes and 600 uniform random formulas in 50 classes.

3.4 Execution of the Study

We first used \texttt{Lingeling} and \texttt{DimetheusJW} to solve each formula exactly once (using multiple runs with different seeds is unnecessary here since the solvers are deterministic). The timeout was set to 2000 seconds. We thereby acquired the success rate and PAR10 time (penalized average runtime with factor 10 for unsuccessful runs) of these solvers for each of the classes.

We then used \texttt{DimetheusMP} and the EDACC/AAC to determine good settings for the MP related parameters $\rho$, $\sigma$, and $p$ for each of the classes. Again, the timeout was set to 2000 seconds. We thereby acquired the success rate and PAR10 time for \texttt{DimetheusMP} regarding the best parameter setting found.

It is important to note that \texttt{DimetheusMP} is not a deterministic solver as \texttt{DimetheusJW} or \texttt{Lingeling}. Two possibilities for applying randomness exist.

First, the \texttt{DimetheusMP} solver could apply randomness in order to initialize the disrespect and warning messages for the initial MP cycle. However, we
decided to use a deterministic initialization based on the clause length for the respective messages in order to avoid using randomness here.

Second, the DimetheusMP solver requires random clause permutations for conducting the Message Passing (see [11] for details on how MP works in general). The question is, whether different seeds do give different results regarding the biases computed by MP, and whether this difference does indeed justify that multiple runs with different seeds must be computed to get an average performance (as it is commonly done when studying SLS solvers).

As it turns out, different seeds do indeed give different clause permutations, but the difference of biases computed with different seeds is practically non-existent. The differences commonly appear behind the third digit after the comma, resulting in very local changes to the bias ranking of the variables. In other words, the biases reported by MP are very stable for multiple seeds. We therefore decided that the additional computational overhead, that is imposed on the study when using multiple runs with different seeds, is simply not worth it. Therefore, DimetheusMP performs, as Lingeling and DimetheusJW, one run for each formula in a class for each of the parameter settings.

The resulting success rates and PAR10 times of the three solvers (with the best parameter settings found for DimetheusMP for each of the formula classes) are provided in the next section.

3.5 Results

Tables 1 and 2 summarize the results of our empirical study. For each formula class we report on the following data. The “S” column indicates whether the formulas are satisfiable (y) or not (n). The “#” column gives the number of formulas in the class. The “%” column gives the percentage of formulas solved in the class (success rate), while the “PAR10” column gives the penalized average runtime in seconds. The “Comp.” column indicates where DimetheusMP significantly outperforms DimetheusJW (marked with *) or Lingeling (marked with **). The ρ, σ, and p columns state the parameter setting for DimetheusMP for which it gave the reported performance in this class. Finally, the “Level” column shows what the most specific MP heuristic is that can achieve this performance. Level 0 (L0) is the group of basic MP heuristics (consisting of BP, SP, EMBPG and EMSPG). Level 1 (L1) is the group of interpolations that are derived from the L0 heuristics by a single application of the ISI technique (consisting of ρSPi, ρEMSPGi, σEMBPGi, and σEMSPGi). Level 2 (L2) is the group consisting of only the MP heuristic ρσPMPi, that is derived by a second application of the ISI technique on two of the L1 heuristics. In summary, the higher the level, the more generality is needed in the MP heuristic in order to achieve the given performance (see [11] for further details).

The following section will summarize and discuss the results, and it will state weaknesses of the empirical study.
Table 1. The table shows the results for the Crafted Benchmark (see text for details).

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Solve Performance</th>
<th>Dimens.</th>
<th>% $\text{PAR}^1$</th>
<th>Dimensality</th>
<th>% $\text{PAR}^2$</th>
<th>Dimensality</th>
<th>Longtime Dimensality</th>
<th># Backtracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>edgematching-compact</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMSPG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crafted Benchmark</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- $^1$PAR refers to Time PAR.
- $^2$PAR refers to Space PAR.
<table>
<thead>
<tr>
<th>Random Benchmark</th>
<th>S #</th>
<th>Lingeling</th>
<th>DimetheusJW</th>
<th>DimetheusMP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>%</td>
<td>PAR10</td>
<td>%</td>
</tr>
<tr>
<td>k3-n40000-r4.200</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n35600-r4.208</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n31400-r4.215</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n27200-r4.223</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n23000-r4.230</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n18800-r4.237</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n14600-r4.245</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n10400-r4.252</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n62000-r4.260</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k3-n20204-r4.267</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n10000-r9.000</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n88000-r9.121</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n67800-r9.223</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n68000-r9.324</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n50500-r9.425</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n40400-r9.526</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n30300-r9.627</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n20800-r9.729</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n10180-r9.830</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k4-n90800-r9.931</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k5-n1600-r20.997</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k5-n1420-r20.155</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k5-n1280-r20.275</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k5-n1140-r20.395</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k5-n10300-r21.117</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>All k6 and k7 on all ratios</td>
<td>y</td>
<td>12.0</td>
<td>20000.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table 2.** The table shows the results regarding the uniform random formulas (see text for details).
3.6 A Summary and a Discussion of the Results

A summary of the Results for Crafted Formulas. The DimetheusMP solver is never worse than the DimetheusJW solver, which is a clear indication that (despite the increased computational effort needed to compute $\rho \sigma PMP^{i}$ in comparison to JW) applying MP is at least not harmful when solving crafted formulas. Furthermore, there are several classes where the MP version significantly outperforms the JW version (in terms of success rate or PAR10 time) – on both satisfiable and unsatisfiable formulas.

Furthermore, the reader should note the comparatively weak performance of DimetheusJW in comparison to Lingeling. Even though the direct comparison between these solvers is questionable, since they differ considerably in their implementation, it is a clear indication that the CDCL-only search performed by DimetheusJW is comparatively weak. However, combining this CDCL search with $\rho \sigma PMP^{i}$ and the MID approach gives a clear performance improvement. The flexibility of the DimetheusMP solver in conjunction with the parameter tuning enables DimetheusMP to sometimes outperform even Lingeling. The author clearly wants to state that this comparison is, however, quite bogus since Lingeling was not tuned on the formula classes. This comparison is merely meant to underline the performance improvement that has been achieved for DimetheusMP in relation to the weak performance of DimetheusJW.

The benefit of the additional flexibility of $\rho \sigma PMP^{i}$ in comparison to all other MP heuristics is implied by the last column of the table. Except for the sgen1-unsat and vanderwaerden-sat class, there was no class for which DimetheusMP achieved the best performance using one of the basic (L0) MP heuristics. In 19 of the 29 classes, the flexibility of the L2 heuristic $\rho \sigma PMP^{i}$ was indeed necessary to achieve the given performance.

All together, the results support the claim that the CDCL-based MID approach using $\rho \sigma PMP^{i}$ is indeed helpful in practice when solving crafted formulas. The results also support the claim that it can be beneficial to apply MP to solve both satisfiable and unsatisfiable crafted formulas.

A summary of the Results for Random Formulas. As expected, the CDCL-only solvers Lingeling and DimetheusJW are unable to produce useful results when solving large uniform random k-SAT formulas. This supports the claim that the CDCL paradigm is simply not applicable for solving these formulas by itself. DimetheusMP is, however, able to substantially improve the results. Two things are worth noting.

First, the preferred MP behavior is clearly in favor of the L1 interpolation $\rho SP^{i}$ (which interpolates BP and SP) with a strong tendency towards SP. The claim, that the SP behavior to solve large random k-SAT instances is indeed comparatively good, has been noted in the literature before [6]. However, $\rho \sigma PMP^{i}$ allows to gradually enforce the convergence with $\sigma > 0$. Yet, the parameter tuning suggests that it is not helpful to do so. This is an additional insight, as $\rho \sigma PMP^{i}$ is the first MP heuristic that can gradually adapt and enforce this convergence behavior.
Second, the PAR10 time is comparatively small when DimetheusMP is indeed able to find a satisfying assignment for random formulas. A closer look at the data reveals the following insight. In case the DimetheusMP solver finds a satisfying assignment it does so without running into a single conflict during MID – that is, no actual search is performed. Additionally, if it does run into conflicts it is unable to find a satisfying assignment – that is, search is never successful. This reveals the CDCL ability to “repair” a partial assignment for constructing a satisfying assignment is rather limited. In other words, the CDCL-based MID approach either does not run into conflicts and finds a satisfying assignment, or it runs into conflicts and fails to do so. The case where it runs into conflicts and repairs the assignment in order to reach a solution practically never happens on random formulas.

All together, the results support the claim that the CDCL-based MID approach using $\rho\sigma$PMP$^i$ is indeed helpful in practice when solving large random k-SAT formulas. The statement that the CDCL paradigm is not applicable to solve these formulas is therefore not entirely correct. It merely requires a reliable initialization to VSIDS and phase-saving to fulfill this task. This does not con- fuse the intuition that CDCL search is not helpful in order to solve large random formulas. In case search does take place (due to conflicts arising from following the MP biases) CDCL seems to be unable to extend the partial assignment to a satisfying one.

**Discussion of the Results.** Even though the authors did their best to provide reliable and meaningful data, there are several issues regarding the empirical study that must be pointed out.

First of all, as can be seen in Table 1, some of the classes contain a very limited amount of formulas. It is therefore questionable how robust the reported parameter settings for DimetheusMP really are. The authors were simply not able to acquire more formulas for these classes due to the lack of formula generators for the related problems.

Furthermore, CDCL solvers might need to learn an exponential amount of clauses in order to return a result. This implies a strong demand for RAM, and might result in out-of-memory crashes if several CDCL algorithms run concurrently on a compute node. The authors had to reduce the amount of concurrent jobs per node (CJPN) to only four (while eight would have been possible). This does not remedy the memory problem for crafted and random formulas, but imposes a serious drawback regarding the total time such an empirical study requires. The maximum allowed CJPN for application formulas is even less, and the authors decided to postpone studying the solver performance on these formulas. The lack of results regarding application formulas is certainly an issue of this empirical study.

Additionally, it has been stated before that DimetheusMP is not a deterministic solver (due to the random clause permutations needed to perform MP). We have already argued that there is no notable effect of using different seeds regarding the bias ranking of variables. Nevertheless, not using the average per-
formance of multiple runs with different seeds is a technical weakness of the empirical study.

Finally, the results on the random formulas from Table 2 show that the performance of DimetheusMP is better on random 4-SAT than on random 3-SAT. This is awkward, since an increased $k$ should result in harder formulas and a worse performance. Talking to one of the organizers (Adrian Balint) of the SC2012 revealed, that the random formulas, that have been selected for the SC2012, were filtered by a set of SOTA SLS solvers to ensure that the formulas are satisfiable and not too hard. These solvers are somewhat “overtuned” to solve random 3-SAT and have a comparatively weak performance on random 4-SAT. In that sense, DimetheusMP “detected” that the 3-SAT formulas are comparatively hard, and the 4-SAT formulas are comparatively easy. Providing any insight on how the MP performance degrades when increasing $k$ is therefore not possible with the given empirical data.

4 Conclusions and Future Work

This paper showed how to use the genMID approach from [11] to derive a CDCL-based MID solver using the MP heuristic $\rho\sigma PMP^i$. This MID solver, called DimetheusMP, is able to perform Message Passing in order to provide biases for initializing VSIDS and phase-saving in order to guide the CDCL search. Using the means of parameter tuning, the paper studied the ability of DimetheusMP to solve crafted and random formulas, and determined good settings for the MP related parameters of DimetheusMP. The resulting performance is then compared to the performance of the two CDCL-only solvers Lingeling and DimetheusJW. The latter solver performs the exact same CDCL search as DimetheusMP but uses Jeroslow-Wang in order to initialize VSIDS and phase-saving. A substantial empirical study was performed in order to compare DimetheusMP and DimetheusJW. The results of this study allow to draw several conclusions.

First, the results clearly indicate that the application of the MID approach in conjunction with $\rho\sigma PMP^i$ gives a clear performance benefit. The performance of DimetheusMP is never worse than that of DimetheusJW, which is a clear indication that (despite the increased computational effort needed to compute $\rho\sigma PMP^i$ in comparison to JW) the application of MP is at least not harmful.

The DimetheusMP solver significantly outperforms DimetheusJW on both satisfiable and unsatisfiable crafted formulas, which implies that the application of MP on crafted formulas can be very beneficial. This result is somewhat surprising, as MP was believed to be helpful to solve only large size uniform random formulas. Furthermore, the results show that DimetheusMP significantly outperforms DimetheusJW on uniform random formulas. This implies that it is possible to use the CDCL paradigm to solve large random $k$-SAT formulas, which is surprising since CDCL has, in general, a comparatively poor performance on these types of formulas.

Second, the results clearly indicate that the good performance of DimetheusMP on crafted formulas requires the L2 MP heuristic $\rho\sigma PMP^i$. No other product-
based MP heuristics allows to achieve this performance on its own. This underlines the importance of the flexibility that this new interpolation-based MP heuristic provides. The good performance of DimetheusMP on uniform random formulas does, however, rely on the L1 interpolation $\rho_{SP}$. This heuristic interpolates between the basic BP and SP. The identified parameter settings show a strong tendency towards SP. This supports the statement that SP is indeed well suited to solve uniform random k-SAT formulas.

Third, even though $\rho_{\sigma PMP}$ can enforce the convergence, the results provided by the parameter tuning indicate that it is not necessarily helpful to do so in practice. On one hand, the parameter settings clearly show that enforcing convergence is never helpful on uniform random k-SAT formulas. On the other hand, the ability to gradually adapt convergence is crucial for the performance on crafted formulas.

So far, no empirical data has been gathered to estimate the usefulness of the CDCL-based MID approach using $\rho_{\sigma PMP}$ for solving application formulas. Gathering this data is definitely a matter of future work.

Furthermore, the presented MID solver characterizes a lightweight integration of MP with CDCL, that only influences the initialization of VSIDS and phase-saving. In theory, it is possible to use the biases provided by MP to also influence other aspects of the CDCL search. For example, it is thinkable to compute a bias-based MP impact of a learned clause, and then use this impact value to perform clause database maintenance. Here, one could focus on removing non-reason clauses that, by themselves, have only a very limited impact on the Message Passing. The fundamental assumption here is that a clause must have a substantial impact on MP to either help estimating a satisfying assignment, or strongly enforce variable assignments to help prune the search space. Determining whether such assumptions are reasonable in practice is another matter of future work.

Furthermore, our empirical study does provide MP parameter settings to adapt the CDCL-based MID approach to a wide variety of crafted and random formulas. However, it is not yet clear if the MP behavior characterized by the parameter settings can, in any way, be linked to specific attributes of the respective CNF formulas. It is therefore necessary to investigate whether the presence or absence of a specific formula attribute already implies a specific MP behavior to achieve good results for solving the respective formula.

Computing CNF attributes in order to classify the formulas and guide a decision for what type of solver is best suited to solve the formula is, by itself, no new idea. In fact, the classification is an integral part of all portfolio solvers [17]. However, in conjunction with the CDCL-based MID approach, one does not need to select a specific solver – the solver used for the search is always the same. The selection merely needs to pick a specific parameter setting for $\rho, \sigma, \text{and } p$ in order to adapt the MID behavior of the solver to be best suited for solving a specific formula. Investigating on the possibilities of CNF classifications in conjunction with parameter adaption for MID solvers is therefore an additional matter of future work.
References

7. bwGRiD, www.bw-grid.de, member of the German D-Grid initiative, funded by the Ministry for Education and Research, Germany.
10. O. Gableske, Dimetheus SAT solver v1.6, https://www.gableske.net/dimetheus