SCIENCE CHINA
Information Sciences

• LETTER •

July 2021, Vol. 64 179102:1–179102:3
https://doi.org/10.1007/s11432-018-9877-x

Revisiting the efficacy of weak consistencies: a study of forward checking

Zhe LI[1,2], Zhezhou YU[1,2], Hongbo LI[3], Jinsong GUO[4] & Zhanshan LI[1,2*]

[1]*Key Laboratory for Symbol Computation and Knowledge Engineering of National Education Ministry, Changchun* 130012*, China;*
[2]*College of Computer Science and Technology, Jilin University, Changchun* 130012*, China;*
[3]*College of Information Science and Technology, Northeast Normal University, Changchun* 130024*, China;*
[4]*Department of Computer Science, University of Oxford, Oxford* OX1 2JD*, UK*

Received 22 October 2018/Revised 27 February 2019/Accepted 8 April 2019/Published online 27 October 2020

Citation Li Z, Yu Z Z, Li H B, et al. Revisiting the efficacy of weak consistencies: a study of forward checking. Sci China Inf Sci, 2021, 64(7): 179102, https://doi.org/10.1007/s11432-018-9877-x

Dear editor,

Arc consistency (AC) [1] is currently the most popular local consistency for solving constraint satisfaction problems (CSP). To solve CSPs more efficiently, extensive work [2–4] has been done to find local consistencies that perform better than AC. Efforts have primarily been focused on consistencies that are stronger than AC. Even though a stronger consistency is able to eliminate more inconsistent values than AC, enforcing a stronger consistency is more expensive. It is never easy to determine the optimal tradeoff between a consistency's pruning ability and its propagation cost. In recent years, numerous stronger local consistencies, such as max-restricted path consistency (maxRPC) [2], partition-one-ac (POAC) [3], neighborhood singleton arc consistency (NSAC) and restricted NSAC (RNSAC) [4] have been shown to be competitive with AC. These stronger consistencies have been shown to outperform AC on a number of problems.

In this study, we explore a different direction compared with the existing studies. Instead of investigating strong consistencies, we revisit the efficacy of weak consistencies. First, we propose algorithms based on bitwise operations so as to boost the efficiency of propagation. Then, we propose a stronger version of forward checking consistency (FCC), called enhanced forward checking consistency (EFCC), the pruning ability of which lies between those of FCC's and AC's. In the propagation triggered by the instantiation of $(x_i,\ a)$, unlike FCC, which merely eliminates values from variables directly constrained by x_i, EFCC eliminates any values that should be eliminated via the process of enforcing arc consistency on the sub-network $\mathcal{P}|_{x_i=a}$. The proposed EFCC algorithm has the same time complexity as FCC. Our extensive experiments on various types of problems show that the search algorithms FC^{bit} and EFC^{bit}, maintaining FCC and EFCC, respectively, can outperform AC3^{bit} and other stronger consistency algorithms in a fairly large number of instances. These results should inspire researchers to revisit the efficacy of weak consistencies.

Additional information. First, we propose a new consistency called enhanced forward checking consistency. Then, we briefly analyze its pruning strength. A lack of space precludes discussing constraint satisfying problems (CSPs) in detail; however, the basic related concepts are found in [5]. Given a constraint network $\mathcal{P} = \langle \mathcal{X},\ \mathcal{D},\ \mathcal{C}\rangle$ and a value a in $D(x_i)$ for some $x_i \in \mathcal{X}$, we use $\text{AC}(\mathcal{P}|_{x_i=a})$ to denote the procedure of restricting $D(x_i)$ to $\{a\}$ and then applying arc consistency to $\mathcal{P}|_{x_i=a}$. The set of values eliminated via this procedure is denoted on $\mathcal{E}_{\text{AC}(\mathcal{P}|_{x_i=a})}$ (for simplicity, the shorter form of $\mathcal{E}_{\text{AC}(x_i,a)}$ is adopted hereafter).

Definition 1 (Enhanced forward checking consistency)**.** Given a constraint network $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}\rangle$, $\mathcal{P}|_{\mathcal{I}}$ denotes a sub-network that has been derived from $\mathcal{P}$ by an instantiation $\mathcal{I}$ over a subset $\mathcal{Y} \subset \mathcal{X}$.

• A value (x_j, b) is enhanced forward checking consistent iff for each variable $x_i \in \mathcal{Y}$, $(x_j, b) \notin \mathcal{E}_{\text{AC}(x_i,\mathcal{I}(x_i))}$, where $\mathcal{I}(x_i)$ is the instantiated value of x_i.

• A variable x_j is enhanced forward checking consistent iff $D(x_j) \neq \emptyset$ and all the values in $D(x_j)$ are enhanced forward checking consistent.

• $\mathcal{P}|_{\mathcal{I}}$ is enhanced forward checking consistent iff all the variables in $\mathcal{P}|_{\mathcal{I}}$ are enhanced forward checking consistent.

In other words, $\mathcal{P}|_{\mathcal{I}}$ is enhanced forward checking consistent iff (i) for each variable $x_i \in \mathcal{Y}$, all the values in $\mathcal{E}_{\text{AC}(x_i,\mathcal{I}(x_i))}$ have been removed; and (ii) for each variable $x_i \in \mathcal{X}$, $D(x_i) \neq \emptyset$. To determine whether a network $\mathcal{P}|_{\mathcal{I}}$ satisfies EFCC, one has to check whether the values in $\mathcal{E}_{\text{AC}(x_i,\mathcal{I}(x_i))}$ derived from the original network have been removed from the current network.

To study the pruning ability of EFCC, the following proposition is presented. Owing to limited space, a full and detailed proof is provided in the Appendix.

Proposition 1. On a network with at least one variable being instantiated, AC is strictly stronger than EFCC and EFCC is strictly stronger than FCC.

* Corresponding author (email: zslizsli@163.com)

Here, we propose bitwise-operation based algorithms for maintaining FCC and EFCC during a search.

For the constraint network it works on, $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, the following binary representation is adopted to store the domains and the supporting relationships between values. Each domain $D(x_i)$ is represented by a bit vector bitDom$[x_i]$ as in [1]. bitSup$[x_i][a][x_j]$ records all the values supporting (x_i, a) in $D(x_j)$. At each level k of the search tree, the variable domains are recorded in an array bitDomAtLevel$[k]$. At the beginning, i.e., at level 0 of the search tree, bitDomAtLevel$[k]$ represents all the values in $\mathcal{D}$ and every bit in bitDomAtLevel$[k]$ is set to 1 because no values have been removed yet. Next, we illustrate how FC$^{\text{bit}}$ applies FCC propagation after an instantiation.

After an instantiation of (x_i, a) at level k, FC removes all the values that are arc inconsistent with (x_i, a) from the domains of the variables neighboring x_i. To realize the propagation, as described in Algorithm 1, FC$^{\text{bit}}$ simply operates bitwise AND operations between bitDomAtLevel$[k-1]$ and the vectors representing the values that are arc consistent with (x_i, a). The logical AND result is stored in bitDomAtLevel of the current level (line 4). If the propagation succeeds, the index of currentLevel moves to the next level.

Algorithm 1 FC$^{\text{bit}}$/EFC$^{\text{bit}}(x_i, a)$

```
1: k ⇐ currentLevel;
   /* If the algorithm is FC^bit */
2: for each variable x_j constrained by x_i do
   /* If the algorithm is EFC^bit */
3: for each variable x_j ∈ X do
4:    bitDomAtLevel[k][x_j] ⇐ bitSup[x_i][a][x_j] &
      bitDomAtLevel[k − 1][x_j];
5:    if bitDomAtLevel[k][x_j] = 0 then
6:       return false;
7:    end if
8: end for
9: currentLevel ⇐ currentLevel + 1;
10: return true;
```

Next, we illustrate how to adjust FC$^{\text{bit}}$ to EFC$^{\text{bit}}$, which is the search algorithm maintaining EFCC during the search. Unlike FC$^{\text{bit}}$, which uses bitSup$[x_i][a][x_j]$ to represent which values are arc consistent with (x_i, a), EFC$^{\text{bit}}$ uses bitSup$[x_i][a][x_j]$ to indicate whether the values of x_j survive the check AC$(\mathcal{P}|_{x_i=a})$ or not (i.e., to indicate whether the values belong to $\mathcal{E}_{\text{AC}(x_i,a)}$).

EFC$^{\text{bit}}$ first employs a preprocessing procedure to generate $\mathcal{E}_{\text{AC}(x_i,a)}$ for each value (x_i, a) by running AC$(\mathcal{P}|_{x_i=a})$ for each (x_i, a). For a variable constrained by x_i, if a value (x_j, b) belongs to $\mathcal{E}_{\text{AC}(x_i,a)}$, then the corresponding bit of (x_j, b) in bitSup$[x_i][a][x_j]$ will be set to 0. Recall that EFCC is defined not only over variables constrained by x_i but also at the scope of the entire sub-network $\mathcal{P}|_{x_i=a}$. For any variable x_k not directly constrained by x_i, EFC$^{\text{bit}}$ also creates a bit array bitSup$[x_i][a][x_k]$ to record the values of x_k that pass the check AC$(\mathcal{P}|_{x_i=a})$. Note that the preprocessing step does not involve establishing singleton arc consistency (SAC) in $\mathcal{P}$but running AC$(\mathcal{P}|_{x_i=a})$ once for each value (x_i, a) in $\mathcal{P}$. If AC$(\mathcal{P}|_{x_i=a})$ fails, the value will be deleted prior to the search. With all the bitSups, we run Algorithm 1 to establish EFCC after each assignment. The difference is that FC$^{\text{bit}}$ only checks variables directly constrained by x_i, while EFC$^{\text{bit}}$ checks all the variables.

Proposition 2. In a network with at least one variable being instantiated, AC is strictly stronger than EFCC and EFCC is strictly stronger than FCC.

Proposition 3. The space complexity of FC$^{\text{bit}}$ is $O(n^2d + ed^2)$, where n, e, and d are the number of variables, number of constraints and maximum domain size, respectively.

The proofs of the above propositions are provided in the Appendix.

Because EFC$^{\text{bit}}$ uses the same propagation algorithm as FC$^{\text{bit}}$, the propagation time complexity of EFC$^{\text{bit}}$ is the same as that of FC$^{\text{bit}}$, which is $O(nd)$. As for the space complexity, EFC$^{\text{bit}}$ requires $O(n^2d^2)$ space for the bitsup arrays (because it maintains a bitSup for each pair of variables) and requires the same space for the bitDomAtLevel arrays as FC$^{\text{bit}}$; therefore, the space complexity of EFC$^{\text{bit}}$ is $O(n^2d^2)$.

Experimental details. We evaluated the performances of FCC and EFCC by comparing the search algorithms FC$^{\text{bit}}$ and EFC$^{\text{bit}}$ with AC3$^{\text{bit}}$ [1], lMaxRPC$^{\text{bit}}$ [2] and rRPC3 [6], which have stronger consistencies. To give readers a straightforward impression of the efficacies of the different local consistencies, we first evaluate the search algorithms using two common heuristics: dom and dom/wdeg. To provide a comprehensive evaluation, we sampled various CSPs from the XCSP3 website [7], including synthetic as well as and real-world problems. More than 1700 test instances were tested.

The runtime results are visualized in an intuitive way in Figure 1, which shows a cactus plot giving the number of instances solved per algorithm as the time limit increases, excluding the instances for which all the algorithms were over the time limit. As the x-axis value increases, i.e., as the problems become harder, FC$^{\text{bit}}$ and EFC$^{\text{bit}}$ solve more instances for any given time limit, with EFC$^{\text{bit}}$ performing the best.

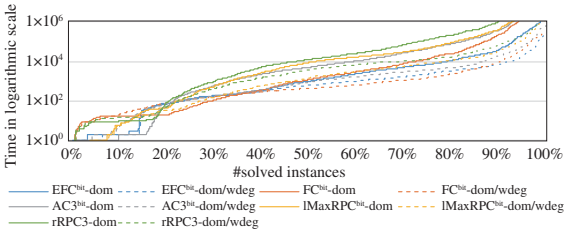

Figure 1 Percentage of instances solved per algorithm as the time allowed increases (cactus plot).

Even though it is already well known that arc consistency does not always work well, we are still excited to see that by adopting bitwise operations, FCC and EFCC outperform AC for many problems. This illustrates the potential of FCC/EFCC, thereby encouraging future work exploiting other consistencies weaker than AC. Please note that we are not claiming that FC$^{\text{bit}}$ and EFC$^{\text{bit}}$ are the most efficient CSP-solving techniques. Instead, we want to inspire an open-minded revisiting of local consistencies that are weaker than AC; we should reexamine their efficacies and exploit them if reasonable.

Conclusion. Extensive research has been conducted towards applying stronger consistencies to achieve a better performance than AC. In this study, we illustrated the feasibility approaching the topic from the opposite direction by revisiting classic forward checking. We proposed an FC$^{\text{bit}}$ algorithm that uses bitwise operations to decrease the propagation cost. In addition, to improve the pruning ability of forward checking consistency, we propose a stronger version, enhanced forward checking consistency. Our algorithm

for maintaining EFCC during a search, EFC^{bit}, is able to retain the same time complexity of propagation as FC^{bit}'s. Extensive experiments show that both FC^{bit} and EFC^{bit} outperform stronger consistencies on most of the problems we tested; this sufficiently illustrates the potential of exploiting weaker consistencies.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61672261, 61802056), Jilin Province Natural Science Foundation (Grant No. 20180101043JC), Jilin Province Development and Reform Commission (Grant No. 2019C053-9), and Jilin Provincial Key Laboratory of Big Data Intelligent Computing (Grant No. 20180622002JC).

Supporting information Appendixes A–C. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Lecoutre C, Vion J. Enforcing arc consistency using bitwise operations. Constraint Programm Lett, 2008, 2: 21-35
2 Guo J, Li Z, Zhang L, et al. MaxRPC algorithms based on bitwise operations. In: Proceedings of International Conference on Principles and Practice of Constraint Programming, Perugia, 2011. 373–384
3 Balafrej A, Bessiere C, Bouyakhf E H, et al. Adaptive singleton-based consistencies. In: Proceedings of AAAI'14, Québec City, 2014. 2601–2607
4 Paparrizou A, Stergiou K. On neighborhood singleton consistencies. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, 2017. 736–742
5 Bessiere C. Constraint propagation. Found Artif Intell, 2006, 2: 29–83
6 Stergiou K. Restricted path consistency revisited. In: Proceedings of International Conference on Principles and Practice of Constraint Programming, Cork, 2015. 419–428
7 Boussemart. XCSP3 benchmarks website. 2019. http://xcsp.org/series