Choco-solver

Charles Prud'homme, Jean-Guillaume Fages

The Journal of Open Source Software

Choco-solver: A Java library for constraint programming

Charles Prud'homme [1¶] **and Jean-Guillaume Fages**[2]

1 TASC, IMT-Atlantique, LS2N-CNRS, Nantes, France **2** COSLING S.A.S., Nantes, France ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @skadio
- @ozgurakgun

Submitted: 13 July 2022
Published: unpublished

Summary

Constraint Programming (CP) is a powerful programming paradigm for solving combinatorial search problems (Rossi et al., 2006). CP is at the intersection of artificial intelligence, computer science, operations research, and many other fields. One of the strengths of the paradigm is the wide variety of constraints it offers. CP is both a rich declarative language for describing combinatorial problems and a set of algorithms and techniques for solving them automatically.

`Choco-solver` is Java library for constraint programming which was created in the early 2000s. Since then, the library has evolved a great deal, but ease of use has always been a guiding principle in its development. The `Choco-solver` API is designed to reduce entry points to a minimum and thus simplifies modelling for users. The wide variety of constraints available allows the user to describe his problem as naturally as possible. The black-box approach to solving allows everyone to focus on modelling. However, `Choco-solver` is also open and modifiable. The implementation of new constraints (Ouellet & Quimper, 2022) or strategies for exploring the search space (Fages & Prud'Homme, 2017; Li et al., 2021) is therefore possible.

As a result, `Choco-solver` is used by the academics for teaching and research, on the other hand it is used by the industry to solve real-world problems.

CP in a nutshell

Constraint programming provides not only a declarative way for users to describe discrete problems, but also techniques for solving them automatically. In that sense, it is very close to integer linear programming or Boolean satisfaction but is distinguished from them with its high-level modeling language and expressiveness. Actually, one of the richness's of the paradigm lies in the wide variety of constraints it proposes, which are also central to the solving stage. Thus, the objective of CP is twofold: firstly to offer a rich declarative language to describe a combinatorial problem, and secondly to provide techniques for solving the problem automatically. In standard use, a user states a problem using variables, their domains (possible values for each variable), and constraints which are called predicates that must hold on the variables. The wide variety of constraints available allows the users to describe their problem as naturally as possible. Each constraint ensures that it holds, otherwise a propagator filters the values that prevent the satisfiability. It is the combination of the selected constraints that defines the problem. The problem is solved by alternating space reduction (usually by a depth-first search) and propagation, thus ensuring the completeness of the approach. This standard usage can be extended in different ways, for example, by hybridisation with local search, Boolean satisfiability, or linear programming techniques.

The Journal of Open Source Software

Statement of need

For constraint programming to be used successfully, it is essential to have a library that incorporates the latest advances in the field, while ensuring reliability, performance, and responsiveness. This was also the motivation for the creation of `Choco-solver`: Providing state-of-the-art algorithms and high resolution performance while offering ease of use and development, all in a free and open-source library.

Achievement

With 20 years of development, `Choco-solver` is now a stable, flexible, extensible, powerful, and user-friendly library. There is a community of users and contributors who actively participate in improving the library. In addition, `Choco-solver` relies on software quality standards (unit and performance tests, continuous integration, code review, etc.) and frequent updates are made. Finally, the choice of Java as programming language makes the integration of the library simple into both academic and industrial projects.

Features and Functionality

Modeling

`Choco-solver` comes with the commonly used types of variables: Integer variables, Boolean variables, set variables (Gervet, 1997), graph variables (Dooms et al., 2005; Fages, 2015), and real variables. Views (Justeau-Allaire & Prud'homme, 2022; Schulte & Tack, 2005) but also arithmetical, relational and logical expressions are supported.

Up to 100 constraints are provided from classic ones, such as arithmetical constraints, to must-have global constraints such as *AllDifferent* (Régin, 1994) or *Cumulative* (Aggoun & Beldiceanu, 1993), and include less common even though useful ones such as *Tree* (Beldiceanu et al., 2005) or *StableKeySort* (Beldiceanu et al., 2015). In many cases, the `Choco-solver` API provides various options in addition to the default signature – corresponding to a robust implementation – of a constraint. This allows users to experiment alternative approaches and tune the model to its instance. The users may also pick some existing propagators to compose a new constraint or create their own one in a straightforward way by implementing a filtering algorithm and a satisfaction checker. Many models are available on the Choco-solver website as modelling tutorials.

Solving

`Choco-solver` has been carefully designed to offer wide range of resolution configurations and good solving performances. Backtrackable primitives and structures are based on trailing (Aggoun & Beldiceanu, 1990; Reischuk et al., 2009). The propagation engine deals with seven priority levels (Prud'homme, Lorca, Douence, et al., 2014; Schulte & Stuckey, 2008) and manages either fine or coarse grain events which enables to get efficient incremental constraint propagators.

The search algorithm relies on three components *Propagate*, *Learn*, and *Move* (Jussien & Lhomme, 2002). Such a generic search algorithm is then instantiated to depth-first search, large neighbourhood search (Prud'homme, Lorca, & Jussien, 2014; Shaw, 1998), limited discrepancy search (Harvey & Ginsberg, 1995), depth-bounded discrepancy search (Walsh, 1997) or hybrid breadth-first search (Allouche et al., 2015).

The search process can also be greatly improved by various built-in search strategies such as *dom/wdeg* (Boussemart et al., 2004) and its *ca-cd* variant (Wattez et al., 2019), *activity-based search* (Michel & Van Hentenryck, 2012), *failure-based searches* (Li et al., 2021), *bound-impact value selector* (Fages & Prud'Homme, 2017), *first-fail* (Haralick & Elliott, 1979), and many

others. Standard restart policies are also available, to take full advantage of the learning strategies. Problem-adapted search strategies are also supported.

One can solve a problem by

- simply checking satisfaction
- finding one or all solutions
- optimizing one or more objectives
- solving on one or more thread.

The search process itself is observable and extensible.

Community tools integration

Several useful extra features are also available such as parsers to XCSP3 format and MiniZinc format. `Choco-solver` is also embedded in PyCSP3, a Python library for modeling and solving combinatorial constrained problems. In addition to offering alternatives to modelling in Java, it also allows participation in the two major constraint solver competitions : MiniZinc Challenge and XCSP3 Competition.

Finally, although it is originally designed to solve discrete mathematical problems, `Choco-solver` also supports natively real variables and constraints, and relies on Ibex-lib to solve the continuous part of the problems (Fages et al., 2013). Equally, a Boolean satisfaction solver (based on MiniSat) is integrated to offer better performance on logical constraints.

These aspects consolidate the place of `Choco-solver` as an important tool in the CP community.

Note that there are a couple of other Java constraint solvers of equivalent maturity, like JaCoP and ACE. Although the performances of these tools are direclty comparable, they are mainly distinguished by the functionalities in terms of modelling and resolution. Among the most noteworthy, `Choco-solver` allows integrating constraints which are based on graph variables or real variables, or it can parse both MiniZinc and XCSP3 input files.

Industrial use

`Choco-solver` is used by the industry to solve many real-world problems, such as cryptanalysis (Delaune et al., 2021), construction planning (Cañizares et al., 2022), automated testing and debugging (Le et al., 2021), scheduling (Lorca et al., 2016), level design (Smith et al., 2011), placement service (Ait Salaht et al., 2019) and many others. In the Railway industry, `Choco-solver` is used to optimize the rail traffic of French train stations, on a daily basis. It is also used at a higher level to run simulations for capacity and maintenance planning. The underlying mathematical problems, involving multi-objective functions with millions of variables and constraints, are solved within seconds by the solver. In the defense sector, `Choco-solver` is used for various applications. One publicly known is the long-term maintenance planning of the Mirage 2000 fleet (Grazzini, 2019). This planning and scheduling problem includes various capacity constraints, load balancing, and mission covering in an over-constrained environment. A fleet of hundred aircraft can be planned by `Choco-solver` for the next fifteen years within a few minutes. Another type of industrial application of `Choco-solver` is Configuration (Charpentier et al., 2021) where the solver is used to solve dynamical constraint models. The underlying mathematical problems are generally simpler than the ones in planning applications and an optimal result is expected within milliseconds. This occurs in quotation systems for sales automation, but also design automation, and system configuration.

In most of those cases, experts set advanced solution techniques, such as specific search strategies, Large Neighborhood Search or *ad hoc* global constraints, in order to improve their model. `Choco-solver` is flexible enough to allow such fine-tuning to tackle challenging problems. With the right approach, `Choco-solver` can come up with nearly optimal solutions in a very short time.

The Journal of Open Source Software

Acknowledgements

We acknowledge contributions from (in alphabetical order) Hadrien Cambazard, Arthur Godet, Fabien Hermenier, Narendra Jussien, Dimitri Justeau-Allaire, Tanguy Lapègue, Alexandre Lebrun, Jimmy Liang, Xavier Lorca, Arnaud Malapert, Guillaume Rochart, João Pedro Schmitt and Mohamed Wahbi.

References

Aggoun, A., & Beldiceanu, N. (1990). Time stamps techniques for the trailed data in constraint logic programming systems. In S. Bourgault & M. Dincbas (Eds.), *SPLT'90, 8ème séminaire programmation en logique, 16-18 mai 1990, trégastel, france* (pp. 487–510).

Aggoun, A., & Beldiceanu, N. (1993). Extending CHIP in order to Solve Complex Scheduling and Placement Problems. *Mathl. Comput. Modelling*, *17*(7), 57–73. https://doi.org/10.1016/0895-7177(93)90068-A

Ait Salaht, F., Desprez, F., Lebre, A., Prud'homme, C., & Abderrahim, M. (2019). Service placement in fog computing using constraint programming. *2019 IEEE International Conference on Services Computing (SCC)*, 19–27. https://doi.org/10.1109/SCC.2019.00017

Allouche, D., De Givry, S., Katsirelos, G., Schiex, T., & Zytnicki, M. (2015). Anytime hybrid best-first search with tree decomposition for weighted CSP. *CP 2015 - 21st International Conference on Principles and Practice of Constraint Programming*, 17 p. https://doi.org/10.1007/978-3-319-23219-5/_2

Beldiceanu, N., Carlsson, M., Flener, P., Lorca, X., Pearson, J., Petit, T., & Prud'Homme, C. (2015, October). A Modelling Pearl with Sortedness Constraints. *Global conference on artificial intelligence*. https://doi.org/10.29007/b4dz

Beldiceanu, N., Flener, P., & Lorca, X. (2005). The tree constraint. In R. Barták & M. Milano (Eds.), *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, second international conference, CPAIOR 2005, prague, czech republic, may 30 - june 1, 2005, proceedings* (Vol. 3524, pp. 64–78). Springer. https://doi.org/10.1007/11493853/_7

Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 146–150.

Cañizares, P. C., Estévez-Martín, S., & Núñez, M. (2022). SINPA: SupportINg the automation of construction PlAnning. *Expert Systems with Applications*, *190*, 116149. https://doi.org/10.1016/j.eswa.2021.116149

Charpentier, A., Fages, J.-G., & Lapègue, T. (2021). COSLING configurator. *ConfWS*.

Delaune, S., Derbez, P., Huynh, P., Minier, M., Mollimard, V., & Prud'homme, C. (2021). Efficient methods to search for best differential characteristics on SKINNY. In K. Sako & N. O. Tippenhauer (Eds.), *Applied cryptography and network security* (pp. 184–207). Springer International Publishing. https://doi.org/10.1007/978-3-030-78375-4_8

Dooms, G., Deville, Y., & Dupont, P. (2005). CP(Graph): Introducing a Graph Computation Domain in Constraint Programming. *Principles and Practice of Constraint Programming - CP 2005*, 211–225. https://doi.org/10.1007/11564751_18

Fages, J.-G. (2015). On the use of graphs within constraint-programming. *Constraints*, *20*(4), 498–499. https://doi.org/10.1007/s10601-015-9223-9

Fages, J.-G., Chabert, G., & Prud'Homme, C. (2013). Combining finite and continuous solvers Towards a simpler solver maintenance. *The 19th International Conference on Principles and Practice of Constraint Programming*, TRICS'13 Workshop: Techniques foR Implementing Constraint programming Systems. https://hal.archives-ouvertes.fr/hal-00904069

Fages, J.-G., & Prud'Homme, C. (2017, November). Making the first solution good! *ICTAI 2017 29th IEEE International Conference on Tools with Artificial Intelligence*. https://doi.org/10.1109/ictai.2017.00164

Gervet, C. (1997). Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, *1*(3), 191–244. https://doi.org/10.1007/BF00137870

Grazzini, F. (2019). *Airbus and COSLING provide software solution optaforce for mirage 2000 maintenance*. https://www.airbus.com/en/newsroom/press-releases/2019-06-airbus-and-cosling-provide-software-solution-optaforce-for-mirage

Haralick, R. M., & Elliott, G. L. (1979). Increasing tree search efficiency for constraint satisfaction problems. *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 1*, 356–364. https://doi.org/10.1016/0004-3702(80)90051-X

Harvey, W. D., & Ginsberg, M. L. (1995). Limited discrepancy search. *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, 607–613. ISBN: 978-1-558-60363-9

Jussien, N., & Lhomme, O. (2002). *Unifying search algorithms for CSP* (Research Report No. RR0203). EMN.

Justeau-Allaire, D., & Prud'homme, C. (2022). Global domain views for expressive and cross-domain constraint programming. *Constraints An Int. J.*, *27*(1), 1–7. https://doi.org/10.1007/s10601-021-09324-7

Le, V.-M., Felfernig, A., Tran, T. N. T., Atas, M., Uta, M., Benavides, D., & Galindo, J. (2021). DirectDebug: A software package for the automated testing and debugging of feature models. *Software Impacts*, *9*, 100085. https://doi.org/10.1016/j.simpa.2021.100085

Li, H., Yin, M., & Li, Z. (2021). Failure based variable ordering heuristics for solving CSPs (short paper). In L. D. Michel (Ed.), *27th international conference on principles and practice of constraint programming, CP 2021, montpellier, france (virtual conference), october 25-29, 2021* (Vol. 210, pp. 9:1–9:10). Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.CP.2021.9

Lorca, X., Prud'homme, C., Questel, A., & Rottembourg, B. (2016). Using constraint programming for the urban transit crew rescheduling problem. In M. Rueher (Ed.), *Principles and practice of constraint programming* (pp. 636–649). Springer International Publishing. https://doi.org/10.1007/978-3-319-44953-1_40

Michel, L., & Van Hentenryck, P. (2012). Activity-based search for black-box constraint programming solvers. In N. Beldiceanu, N. Jussien, & É. Pinson (Eds.), *Integration of AI and OR techniques in contraint programming for combinatorial optimzation problems* (pp. 228–243). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29828-8_15

Ouellet, Y., & Quimper, C.-G. (2022). A MinCumulative resource constraint. In P. Schaus (Ed.), *Integration of constraint programming, artificial intelligence, and operations research* (pp. 318–334). Springer International Publishing. https://doi.org/10.1007/978-3-031-08011-1_21

Prud'homme, C., Lorca, X., Douence, R., & Jussien, N. (2014). Propagation engine prototyping with a domain specific language. *Constraints An Int. J.*, *19*(1), 57–76. https://doi.org/10.1007/s10601-013-9151-5

Prud'homme, C., Lorca, X., & Jussien, N. (2014). Explanation-based large neighborhood search. *Constraints*, *19*(4), 339–379. https://doi.org/10.1007/s10601-014-9166-6

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, 362–367. ISBN: 0262611023

Reischuk, R. M., Schulte, C., Stuckey, P. J., & Tack, G. (2009). Maintaining state in propagation solvers. In I. P. Gent (Ed.), *Principles and practice of constraint programming - CP 2009, 15th international conference, CP 2009, lisbon, portugal, september 20-24, 2009, proceedings* (Vol. 5732, pp. 692–706). Springer. https://doi.org/10.1007/978-3-642-04244-7/_54

Rossi, F., Beek, P. van, & Walsh, T. (Eds.). (2006). *Handbook of constraint programming* (Vol. 2). Elsevier. ISBN: 978-0-444-52726-4

Schulte, C., & Stuckey, P. J. (2008). Efficient constraint propagation engines. *ACM Trans. Program. Lang. Syst.*, *31*(1), 2:1–2:43. https://doi.org/10.1145/1452044.1452046

Schulte, C., & Tack, G. (2005). Views and iterators for generic constraint implementations. In P. van Beek (Ed.), *Principles and practice of constraint programming - CP 2005, 11th international conference, CP 2005, sitges, spain, october 1-5, 2005, proceedings* (Vol. 3709, pp. 817–821). Springer. https://doi.org/10.1007/11564751/_71

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. J. Maher & J.-F. Puget (Eds.), *Principles and practice of constraint programming - CP98, 4th international conference, pisa, italy, october 26-30, 1998, proceedings* (Vol. 1520, pp. 417–431). Springer. https://doi.org/10.1007/3-540-49481-2/_30

Smith, G., Whitehead, J., & Mateas, M. (2011). Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 201–215. https://doi.org/10.1109/TCIAIG.2011.2159716

Walsh, T. (1997). Depth-bounded discrepancy search. *In Proceedings of IJCAI-97*, 1388–1393.

Wattez, H., Lecoutre, C., Paparrizou, A., & Tabary, S. (2019). Refining constraint weighting. *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, 71–77. https://doi.org/10.1109/ICTAI.2019.00019