



HAL
open science

Choco-solver

Charles Prud'homme, Jean-Guillaume Fages

► **To cite this version:**

Charles Prud'homme, Jean-Guillaume Fages. Choco-solver. Journal of Open Source Software, 2022, 7 (78), pp.4708. 10.21105/joss.04708 . hal-03932507

HAL Id: hal-03932507

<https://hal.science/hal-03932507v1>

Submitted on 10 Jan 2023


HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License




1 Choco-solver: A Java library for constraint 2 programming

3 Charles Prud'homme ¹¶ and Jean-Guillaume Fages²

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

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Mehmet Hakan Satman](#) 

Reviewers:

- [@skadio](#)
- [@ozgurakgun](#)

Submitted: 13 July 2022

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

6 Summary

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

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

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

22 CP in a nutshell

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

39 Statement of need

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

45 Achievement

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

52 Features and Functionality

53 Modeling

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

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

68 Solving

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

75 The search algorithm relies on three components *Propagate*, *Learn*, and *Move* ([Jussien &
76 Lhomme, 2002](#)). Such a generic search algorithm is then instantiated to depth-first search,
77 large neighbourhood search ([Prud'homme, Lorca, & Jussien, 2014](#); [Shaw, 1998](#)), limited
78 discrepancy search ([Harvey & Ginsberg, 1995](#)), depth-bounded discrepancy search ([Walsh,
79 1997](#)) or hybrid breadth-first search ([Allouche et al., 2015](#)).

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

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

86 One can solve a problem by

- 87 ▪ simply checking satisfaction
- 88 ▪ finding one or all solutions
- 89 ▪ optimizing one or more objectives
- 90 ▪ solving on one or more thread.

91 The search process itself is observable and extensible.

92 Community tools integration

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

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

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

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

108 Industrial use

109 Choco-solver is used by the industry to solve many real-world problems, such as cryptanalysis
110 ([Delaune et al., 2021](#)), construction planning ([Cañizares et al., 2022](#)), automated testing
111 and debugging ([Le et al., 2021](#)), scheduling ([Lorca et al., 2016](#)), level design ([Smith et al.,](#)
112 [2011](#)), placement service ([Ait Salaht et al., 2019](#)) and many others. In the Railway industry,
113 Choco-solver is used to optimize the rail traffic of French train stations, on a daily basis. It
114 is also used at a higher level to run simulations for capacity and maintenance planning. The
115 underlying mathematical problems, involving multi-objective functions with millions of variables
116 and constraints, are solved within seconds by the solver. In the defense sector, Choco-solver
117 is used for various applications. One publicly known is the long-term maintenance planning of
118 the Mirage 2000 fleet ([Grazzini, 2019](#)). This planning and scheduling problem includes various
119 capacity constraints, load balancing, and mission covering in an over-constrained environment.
120 A fleet of hundred aircraft can be planned by Choco-solver for the next fifteen years within
121 a few minutes. Another type of industrial application of Choco-solver is Configuration
122 ([Charpentier et al., 2021](#)) where the solver is used to solve dynamical constraint models. The
123 underlying mathematical problems are generally simpler than the ones in planning applications
124 and an optimal result is expected within milliseconds. This occurs in quotation systems for
125 sales automation, but also design automation, and system configuration.

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

131 **Acknowledgements**

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

136 **References**

- 137 Aggoun, A., & Beldiceanu, N. (1990). Time stamps techniques for the trailed data in constraint
138 logic programming systems. In S. Bourgault & M. Dincbas (Eds.), *SPLT'90, 8^{ème} séminaire*
139 *programmation en logique, 16-18 mai 1990, trégastel, france* (pp. 487–510).
- 140 Aggoun, A., & Beldiceanu, N. (1993). Extending CHIP in order to Solve Complex Scheduling
141 and Placement Problems. *Mathl. Comput. Modelling*, 17(7), 57–73. [https://doi.org/10.1016/0895-7177\(93\)90068-A](https://doi.org/10.1016/0895-7177(93)90068-A)
142
- 143 Ait Salaht, F., Desprez, F., Lebre, A., Prud'homme, C., & Abderrahim, M. (2019). Service
144 placement in fog computing using constraint programming. *2019 IEEE International*
145 *Conference on Services Computing (SCC)*, 19–27. <https://doi.org/10.1109/SCC.2019.00017>
146
- 147 Allouche, D., De Givry, S., Katsirelos, G., Schiex, T., & Zytnicki, M. (2015). Anytime hybrid
148 best-first search with tree decomposition for weighted CSP. *CP 2015 - 21st International*
149 *Conference on Principles and Practice of Constraint Programming*, 17 p. https://doi.org/10.1007/978-3-319-23219-5/_2
150
- 151 Beldiceanu, N., Carlsson, M., Flener, P., Lorca, X., Pearson, J., Petit, T., & Prud'Homme, C.
152 (2015, October). A Modelling Pearl with Sortedness Constraints. *Global conference on*
153 *artificial intelligence*. <https://doi.org/10.29007/b4dz>
- 154 Beldiceanu, N., Flener, P., & Lorca, X. (2005). The tree constraint. In R. Barták &
155 M. Milano (Eds.), *Integration of AI and OR techniques in constraint programming for*
156 *combinatorial optimization problems, second international conference, CPAIOR 2005,*
157 *prague, czech republic, may 30 - june 1, 2005, proceedings* (Vol. 3524, pp. 64–78).
158 Springer. https://doi.org/10.1007/11493853/_7
- 159 Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search
160 by weighting constraints. *Proceedings of the 16th European Conference on Artificial*
161 *Intelligence, ECAI'2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004,*
162 *Valencia, Spain, August 22-27, 2004*, 146–150.
- 163 Cañizares, P. C., Estévez-Martín, S., & Núñez, M. (2022). SINPA: SupportINg the automation
164 of construction PIAnning. *Expert Systems with Applications*, 190, 116149. <https://doi.org/10.1016/j.eswa.2021.116149>
165
- 166 Charpentier, A., Fages, J.-G., & Lapègue, T. (2021). COSLING configurator. *ConfWS*.
- 167 Delaune, S., Derbez, P., Huynh, P., Minier, M., Mollimard, V., & Prud'homme, C. (2021).
168 Efficient methods to search for best differential characteristics on SKINNY. In K. Sako
169 & N. O. Tippenhauer (Eds.), *Applied cryptography and network security* (pp. 184–207).
170 Springer International Publishing. https://doi.org/10.1007/978-3-030-78375-4_8
- 171 Dooms, G., Deville, Y., & Dupont, P. (2005). CP(Graph): Introducing a Graph Computation
172 Domain in Constraint Programming. *Principles and Practice of Constraint Programming -*
173 *CP 2005*, 211–225. https://doi.org/10.1007/11564751_18
- 174 Fages, J.-G. (2015). On the use of graphs within constraint-programming. *Constraints*, 20(4),
175 498–499. <https://doi.org/10.1007/s10601-015-9223-9>

- 176 Fages, J.-G., Chabert, G., & Prud'Homme, C. (2013). Combining finite and continuous solvers
177 Towards a simpler solver maintenance. *The 19th International Conference on Principles and*
178 *Practice of Constraint Programming*, TRICS'13 Workshop: Techniques foR Implementing
179 Constraint programming Systems. <https://hal.archives-ouvertes.fr/hal-00904069>
- 180 Fages, J.-G., & Prud'Homme, C. (2017, November). Making the first solution good! *ICTAI*
181 *2017 29th IEEE International Conference on Tools with Artificial Intelligence*. <https://doi.org/10.1109/ictai.2017.00164>
182
- 183 Gervet, C. (1997). Interval propagation to reason about sets: Definition and implementation
184 of a practical language. *Constraints*, 1(3), 191–244. <https://doi.org/10.1007/BF00137870>
- 185 Grazzini, F. (2019). *Airbus and COSLING provide software solution optaforce for*
186 *mirage 2000 maintenance*. [https://www.airbus.com/en/newsroom/press-releases/](https://www.airbus.com/en/newsroom/press-releases/2019-06-airbus-and-cosling-provide-software-solution-optaforce-for-mirage)
187 [2019-06-airbus-and-cosling-provide-software-solution-optaforce-for-mirage](https://www.airbus.com/en/newsroom/press-releases/2019-06-airbus-and-cosling-provide-software-solution-optaforce-for-mirage)
- 188 Haralick, R. M., & Elliott, G. L. (1979). Increasing tree search efficiency for constraint
189 satisfaction problems. *Proceedings of the 6th International Joint Conference on Artificial*
190 *Intelligence - Volume 1*, 356–364. [https://doi.org/10.1016/0004-3702\(80\)90051-X](https://doi.org/10.1016/0004-3702(80)90051-X)
- 191 Harvey, W. D., & Ginsberg, M. L. (1995). *Limited discrepancy search*. *Proceedings of*
192 *the 14th International Joint Conference on Artificial Intelligence - Volume 1*, 607–613.
193 ISBN: 978-1-558-60363-9
- 194 Jussien, N., & Lhomme, O. (2002). *Unifying search algorithms for CSP* (Research Report No.
195 RR0203). EMN.
- 196 Justeau-Allaire, D., & Prud'homme, C. (2022). Global domain views for expressive and
197 cross-domain constraint programming. *Constraints An Int. J.*, 27(1), 1–7. <https://doi.org/10.1007/s10601-021-09324-7>
198
- 199 Le, V.-M., Felfernig, A., Tran, T. N. T., Atas, M., Uta, M., Benavides, D., & Galindo, J. (2021).
200 DirectDebug: A software package for the automated testing and debugging of feature
201 models. *Software Impacts*, 9, 100085. <https://doi.org/10.1016/j.simpa.2021.100085>
- 202 Li, H., Yin, M., & Li, Z. (2021). Failure based variable ordering heuristics for solving CSPs
203 (short paper). In L. D. Michel (Ed.), *27th international conference on principles and*
204 *practice of constraint programming, CP 2021, montpellier, france (virtual conference),*
205 *october 25-29, 2021* (Vol. 210, pp. 9:1–9:10). Schloss Dagstuhl - Leibniz-Zentrum für
206 Informatik. <https://doi.org/10.4230/LIPIcs.CP.2021.9>
- 207 Lorca, X., Prud'homme, C., Questel, A., & Rottembourg, B. (2016). Using constraint
208 programming for the urban transit crew rescheduling problem. In M. Rueher (Ed.),
209 *Principles and practice of constraint programming* (pp. 636–649). Springer International
210 Publishing. https://doi.org/10.1007/978-3-319-44953-1_40
- 211 Michel, L., & Van Hentenryck, P. (2012). Activity-based search for black-box constraint
212 programming solvers. In N. Beldiceanu, N. Jussien, & É. Pinson (Eds.), *Integration of AI*
213 *and OR techniques in constraint programming for combinatorial optimization problems* (pp.
214 228–243). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29828-8_15
- 215 Ouellet, Y., & Quimper, C.-G. (2022). A MinCumulative resource constraint. In P. Schaus (Ed.),
216 *Integration of constraint programming, artificial intelligence, and operations research* (pp.
217 318–334). Springer International Publishing. [https://doi.org/10.1007/978-3-031-08011-1_](https://doi.org/10.1007/978-3-031-08011-1_21)
218 [21](https://doi.org/10.1007/978-3-031-08011-1_21)
- 219 Prud'homme, C., Lorca, X., Douence, R., & Jussien, N. (2014). Propagation engine prototyping
220 with a domain specific language. *Constraints An Int. J.*, 19(1), 57–76. [https://doi.org/10.](https://doi.org/10.1007/s10601-013-9151-5)
221 [1007/s10601-013-9151-5](https://doi.org/10.1007/s10601-013-9151-5)
- 222 Prud'homme, C., Lorca, X., & Jussien, N. (2014). Explanation-based large neighborhood
223 search. *Constraints*, 19(4), 339–379. <https://doi.org/10.1007/s10601-014-9166-6>

- 224 Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. *Proceed-*
225 *ings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, 362–367.
226 ISBN: 0262611023
- 227 Reischuk, R. M., Schulte, C., Stuckey, P. J., & Tack, G. (2009). Maintaining state in
228 propagation solvers. In I. P. Gent (Ed.), *Principles and practice of constraint programming*
229 *- CP 2009, 15th international conference, CP 2009, lisbon, portugal, september 20-*
230 *24, 2009, proceedings (Vol. 5732, pp. 692–706)*. Springer. [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-642-04244-7/_54)
231 [978-3-642-04244-7/_54](https://doi.org/10.1007/978-3-642-04244-7/_54)
- 232 Rossi, F., Beek, P. van, & Walsh, T. (Eds.). (2006). *Handbook of constraint programming*
233 (Vol. 2). Elsevier. ISBN: 978-0-444-52726-4
- 234 Schulte, C., & Stuckey, P. J. (2008). Efficient constraint propagation engines. *ACM Trans.*
235 *Program. Lang. Syst.*, 31(1), 2:1–2:43. <https://doi.org/10.1145/1452044.1452046>
- 236 Schulte, C., & Tack, G. (2005). Views and iterators for generic constraint implementations.
237 In P. van Beek (Ed.), *Principles and practice of constraint programming - CP 2005, 11th*
238 *international conference, CP 2005, sitges, spain, october 1-5, 2005, proceedings (Vol. 3709,*
239 *pp. 817–821)*. Springer. https://doi.org/10.1007/11564751/_71
- 240 Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle
241 routing problems. In M. J. Maher & J.-F. Puget (Eds.), *Principles and practice of*
242 *constraint programming - CP98, 4th international conference, pisa, italy, october 26-*
243 *30, 1998, proceedings (Vol. 1520, pp. 417–431)*. Springer. [https://doi.org/10.1007/](https://doi.org/10.1007/3-540-49481-2/_30)
244 [3-540-49481-2/_30](https://doi.org/10.1007/3-540-49481-2/_30)
- 245 Smith, G., Whitehead, J., & Mateas, M. (2011). Tanagra: Reactive planning and constraint
246 solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence*
247 *and AI in Games*, 3(3), 201–215. <https://doi.org/10.1109/TCIAIG.2011.2159716>
- 248 Walsh, T. (1997). Depth-bounded discrepancy search. In *Proceedings of IJCAI-97*, 1388–1393.
- 249 Watez, H., Lecoutre, C., Paparrizou, A., & Tabary, S. (2019). Refining constraint weighting.
250 *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019,*
251 *Portland, OR, USA, November 4-6, 2019, 71–77*. [https://doi.org/10.1109/ICTAI.2019.](https://doi.org/10.1109/ICTAI.2019.00019)
252 [00019](https://doi.org/10.1109/ICTAI.2019.00019)