# Dependency-Curated Large Neighbourhood Search

**Frej Knutar Lewander**[1]
Uppsala University, Department of Information Technology, Uppsala, Sweden

**Pierre Flener**
Uppsala University, Department of Information Technology, Uppsala, Sweden

**Justin Pearson**
Uppsala University, Department of Information Technology, Uppsala, Sweden

## Abstract

In large neighbourhood search (LNS), an incumbent initial solution is incrementally improved by selecting a subset of the variables, called the *freeze set*, and fixing them to their values in the incumbent solution, while a value for each remaining variable is found and assigned via solving (such as constraint programming-style propagation and search). Much research has been performed on finding generic and problem-specific LNS selection heuristics that select freeze sets that lead to high-quality solutions. In constraint-based local search (CBLS), the relations between the variables via the constraints are fundamental and well-studied, as they capture dependencies of the variables. In this paper, we apply these ideas from CBLS to the LNS context, presenting the novel dependency curation scheme, which exploits them to find a low-cardinality set of variables that the freeze set of any selection heuristic should be a subset of. The scheme often improves the overall performance of generic selection heuristics. Even when the scheme is used with a naïve generic selection heuristic that selects random freeze sets, the performance is competitive with more elaborate generic selection heuristics.

**2012 ACM Subject Classification** Computing methodologies → Heuristic function construction

**Keywords and phrases** Combinatorial Optimisation, Large Neighbourhood Search (LNS), Constraint-Based Local Search (CBLS)

**Category** Extended Abstract

**Related Version** *Full Version*: 10.4230/LIPIcs.CP.2025.16

**Supplementary Material** *Software (Solver)*: https://github.com/astra-uu-se/gecode-lns
*Software (Results, Experiments, and Scheme Generator)*: https://github.com/astra-uu-se/gecode-lns-experiments

**Funding** Supported by grant 2018-04813 of the Swedish Research Council (VR).

**Acknowledgements** We thank Mikael Zayenz Lagerkvist and Dexter Leander for their help with the Gecode implementation.

## 1 Introduction

Large neighbourhood search (LNS) [12, 9] is a method that combines systematic search with local search to improve the scalability of the former on constrained optimisation problems using heuristics of the latter. LNS starts from an incumbent solution that is iteratively improved by fixing a subset of the variables to their values in the incumbent solution and a value for each remaining variable is found and assigned via solving (such as constraint programming-style propagation and search). This method has been very successful on a wide variety of problems, such as vehicle routing [12, 1], bin packing [16], and scheduling [2, 13].

[1] Corresponding author

Traditionally, a modeller has to construct a problem-specific selection heuristic for the determination of the subset of variables to fix [2, 11]. However, there are now many variants of LNS that automatically determine that subset with good search performance, such as (but not limited to) (reverse) propagation guided LNS [8], cost impact guided LNS [4], explanation-based LNS [10], self-adaptive LNS [14], and variable-relationship guided LNS [13].

In most combinatorial optimisation solvers, such as for mixed integer linear programming, constraint programming (CP), Boolean satisfiability (SAT), and constraint-based local search (CBLS), when some variable $x$ becomes fixed, the values of some other variables can be found and assigned via search-free unique solving (via inference such as CP propagation, SAT unit propagation, and CBLS invariant propagation). Therefore, the relations between $x$ and the assigned variables are functional dependencies. In CBLS, these functional dependencies are fundamental and heavily studied [6, 7, 15, 3]. We have not found generic LNS selection heuristics or CP branching heuristics that automatically exploit these functional dependencies that are exploited in CBLS. However, they are often exploited manually by the modeller when for example creating a problem-specific CP branching heuristic that guides search.

Our contributions are:

- applying from CBLS to a CP context (and hence to CP-based LNS) the idea of a directed possibly cyclic graph that is induced by the functional dependencies between variables via the constraints of the model;
- designing a scheme that exploits the induced directed graph to remove variables from the LNS space that are functionally defined by others;
- describing how our scheme can be automated;
- showing that state-of-the-art generic LNS selection heuristics are easily extended to make use of our scheme;
- showing that our scheme often improves the overall performance when used with state-of-the-art generic selection heuristics; and
- showing that our scheme, when used with a naïve generic LNS selection heuristic that fixes a set of variables selected at random inside each LNS iteration, is competitive with more elaborate state-of-the-art generic LNS selection heuristics, even when they also use our scheme.

We first present how to apply the idea of a dependecy graph to a CP context (and hence to CP-based LNS) in Section 2. We give the background to our scheme in Section 3. Finally, we conclude in Section 4.

## 2 The Dependency Graph of a CP Model

In a CP model, some constraints are dependency constraints, where a *dependency constraint* $c$ on the variables $\mathcal{X} = \mathcal{I} \cup \mathcal{O}$ determines the values of the variables in $\mathcal{O}$ when the variables in $\mathcal{I}$ become fixed. We can view $c$ as a function that determines the values of output variables $\mathcal{O}$ when the input variables $\mathcal{I}$ become fixed. We say that $c$ is a *dependency constraint* and that $\mathcal{I}$ *functionally defines* $\mathcal{O}$ via $c$, denoted by $\mathcal{I} \overset{c}{\Longrightarrow} \mathcal{O}$, or simply that $\mathcal{I}$ *functionally defines* $\mathcal{O}$.

For example, consider the integer variables $y$ and $i$, the array $\mathcal{P}$ of integers, and the constraint $\textsc{Element}(\mathcal{P}, i, y)$, which constrains $y$ to be equal to the integer in $\mathcal{P}$ at index $i$. When $i$ becomes fixed to some value $v$, the value of $y$ becomes assigned to the integer in $\mathcal{P}$ at index $v$. The constraint is a dependency constraint, via which $i$ functionally defines $y$

For a CP model, the dependency constraints and the variables induce a directed graph, called the *dependency graph* [5], where for each set $\mathcal{I}$ of variables that functionally defines a set $\mathcal{O}$ of variables via some dependency constraint $c$, there is a vertex $d$, an arc $x \to d$ for each vertex $x \in \mathcal{I}$, and an arc $d \to y$ for each vertex $y \in \mathcal{O}$. Note that the dependency graph is possibly cyclic. Also note that all the variables and all the dependency constraints are in the dependency graph, while the other (non-dependency) constraints are not.

For any acyclic dependency graph, the source variables transitively functionally define all remaining variables. Therefore, for any acyclic dependency graph, the minimum-cardinality set of such variables is the set of source variables. However, this does not always hold for a cyclic dependency graph. In Section 3, we present a greedy scheme that, given a CP model (inducing a cyclic or acyclic dependency graph), finds a low-cardinality set of such variables, which can be exploited to guide LNS.

## 3 Dependency Curation for LNS

Typically, in generic selection heuristics, the freeze set is gradually updated inside each LNS iteration to include variables such that many other variables are assigned values via CP-style propagation [8, 13]. For these selection heuristics, such variables are found either experimentally or heuristically *during* search.

We call any set of variables that transitively functionally define all remaining variables a set of *search variables*, as only their values must be found via search.

Consider a CP model with the set $\mathcal{V}$ of variables. Finding a set $\mathcal{S} \subseteq \mathcal{V}$ of search variables is trivial as $\mathcal{V}$ itself is a set of search variables, though of maximum cardinality. Our idea is that the smaller the set $\mathcal{S}$ is, the more CP-style propagation will occur, as the value of each variable in $\mathcal{V} \setminus \mathcal{S}$ is found and assigned via only propagation (as $\mathcal{S}$ transitively functionally defines $\mathcal{V} \setminus \mathcal{S}$). Additionally, for any (generic or problem-specific) selection heuristic, if the freeze set is forced to become a subset of $\mathcal{S}$, then the LNS space is reduced, no data has to be stored, and no operations have to be performed on any variable in $\mathcal{V} \setminus \mathcal{S}$ by the selection heuristic, potentially improving its memory footprint and running time.

Note that if the dependency graph is acyclic, then the minimum-cardinality set of search variables is the set of source variables. Otherwise, the dependency graph contains at least one strongly connected component (SCC) with two or more vertices, and finding a low-cardinality set of search variables is a subtle issue, as discussed next.

As the variables and dependency constraints are known up-front, a low-cardinality set of search variables can be constructed before search starts. A (generic or problem-specific) selection heuristic can use the constructed set of search variables throughout search by forcing the freeze set to be a subset of that set.

Our scheme, called the *dependency curation scheme* (DCS), finds a low-cardinality set of search variables given the variables, vertices, and arcs of a dependency graph.

## 4 Conclusion and Future Work

We have presented our dependency curation scheme (DCS), which can be used with any (generic or problem-specific) LNS selection heuristic. We have compared the performance of a naïve generic randomised selection heuristic and more elaborate state-of-the-art generic selection heuristics from the literature both with and without DCS, revealing overall improved performance when using DCS. Our experiments show that the performance of using DCS with the naïve randomised selection heuristic is competitive with the more elaborate state-of-the-art generic selection heuristics, even when they use DCS.

## References

1 Russell Bent and Pascal Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers and Operations Research*, 33(1):875–893, January 2006. doi:10.1016/j.cor.2004.08.001.

2 Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *ICAPS 2005*, pages 81–89. AAAI Press, 2005.

3 Frej Knutar Lewander, Pierre Flener, and Justin Pearson. Invariant graph propagation in constraint-based local search. *Journal of Artificial Intelligence Research*, 2025. Forthcoming.

4 Michele Lombardi and Pierre Schaus. Cost impact guided LNS. In Helmut Simonis, editor, *CP-AI-OR 2014*, volume 8451 of *LNCS*, pages 293–300. Springer, 2014. doi:10.1007/978-3-319-07046-9_21.

5 Toni Mancini and Marco Cadoli. Exploiting functional dependencies in declarative problem specifications. *Artificial Intelligence*, 171(16–17):985–1010, November 2007. doi:10.1016/j.artint.2007.04.017.

6 Laurent Michel and Pascal Van Hentenryck. Localizer: A modeling language for local search. In Gert Smolka, editor, *CP 1997*, volume 1330 of *LNCS*, pages 237–251. Springer, 1997. doi:10.1007/BFb0017443.

7 Laurent Michel and Pascal Van Hentenryck. Localizer. *Constraints*, 5(1–2):43–84, 2000. doi:10.1023/A:1009818401322.

8 Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In Mark Wallace, editor, *CP 2004*, volume 3258 of *LNCS*, pages 468–481. Springer, 2004. doi:10.1007/978-3-540-30201-8_35.

9 David Pisinger and Stefan Ropke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 272 of *ORMS*, chapter 4, pages 99–127. Springer, 2019. doi:10.1007/978-3-319-91086-4_4.

10 Charles Prud'homme, Xavier Lorca, and Narendra Jussien. Explanation-based large neighborhood search. *Constraints*, 19(4):339–379, October 2014. doi:10.1007/s10601-014-9166-6.

11 Pierre Schaus, Pascal Van Hentenryck, Jean-Noël Monette, Carleton Coffrin, Laurent Michel, and Yves Deville. Solving steel mill slab problems with constraint-based techniques: CP, LNS, and CBLS. *Constraints*, 16(2):125–147, April 2011. doi:10.1007/s10601-010-9100-5.

12 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-François Puget, editors, *CP 1998*, volume 1520 of *LNCS*, pages 417–431. Springer, 1998. doi:10.1007/3-540-49481-2_30.

13 Filipe Souza, Diarmuid Grimes, and Barry O'Sullivan. An investigation of generic approaches to large neighbourhood search. In Paul Shaw, editor, *CP 2024*, volume 307 of *LIPIcs*, pages 39:1–39:10. Dagstuhl Publishing, 2024. doi:10.4230/LIPIcs.CP.2024.39.

14 Charles Thomas and Pierre Schaus. Revisiting the self-adaptive large neighborhood search. In Willem-Jan van Hoeve, editor, *CP-AI-OR 2018*, volume 10848 of *LNCS*, pages 557–566. Springer, 2018. doi:10.1007/978-3-319-93031-2_40.

15 Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search.* The MIT Press, 2005.

16 Gerhard Wäscher and Thomas Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spektrum*, 18:131–144, 1996. doi:10.1007/BF01539705.