Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers

Emir Demirović(✉), Geoffrey Chu, and Peter J. Stuckey

School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
{emir.demirovic,pstuckey}@unimelb.edu.au

Abstract. Large neighbourhood search, a meta-heuristic, has proven to be successful on a wide range of optimisation problems. The algorithm repeatedly generates and searches through a neighbourhood around the current best solution. Thus, it finds increasingly better solutions by solving a series of simplified problems, all of which are related to the current best solution. In this paper, we show that significant benefits can be obtained by simulating local-search behaviour in constraint programming by using phase saving based on the best solution found so far during the search, activity-based search (VSIDS), and nogood learning. The approach is highly effective despite its simplicity, improving the highest scoring solver, Chuffed, in the free category of the MiniZinc Challenge 2017, and can be easily integrated into modern constraint programming solvers. We validated the results on a wide range of benchmarks from the competition library, comparing against seventeen state-of-the-art solvers.

1 Introduction

Large neighbourhood search (LNS) [14] is a widely used metaheuristic for constrained optimisation. A neighbourhood of a given solution is the set of solutions that can be obtained by performing perturbations on a target solution. The size of the neighbourhood is determined by the used perturbations. Conventional local search considers small neighbourhoods due to efficiency. However, since the scope of the search is narrow, such methods are prone to be trapped in local optima. In contrast, large neighbourhood search uses significantly larger neighbourhoods. Thus, the optimisation algorithm has more options to escape local optima, while retaining the advantages of local search. Different techniques may be used to explore the neighbourhoods, including systematic search methods such as constraint programming or problem-specific heuristics. The use of constraint programming for neighbourhood exploration is particularly suitable for highly constrained optimisation problems where propagation and systematic search are advantageous compared to heuristic algorithms.

J. Hooker (Ed.): CP 2018, LNCS 11008, pp. 99–108, 2018.
https://doi.org/10.1007/978-3-319-98334-9_7

Phase saving is an approach, originally from SAT solvers [11], where the last value assigned for a variable in the search is given priority the next time the variables is branched on. The advantage is that phase saving interacts well with restarting since it was presumably nontrivial to find the value before the restart.

Solution-based phase saving is different, and not so commonly applied (see e.g. [1]). It gives priority to the value the variable is assigned to in the last *solution* found. For satisfaction problems, this is meaningless, as the search terminates once a solution is found. In optimisation problems, however, this concentrates the search around the current best solution, just as in LNS.

We use solution-based phase saving with activity-based search and nogood learning as a means of focusing the search around the best solution, mimicking typical local search behaviour. An advantage of this approach is that standard CP machinery can be used; hence it can be easily incorporated in most CP solvers. The method bears similarities with large neighbourhood search, as activity-based search and restarts can be seen as defining a "neighbourhood".

Our experimental results on a wide range of benchmarks from the MiniZinc Challenge 2017 demonstrate that by using the described approach, we can achieve significant improvements over Chuffed [4]. Furthermore, our approach can be easily integrated into modern constraint programming solvers, and it does not introduce additional parameters. To summarise:

- We introduce solution-based phase saving in constraint programming as a means to capture some of the benefits of LNS in a complete solving approach.
- We combine solution-based phase saving, activity-based search, Luby restarts, and nogood learning to obtain a complete CP algorithm that focuses its search around the best solution found so far. Our experiments show that activity-based search, perhaps unsurprisingly, is better suited for this task than random variable selection.
- We evaluate the proposed approach using benchmarks from the MiniZinc Challenge 2017 competition and compare with state-of-the-art solvers used in the competition. Overall, the results demonstrate that the approach is highly effective, improving the results for the highest scoring solver Chuffed.
- We discuss the relationship of our approach, as a contribution to search and black-box solvers, to large neighbourhood search.

2 Preliminaries

Constraint Programming. A *constraint satisfaction problem* (CSP) is a tuple $P \equiv (V, D, C)$, where V is a set of variables, D is a mapping from variable $v \in V$ to a set of values $D(v)$, and C is a set of constraints. An assignment θ assigns each $v \in V$ to an element $\theta(v) \in D(v)$. A *solution* is an assignment that satisfies all constraints in C. A *constraint optimization problem* (COP) is CSP augmented by an objective function f that maps each assignment to a value. The aim is to compute the *optimal solution* $\theta*$ such that $\forall\theta' : f(\theta*) \leq f(\theta')$.

Nogood Learning. Upon reaching a conflict, nogood learning solvers analyse conflicts to determine the assignments responsible for its cause. The reason for

failure is recorded in the form of a clause and added to the database. The mechanism allows *nonchronological backtracking*, where backjumps can take place several decision levels above the current level.

Restarts. To avoid searching extensively around local optima, solvers perform restarts after reaching a certain number of conflicts. The key is to strike a balance between diversification (frequent restarts) and intensification (infrequent restarts). Luby restarts [7] are widely used in SAT/CP solvers and aim to introduce a variety of restart frequencies, with smaller restarts being significantly more common, i.e. a partial Luby sequence is as follows: 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8...

Dynamic Search. The search algorithm repeatedly decides on the branching variables. A common approach is to select variable based on their recent activity in conflicts (VSIDS scheme [8]). After a conflict is detected, the *activity* of involved variables is increased and periodically a decay on all activities is applied. Thus, variables that we the cause of recent conflicts have priority. Other methods include *first fail* [5], *dom/wdeg* [2], and *impact-based search* [13].

Phase Saving [11]. As explained previously, when branching solvers must decide on a value assignment. A wide-spread approach for value selection, originally used in SAT solvers, is to choose the value used most recently for the variable. Therefore, after backtracking, the solver aims to return to its previous state as closely as possible. This behaviour is particularly well suited for restarts, as the solver can continue using the previous assignments instead of searching again. In addition, the information learned about its previous region of the search space through clause learning will still be relevant.

Large Neighbourhood Search [14]. The assignments of a subset of variables is fixed with respect to a given solution and a search algorithm, either local search or an exhaustive technique, is used to determine the best assignment for the remaining variables. The number of fixed variables is typically chosen adaptively: initially the algorithm selects a large number of variables to intensify the search and gradually diversifies by decreases the number of selected variables over the course of the algorithm.

3 Experiments

Solvers. We implemented solution-based phase saving for the value-selection strategy in the CP solver Chuffed [4]. In the experimentation, we compare with seventeen state-of-the-art solvers and their variants which were submitted to the MiniZinc Challenge 2017. For the sake of brevity we do not reference each solver individually but refer the interested reader to the competition.

Benchmarks. We used the 20 benchmarks with five instances of the competition. They encompass a wide range of problems.

Hardware, Time Limits, and Experimental Setup. We run the experiments using the same hardware and setting as in the *free search* category from

Table 1. Comparison of our approaches with Chuffed as the baseline solver.

Solver	score	iscore	area
chuffed-free	94.25	98.58	25887171
chuffed-random-free	76.63	77.17	40247361
chuffed-sbps-free	**126.12**	**121.25**	**23332231**

the MiniZinc Challenge 2017, where no restrictions are imposed on the solvers regarding search strategies. By doing so, we can position the approach directly to all solvers that participated in the competition.

Evaluation Metrics. We used the evaluation metrics of the Challenge. In short, score treats each benchmark instance as a vote between two solvers. It awards one point to the solver that found the better result and zero points for the other. If they perform equally well, the point is split in inverse proportion to their run times. The score for a solver is defined as the sum over all benchmark instances and considered solvers. The variation for incomplete solvers, iscore, ignores proof of optimality when comparing the performance of solvers.

The area score gives a measure of anytime performance of the solver. It computes the area under the curve defined by the function: 5000 if no solution found, $2500 \times (s - best)(worst - best) + 1250$ for a solution of value s where $worst$ and $best$ are the worst and best solutions found by any solver, and zero for proving optimality. This function effectively combines 25% of points for finding a solution, 50% for finding good solutions and 25% for proving optimality. It represents the area under the score curve over the twenty minutes run time, averaged across all benchmarks.

3.1 Comparison

A detailed breakdown of the results, in the style of the MiniZinc challenge results (minizinc.org/challenge2017/results2017.html), is available online: emirdemirovic.com/misc/cp18-sbps-comparison/.

Comparison with Chuffed: The first experiment compares variants of chuffed: chuffed-free is the competition version with Luby restarts and alternating free (activity-based) and fixed search; chuffed-sbps-free simply add solution-based phase saving to this; chuffed-random-free adds solution-based phase saving while using random variable selection in the free search; thus effectively mimicking random neighbourhoods (see Sect. 4).

We show the results in Table 1. Clearly, the use of solution-based phase saving significantly improves the performance over the baseline. Random neighbourhoods are not beneficial, as the advantages of solution-based phase saving are defeated by the random variable selection.

Looking more closely at the individual benchmarks: chuffed-sbps-free improves on all 20 benchmarks except `opt-cryptanalysis` where they are identical, and `cargo`, `crosswords`, `hrc` and `rc-graph-coloring`. The reason why

it underperforms is that the baseline approach can prove optimality of one or more instances where solution-based phase saving cannot. Clearly, solution-based phase saving is not as effective in proving optimality.

Interestingly, random neighbourhoods are preferable to activity-based ones on benchmarks `mario`, `opd`, and `rcpsp-wet`, showing its effectiveness for these particular situations. On other benchmarks the performance can be poor.

Comparison with All Solvers. chuffed-free was the highest points scorer in the free category of the Challenge. Comparing our variants, we find chuffed-sbps-free is clearly better than all other solvers and it reduces the area under the curve with respect to the baseline by 10%. Surprisingly, the use of random neighbourhoods is still ahead of all solvers except three, showing that solution-based phase saving is still powerful, even with a poor neighbourhood strategy.

Comparison with Local Search Solvers. Since LNS and local search perform well on the same problems, we compare the use of solution-based-phase-saving against the local search solvers in the competition, in particular on the problems where local search provided good results.

The results shown in Table 2 restrict the comparison to eight benchmarks where a local search solver ranked in the top three. Solution-based phase saving provides a substantial difference, pushing chuffed from below the performance of all local search solvers, to better than all of them. Interestingly, the low area suggests that chuffed-free finds good solutions early, but then gets stuck, where the local search solvers continue to improve. Solution-based phase saving is much better at continuing to improve solutions.

If we restrict our attention to the two problems where a local search solver was the best solver (oscar-free in both cases), we see that for these benchmarks even random neighbourhoods can improve on the baseline performance. Solution based phase-saving is not able to compete with the best local search in this case, but certainly markedly enhances the performance over the baseline (Table 3).

Table 2. Comparison of our approaches and local search solvers on benchmarks where local search solvers scored in the top three ranks.

Solver	score	iscore	area
chuffed-free	69.53	71.00	11043751
chuffed-random-free	49.42	51.00	17323701
chuffed-sbps-free	**99.70**	**93.00**	**8198639**
izplus-par	81.67	84.50	11368249
oscar-free	74.83	74.50	14734476
yuck-free	71.85	73.00	13638341

Table 3. Comparison of our approaches and local search solvers on the two benchmarks sets, `road-cons` and `opd`, where local search was the most effective method.

Solver	score	iscore	area
chuffed-free	19.88	23.00	3684082
chuffed-random-free	22.81	24.50	2831003
chuffed-sbps-free	26.45	27.00	2710511
izplus-par	8.80	11.50	6040284
oscar-free	**45.40**	**40.00**	**795009**
yuck-free	23.67	21.00	5853594

4 Relationship with Large Neighbourhood Search

We now contrast a modern CP solver using restarts, dynamic variable selection, and solution-based phase saving versus large neighbourhood search. We shall see that there are striking similarities between the two.

4.1 Restarts Versus Neighbourhood Size

Most uses of CP incorporate restarts to avoid being trapped in large useless parts of the search space. Restarts are managed by limiting some resource, such as time or number of conflicts. Once the limit is reached, the search is restarted. Limits usually increase over time to maintain completeness, e.g. using either geometric [15] or Luby [7] sequences. Restarting requires either nogoods [9] or randomisation in search to avoid repeating previous work.

LNS usually defines neighbourhoods by fixing a set of variables to their value in the best solution. The search of the neighbourhood continues until it finds a better solution or it thoroughly explores the neighbourhood. The size of the neighbourhood gives an implicit limit on the computation of this subsolve. In addition, LNS often explicitly restricts resource usage for the subsolve, to avoid cases where the neighbourhood defined is too large to explore exhaustively. Similarly, as for restarts, the limits are typically increased with time and randomisation in used to avoid searching through the same neighbourhood.

Therefore, both restarts and LNS tackle a series of subproblems while using randomisation and imposing limits on the computation for each subproblem.

Luby Restarts for Neighbourhood Size. Using small neighbourhoods can provide quick improvements but cannot escape local optima effectively, while large neighbourhoods provide the reverse effect. Thus, a balance between the two is often sought for, and many LNS algorithms adopt adaptive strategies to determine the size of the neighbourhoods. In CP solvers, the Luby [7] sequence can be used to determine the restart limits, and it achieves the desired behaviour: a balance between frequent and extended restarts. It simulates the adaptive strategies often seen in LNS.

4.2 Dynamic Search and Phase Saving Versus Neighbourhoods

In LNS, a significant subset of variables is selected, and the variables are assigned values according to an incumbent solution while leaving the remaining variables to the search strategy to explore. In CP solvers with dynamic variable ordering and solution-based phase saving, the search selects variables according to the variable-selection strategy and sets them to their value in the current best solution. Hence, the search will not fail until it fixes most variables since only the requirement for finding a better solution can invalidate the current best solution.

Thus, a similarity can be seen between the two approaches. The CP approach will fix almost all variables to their current best solution value, and afterwards, explore around this selection. Given the computation limits, it will backtrack only a subset of these decisions. Hence, it *implicitly* defines a neighbourhood given by the set of variables that are never reached during backtracking.

The first effective difference is that LNS may exhaust the neighbourhood before reaching its search limits, after which it terminates the search. In contrast, the CP approach will in effect *expand the neighbourhood* it explores, until hitting the restart limit. The second difference is that solution-based phase saving will always set a variable to its value in the best solution if possible, whereas in LNS this is not necessarily the case.

VSIDS as an Implicit Neighbourhood Selection Strategy. In LNS, it is crucial to select strongly related variables to avoid defining highly restrictive neighbourhoods with few solutions. In CP, activity-based search (VSIDS) tends to select connected variables as well, and when coupled with solution-based phase saving, we claim it implicitly builds "neighbourhoods".

We make the following observation to show the type of neighbourhoods generated by VSIDS. When conflicts occur during the search, activities of involved variables will be increased. As VSIDS prioritises variables with high activities for branching, this will create a positive feedback loop as more conflicts among related variables will be generated, hence further increasing their activity. Moreover, the exponential decay rate in VSIDS ensures that a variable's activity is largely determined by its most recent involvement in failure. Therefore, variables with similar activity values have been active at similar times. Conversely, related variables are likely to be active at the same time. Thus, as VSIDS branches on variables based on their activity, when coupled with solution-based phase saving, the resulting neighbourhood will consist of strongly connected variables with their values assigned as in the best solution found so far. This can be seen as *emergent LNS* behavior.

We note that solution-based phase saving with VSIDS has built-in diversification. Upon restart, variables that were previously selected first will have low activity values since they are unlikely to take part in many conflicts directly. Hence, they will not be selected early again, while the most active variables from the previous restart will now be placed at the top of the search tree. Thus, the variable-selection strategy will cycle through variables.

4.3 Further Differences

Large neighbourhood search has many variations, and not all of these are easily captured by solution-based phase saving, dynamic variable ordering, and restarts. We discuss this in the following text.

Local Objectives. While using the global objective within a neighbourhood is often appropriate, in some cases each LNS subsolve uses a different objective. This variation is particularly important when the global objective is likely to be fixed by variables outside the neighbourhood. The CP approach uses a single global objective and note that it cannot be "fixed" by the variable-selection procedure since that will cause failure, but this effectively means that some variables high in the dynamic ordering will be fixed to a different value than in the best solution.

Adaptive LNS. Often a set of different neighbourhoods are defined in LNS, which are used adaptively, biasing choices to those that lead to improvements during the search. However, while it would be possible to generate a dynamic variable ordering strategy that acts similarly, it is not standard.

Acceptance Heuristics. Variants of LNS will accept equally good solutions, so-called *side moves*, or slightly worse solutions (e.g. using a simulated annealing approach) to give more diversification to the search. Merely changing the variable ordering cannot achieve this.

5 Related Work and Conclusion

Solution-based phase saving can be used to mimic a local search strategy in CP solvers. While it is merely a value-selection heuristic, when combined with activity-based search and restarts, it is very similar to LNS. There are a number of other approaches to automatic neighbourhood generation.

In [10] neighbourhoods are created based on the propagation between variables. Neighbourhoods are built in two ways: by reduction or expansion. The first *reduction* approach iteratively selects a variable from a list of fixed size if possible and random otherwise. It is assigned its value in the best solution and variables whose domain was reduced by propagation of the assignment are added to the list. The next variable is chosen as that in the list with the most significant domain reduction. The process continues until the remaining problem is deemed small enough. The *expansion* approach works in the reverse direction.

The approach of [6] selects neighbourhood variables randomly with a bias towards those with high *impact* on the objective function. The rationale is that these variables are responsible for the current value of the solution. By changing their assignments, it can presumably obtain better solutions. The effectiveness of the approach is further improved by considering a combination of impact and *proximity*, where *proximity* follows a similar idea as closeness in [10].

The intuition is that impactful variables should be accompanied by related variables as otherwise the neighbourhood might be too restrictive.

In [12] neighbourhoods based on explanations arising from conflicts are investigated. The reasoning is that variables involved in conflicts are related, and hence form a suitable neighbourhood. This method is similar to VSIDS-based neighbourhoods, but here explanations are restricted to decisions, and they choose neighbourhoods based on the variables that lead to most conflicts, which is in some sense the opposite of the VSIDS approach.

A methodology for devising large neighbourhood search algorithms was presented in [3]. Unlike the previously discussed methods, it is not fully automated but instead offers guidelines for the design of large neighbourhood search algorithms. The authors suggest the following three principles: neighbourhood design should focus around the part of the problem that contributes the cost to the objective, several different adaptive neighbourhoods should be considered to ensure completeness of the approach, and learning techniques should be employed to determine the most effective combination of neighbourhoods and their resource limitations. The approach can be applied to a wide range of problems, and the authors demonstrate its effectiveness on job-shop scheduling.

Solution-based phase saving is a straightforward addition to a CP solver. It offers a substantial improvement on a wide range of benchmarks, significantly improving the best performing solver in the free category of the MiniZinc Challenge 2017, Chuffed. We expect other solvers to adopt solution-based phase saving as a powerful yet simple value selection strategy.

Acknowledgements. We would like to thank Andreas Schutt for his exceptional assistance with comparing solvers and Graeme Gange for his insight on the implementation.

References

1. Roig, I.A.: Solving hard industrial combinatorial problems with SAT. Ph.D. thesis, Technical University of Catalonia (UPC) (2013)
2. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI 2004, pp. 146–150 (2004)
3. Carchrae, T., Beck, J.C.: Principles for the design of large neighborhood search. J. Math. Model. Algorithms **8**(3), 245–270 (2009)
4. Chu, G.: Improving combinatorial optimization. Ph.D. thesis, The University of Melbourne (2011)
5. Haralick, R., Elliott, G.: Increasing tree search efficiency for constraint satisfaction problems. Artif. Intell. **14**, 263–313 (1980)
6. Lombardi, M., Schaus, P.: Cost impact guided LNS. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 293–300. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07046-9_21
7. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Inf. Proc. Let. **47**(4), 173–180 (1993)
8. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of DAC 2001, pp. 530–535 (2001)

9. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
10. Perron, L., Shaw, P., Furnon, V.: Propagation guided large neighborhood search. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 468–481. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_35
11. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 294–299. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72788-0_28
12. Prud'homme, C., Lorca, X., Jussien, N.: Explanation-based large neighborhood search. Constraints **19**(4), 339–379 (2014)
13. Refalo, P.: Impact-based search strategies for constraint programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_41
14. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49481-2_30
15. Walsh, T.: Search in a small world. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 1999, pp. 1172–1177 (1999)