Solution-Based Phase Saving and Large Neighbourhood Search

No Author Given

No Institute Given

Abstract. Large neighbourhood search, a type of 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, increasingly better solutions are found by solving a series of simplified problems, all of which are related to a the current best solution. In this paper, we argue that many of the benefits of large neighbourhood search can be achieved using a combination of three techniques: restarts, dynamic search, and solution-based phase saving. We show that using phase saving based on the best solution found so far during the search, when coupled with activity-based search (VSIDS) and nogood learning, provides a suitable means for effectively generating "neighbourhoods". 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 in modern constraint programming solvers. The results are validated on a wide range of benchmarks from the competition library and a comparison is performed 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. In conventional local search, small neighbourhoods are considered due to efficiency. However, since the scope of the search is narrow, such methods are prone to be trapped in local optima. In contrast, in large neighbourhood search, significantly larger neighbourhoods are considered. Thus, the optimisation algorithm is given more options to escape local optima, while retaining the advantages of local search. Different techniques may be used to explore the neighborhoods, 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.

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 decided on. The advantage of phase-saving is that it interacts well with restarting, since it was presumably nontrivial to find the value for the variable before the restart.

Solution-based phase saving is different, and not so commonly applied (see e.g. [1]). The priority is given 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 optimization 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 to obtain effective neighbourhoods. An advantage of this approach is that the size of the neighbourhoods is implicitly defined through the restart mechanism. Our experimental results on a wide range of benchmarks from the MiniZinc Challenge 2017 demonstrate that significant improvements can be achieved over Chuffed [4], the baseline solver. Furthermore, our approach can be easily integrated in modern constraint programming solvers and it does not introduce additional parameters. To summarise, our contributions are as follows:

- We introduce solution-based phase saving in constraint programming as a means to capture the essence of LNS in a complete solver.
- We combine solution-based phase saving, activity-based search, luby restarts, and nogood learning to obtain a generic large neighbourhood search style algorithm. Our experiments show that activity-based search, perhaps unsurprisingly, provides better neighbourhoods then randomly generated ones.
- 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.

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 a certain number of conflicts have been reached. 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 which variables to branch on. 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 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 select 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 Solution-based Phase Saving versus LNS

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 significant similarities between the two.

3.1 Restarts versus Neighbourhood Size

Almost all uses of CP make use of restarts to avoid being trapped in large useless parts of the search space. Restarts are usually managed by limiting some resource: time, fails, backtracks. When the limit is reached, the search is restarted. Limits usually increase over time to maintain completeness using either geometric [15] or Luby [7] sequences. Restarting requires either nogoods [9] or randomization in search to avoid repeating previous work.

In LNS neighbourhoods are defined, usually by fixing a set of variables to their value in the best solution. Search of the neighbourhood continues until a better solution is found or the neighbourhood is completely explored. Clearly the size of the neighbourhood gives an implicit limit on the computation of this subsolve. In addition, LNS often makes use of explicit limits on some resource for the subsolve, to avoid cases where the neighbourhood defined is too large to completely search. Limits for LNS are usually fixed in size. LNS requires randomization to avoid exploring the same (non-improving) neighbourhood.

Therefore, both restarting and large neighbourhood search tackle a series of subproblems while imposing limits on the computation in that subproblem.

3.2 Dynamic Search and Phase Saving versus Fixing Variables

In LNS, a subset of variables are selected and fixed to their value in some incumbent solution. The remaining variables are left to the search strategy to explore. Using CP with dynamic variable ordering and phase saving, the search selects variables in some order, and sets them to their value in the current best solution. Hence, search will not fail until most variables are fixed, since only the requirement for finding a better solution can invalidate the current best solution.

Thus, a similiarity can be seen between the two approaches, as both impose limits on the subsolve/restart. The CP approach will fix almost all variables to their current best solution value, and afterwards explore around this selection. Given the computation limits, only a subset of these decisions will be backtracked. 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, and will terminate 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 the solution-based phase saving approach will always set a variable to its value in the best solution if possible, whereas in LNS the value selection approach is given by the search strategy.

3.3 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 good, in some cases each LNS subsolve uses a different objective. This 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, although note that it cannot be "fixed" by the neighbourhood, 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 LNS a set of different neighbourhoods are defined, 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.

4 Emergent LNS

Once dynamic variables ordering, solution-based phase saving, and restarts are used, the CP search starts resembling a LNS search. We denote by *emergent*

LNS the resulting kind of search that arises, and show that widely used dynamic variable ordering and restart strategies ideed make the CP search closer to LNS.

4.1 VSIDS as a Neighbourhood Selection Strategy

It is important to select strongly related variables to avoid defining highly restrictive neighbourhoods with few solutions. We claim that activity-based search (VSIDS) used as a dynamic search strategy with solution-based phase saving builds implicit neighbourhoods that have this property.

To understand the neighbourhoods generated by VSIDS, we make the following observations. 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, the resulting neighbourhood will consists of strongly connected variables.

VSIDS as an implicit neighbourhood selection strategy also has has built-in diversification. Upon restart, variables that were previously selected first will have low activity values since they are unlikely to directly take part in many conflicts. Hence they will not selected early again, while the most active variables from the previous restart will now be placed at the top of the search tree. Hence, implicit neighbourhoods variables will cycle through variables.

4.2 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, restarts occur after a predefined resource limit is reached. By using the Luby [7] sequence to determine the restart limits, we can achieve the desired behavior: frequent restarts with occasional larger limits. This simulates the adaptive strategies often seen in LNS.

5 Experimental Results

Solvers. We implemented solution-based phase saving 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 instead refer the interested reader to the competition for more details.

Benchmarks. We used the 20 benchmarks with 5 instances of the competition. They encompass a wide range of problems.

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**

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

Hardware, Time Limits, and Experimental Setup. We run the experiments using the same hardware and setting as in the *free search* category from the MiniZinc Challenge 2017, where no restrictions are imposed on the solvers in terms of search strategies. By doing so, we are able to position the approach directly with respect 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. The solver finding the better result is awarded one point and zero for the other. If they perform equally well on result 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 performance of solvers.

The area score gives a measure of anytime performance of the solver. The area under the curve is computed from 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.

5.1 Comparison

We compare our approach in a number of experiments. A full breakdown of the results, in the style of the MiniZinc challenge results (`http://www.minizinc.org/challenge2017/results2017.html`), is available at `ommited~for~anonymization`.

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).

The results are shown in Table 1. Clearly the use of solution-based phase saving significantly improves the performance over the baseline. Random neighbourhoods are not effective, 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 is able to prove optimality of

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

one or more instances where the 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 effectivness 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 new variants, we find chuffed-sbps-free is clearly is better than all other solvers and it reduces the area under the curve wrt 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. Clearly, solution-based phase saving provides a substantial difference, pushing chuffed from below the perfomance 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 are able to 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 improves the performance over the baseline.

6 Related Work and Conclusion

Solution-based phase saving can be used to mimic a local search strategy in CP solvers. It effectively provides a form of automatic neighbourhood selection. 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

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

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 approach.

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 greatest 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, presumably better solutions can be obtained. To improve the effectiveness of the approach, a combination of impact and *proximity* of variables is taken into account, 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 explainations arising from conflicts are investigated. The reasoning is that variables involved in conflicts are related, and hence form a suitable neighbourhood. This is very 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 automatised but instead offers guidelines for the design of large neighbourhood search algorithms. The authors suggest the following three principles: neighbourhood design should be focused 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 its effectivenss is demonstrated on jobshop scheduling.

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

References

1. Ignasi Abío Roig. Solving hard industrial combinatorial problems with SAT. *Ph.D. thesis, Technical University of Catalonia (UPC)*, 2013.
2. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Procs. of ECAI04*, pages 146–150, 2004.
3. Tom Carchrae and J. Christopher Beck. Principles for the design of large neighborhood search. *Journal of Mathematical Modelling and Algorithms*, 8(3):245–270, 2009.
4. Geoffrey Chu. *Improving Combinatorial Optimization.* PhD thesis, The University of Melbourne, 2011.
5. R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
6. Michele Lombardi and Pierre Schaus. Cost impact guided lns. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 293–300, Cham, 2014. Springer International Publishing.
7. M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Inf. Proc. Let.*, 47(4):173 – 180, 1993.
8. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC'01*, pages 530–535.
9. O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.
10. Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 468–481, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
11. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.
12. Charles Prud'homme, Xavier Lorca, and Narendra Jussien. Explanation-based large neighborhood search. *Constraints*, 19(4):339–379, October 2014.
13. P. Refalo. Impact-based search strategies for constraint programming. In *Procs. of CP2004*, pages 557–571, 2004.
14. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the Principles and Practice of Constraint Programming (CP 1998)*, pages 417–431. Springer.
15. Toby Walsh. Search in a small world. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 1172–1177, 1999.

A Full comparison versus all solvers

The full table against all solvers is shown below for completeness, the new variants perform as already seen in Table 1.

Solver	score	iscore	area
choco4-free	1161.48	1148.11	44624585
choco5-free	961.34	1018.61	52150698
chuffed-free	1483.39	1401.75	26042248
chuffed-random-free	1259.87	1178.83	40250386
chuffed-sbps-free	**1642.59**	**1521.75**	**23228693**
concrete-free	711.24	730.94	70935065
g12fd-free	640.01	690.00	66529554
gecode-fd	1167.35	1181.50	41747717
haifacsp-free	894.10	911.53	54896319
izplus-free	1427.73	1505.86	34294731
jacop-fd	1122.05	1129.50	45282205
lcg-glucose-free	1256.33	1185.74	37753989
mistral-free	1195.39	1285.05	46352131
mzn-cbc-free	499.97	479.50	86869418
mzn-gurobi-free	1321.01	1190.67	45898681
or-tools-cp-free	970.92	963.00	52065779
or-tools-lcg-core-free	1154.91	1026.80	48499372
or-tools-lcg-free	1366.49	1219.55	37685346
oscar-free	566.33	587.50	83101752
picat-cp-fd	670.11	680.50	66347667
picat-sat-free	1238.55	1174.78	42902781
sicstus-fd	800.38	833.50	58766931
yuck-free	684.86	759.50	71566880

Table 4. Comparison of our approaches and other solvers, according to MiniZinc Challenge 2017 rules.