Commentary on "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems"

Paul Shaw

September 2019

Large Neighbourhood Search[1] was an idea that came to me in the spring of 1997 or thereabouts when I was working on a European project called "GreenTrip" with Patrick Prosser and Philip Kilby. The project was on vehicle routing problems and I had been looking at insertion heuristics. The idea is that you start with an empty routing plan (except for the depot from which trucks leave and to which they return), and you add in customer visits one at a time between two existing visits on a truck until you get to a full solution. According to which position you add each visit, you will get different solutions. Of course, at least one of the sets of possible positions will get you to an optimal solution (say, lowest total distance). I spent countless hours drawing circles and lines on paper (erasing lines when I did some insertion), trying to work out some good insertion strategies.

At the time, there were a lot of thoughts about decision repair in tree search (at least where I was working), probably brought about by the publication of limited discrepancy search (LDS) [7] a couple of years before. LDS was a really nice idea that allowed decisions *anywhere* in the tree to be repaired, and was also in the same spirit as iterative deepening—okay, you might redo a little bit of work, but who cares? LDS was for decision problems (finding a single solution). But I had an optimization problem and I wanted to apply a similar kind of thinking to it—I wanted to be able to repair some of my insertion decisions[2] that I had made when producing my solution. Just as for depth-first search, it turns out that you can easily adapt LDS to an optimization problem by following the tree traversal rule but not stopping when a solution is found. You essentially just keep the current best solution you find in the search tree, all the while pruning nodes that have a lower objective bound greater than the value of the current best solution. This produces a series of progressively better solutions. I tried this method using the LDS tree-traversal strategy with my best insertion heuristics, but the result was not great even on the moderately-sized problems I was looking at. But I kept that little backtracking code.

At some point I realized that using LDS in this way on optimization problems wasn't very clever. When LDS finds a solution to a decision problem, it has gained some information, namely it has found out what heuristic it *should have* been following instead of the left branch of the search tree. But, since the search is over, it can't do anything with that information. Now, with optimization problems, you keep going to try to find better solutions, so surely this information could be useful? Also, as better and better solutions are found, maybe the decision paths followed to get to those solutions indicate better and better heuristics? This kind of heuristic updating

[1]Thanks to Ian Gent for coming up with the name!

[2]Repairing an insertion decision means inserting the customer visit at a less preferred spot than the original spot chosen by the heuristic.

has actually been used by methods such as [8, 16]. I cannot recall why I did not follow exactly that route at the time, but I am glad I didn't because I think LNS is conceptually simpler.

I had a few different things in my head. One was that repairing decisions regardless of when they were taken when building a solution was a good thing (and probably even better was to not even consider they were taken in any kind of order in the first place—they were just given). Another was that heuristics probably work better when they don't have a lot of decisions to make (a corollary of the logic used in LDS). The last thing was that in GreenTrip, we were using a lot of local search, and so seeing things through a local search lens was more natural to me at the time than visualizing a search tree. Taking all these things together, LNS seemed to me to be the most natural thing in the world[3]:

```
Find a solution $\mathcal{S}$
repeat for as long as desired
    Undo some decisions in $\mathcal{S}$ creating $\mathcal{T}$
    Try to complete $\mathcal{T}$ to $\mathcal{U}$, an improvement
        upon $\mathcal{S}$, by taking more decisions
    If $\mathcal{U}$ could be found, replace $\mathcal{S}$ by $\mathcal{U}$
return $\mathcal{S}$
```

The key part about LNS is the fact that one typically uses some kind of tree-based search mechanism to complete $\mathcal{T}$ to $\mathcal{U}$, which makes it particularly easy to implement if you happen to have a CP (or MIP) solver already. The backtracking code that I wrote for visit insertion came in handy right there.

I remember feeling that I knew LNS would work before I had coded it (at least for routing problems). I had spent so long playing around with circles and lines on a notepad, I knew I could get good solutions even manually by rubbing out some lines somewhere on the pad and redrawing them as best I could. The ever-wise Ian Gent encouraged me to write a paper, which became a technical report, and then was submitted to CP-98 the next year.

[3] These kinds of ideas seemed really to have been in the air: [11] and [18] are two good examples. At the time, I was not aware of anyone else working on this kind of thing though.

I never did write a follow-up paper dedicated to LNS in a more general setting with a literature review and different target problems: it just seemed like such a dull thing to do. However, if you are looking for an introduction, there is [2, 14, 19], and for routing specifically, I like [13]. There is now even a Coursera course by Peter Stuckey and Jimmy Lee with a full section on LNS. Since CP-98, LNS has been used extensively in too many papers to cite: probably the best thing is to use an internet search, as new papers using LNS are being published every year.

For me, the best part is that LNS has become just as much a practical success as an academic one, especially as a complement to backtracking solvers. Basically, I think this is because it is easy to implement on top of a backtracking solver, and in many cases you can get large improvements to solving speed. I moved to ILOG in 1998 (just before the CP-98 conference actually) and some time after, LNS or LNS-inspired techniques were being used in numerous different ILOG products and projects [3, 4, 5, 6, 9, 12]. I always enjoy hearing about projects that have been given a big boost by using LNS—often the sentiment is one of surprise that something so simple could work so well.

Over the years, different methods of making LNS "smarter" have been proposed. To name a few: using reinforcement learning [9, 15, 20], using information from constraint propagation [12], using information from the cost function [1, 10], and also the use of explanations [17]. Since LNS is also just a hill climbing algorithm, any meta-heuristic techniques from the local search literature can be used with it too. This creates myriad possibilities for hybrid techniques. Even though LNS is over twenty years old, I still think there is life in the old dog yet, both on the fundamentals, and in applying LNS to real problems.

Last year, 2018, was a nostalgic year for me as it was 20 years after I published the LNS paper in 1998, and 10 years after what I consider the

beginning of CP Optimizer in 2008. Happily, I had the chance to talk about both of these at two different venues in June. In early June, I was invited to give a talk on combinations of constraint programming and local search at the University of Toulouse. A good deal of my talk was about LNS, with hints and tips, and some ideas for future work. You can find the slides I used and a video of the talk at
`http://www.cimi.univ-toulouse.fr/optimisation/en/masterclass` (You need to go down to close to the bottom of the page.) Near the end of June, I gave a talk on "Ten Years of CP Optimizer" at CP-AI-OR in Delft. You can find those slides on `slideshare.net` by searching for my name. LNS is a key component of CP Optimizer, essential to solving the real-world problems to which CP Optimizer is applied every day. I feel very lucky to have seen this pencil-and-paper idea be realized in this way.

References

[1] T. Carchrae and J. C. Beck. Cost-based large neighborhood search. In *Workshop on the Combination of Metaheuristic and Local Search with Constraint Programming Techniques*, 2005.

[2] Tom Carchrae and J. Christopher Beck. Principles for the design of large neighborhood search. *Journal of Mathematical Modelling and Algorithms*, 8:245–270, 2009.

[3] A. Chabrier, E. Danna, C. Le Pape, and L. Perron. Solving a network design problem. *Annals of Operations Research*, 130:217–239, 2004.

[4] E. Danna and L. Perron. Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In *Proceedings of CP 2003*, pages 817–821, 2003.

[5] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming Series A*, 102:71–90, 2005.

[6] D. Godard, P. Laborie, and W. Nuijten. Randomized large neighborhood search for cumulative scheduling. In *Proceedings of ICAPS*, pages 81–89, 2005.

[7] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th IJCAI*, pages 607–615. Morgan Kaufmann Publishers, 1995.

[8] N. Jussien and O. Lhomme. The path-repair algorithm. In *In CP-98 Workshop on Large Scale Combinatorial Optimization and Constraints*, pages 73–86, 1999.

[9] P. Laborie and D. Godard. Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Proceedings of MISTA-07*, pages 276–284, 2007.

[10] Michele Lombardi and Pierre Schaus. Cost impact guided lns. In *Proceedings of CP-AI-OR 2014*, pages 293–300, 2014.

[11] T. Mautor and P. Michelon. MIMAUSA: A new hybrid method combining exact solution and local search. In *Second International Conference on Meta-Heuristics*, 1997.

[12] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In *Proceedings of CP 2004*, pages 468–481, 2004.

[13] D. Pisinger and S. Røpke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34(8):2403–2435, 2007.

[14] D. Pisinger and S. Røpke. Large neighborhood search. In M. Gendreau, editor, *Handbook of Metaheuristics*, pages 399–420. Springer, 2010.

[15] S. Røpke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, 2006.

[16] S. Prestwich. Combining the scalability of local search with the pruning techniques of systematic search. *Annals of Operations Research*, 115:51–72, 2002.

[17] Charles Prud'homme, Xavier Lorca, and Narendra Jussien. Explanation-based large neighborhood search. *Constraints*, 19(4):339–379, jul 2014.

[18] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159:139–171, 2000.

[19] Paul Shaw. Constraint programming and local search hybrids. In Michela Milano and Pascal van Hentenryck, editors, *Hybrid Optimization: The Ten Years of CPAIOR*, pages 271–303. Springer, 2011.

[20] Charles Thomas and Pierre Schaus. Revisiting the self-adaptive large neighborhood search. In *Proceedings of CP-AI-OR 2018*, pages 557–566, 2018.