Constraint Directed Variable Neighbourhood Search

Alastair Andrew
University of Strathclyde,
16 Richmond Street,
Glasgow,
G1 1XH
UK
alastair.andrew@cis.strath.ac.uk

Abstract

Local search depends on making small perturbations to candidate solutions to arrive at new and potentially better candidates. There are different schemes for managing the perturbations and the selection of candidates, but in this work we focus on the neighbourhood defined by particular perturbations. Previous work has observed that the problem of local search becoming trapped in local optima is only experienced with respect to a particular neighbourhood. The Variable Neighbourhood Search strategy attempts to avoid these optima by linearly switching between neighbourhoods. We propose a new method where the selection of the neighbourhoods is dynamic and based upon the violations of the problem constraints, Constraint Directed Variable Neighbourhood Search. We compare this approach with Variable Neighbourhood Search and demonstrate that the same search progress is achieved whilst exploring only a fraction of the states.

Introduction

All problems in the real world are subject to some form of constraints on their possible solutions. If there were no constraints then potentially any solution would be valid. This is not the case though as actual scheduling, timetabling, and rostering problems are heavily constrained. The laws of physics are surprisingly inflexible and any attempt to schedule two processes to occur simultaneously on the same resource is destined to fail. Other constraints may not be impossible to violate but it may be undesirable; few employees would appreciate being rostered to work for 24 consecutive hours! It should be clear that to produce any useful applications for these domains then they must be able to work with the underlying constraints of the problem structure.

One approach which has successfully been applied to many of these problems is local search. Local search is conceptually a very simple algorithm. To find the optimal solution for a problem take an existing solution and try to make some small changes to it. If these changes improve the quality of the solution then accept that solution and try to optimise it further. Local search has been shown to give good quality solutions within acceptable time limits where other techniques such as Constraint Programming or Linear Programming struggle.

The weakness of local search is its propensity to become trapped at local optima. These are states where the search has explored all the surrounding solutions and found none better than the current state, however the current state is not the global optima. Good local search algorithms need the ability to escape from, or avoid becoming trapped at, these local optima. Various different techniques have been adopted, most research has gone into meta-heuristics such as Simulated Annealing and Tabu Search. These strategies alter the acceptance function of the algorithm so that it can accept solutions which are not necessarily better than the current state in the hope that extra exploration will ultimately lead to improved solutions.

Another approach put forward by (Mladenovic & Hansen 1997) is Variable Neighbourhood Search (VNS). They observed that when the search becomes trapped at a local optima, this is only because none of the states explored using its neighbourhood offer an improvement. By exploring a different neighbourhood then different surrounding states will be reachable and the search may avoid becoming trapped. The canonical VNS explores its alternative neighbourhoods in a linear fashion, the order of which is specified by the algorithm designer. Typically though the neighbourhoods are chosen in order of ascending size.

An interesting development by (Viana, de Sousa, & Matos 2005) provides a different approach, Constraint Oriented Neighbourhoods. Viana et al note that when searching often small changes can make the solution infeasible and place it into a state that requires several moves to return to feasibility. They attempt to avoid this by computing chains of potential neighbourhood moves which can return a solution quickly to feasibility. This notion of linking the selection of the neighbourhood to the violations of the problem constraints is expanded upon within our work.

The approach we propose is that by keeping track of the violations of problem constraints then appropriate neighbour-

hoods to explore can be chosen. Rather than linearly exploring all the possible neighbourhoods just a subset which can improve the current constraint violations are chosen. The aim is create a more efficient algorithm which requires less searching and achieves this through intelligent exploitation of a problem's structure. However we want this technique to retain enough generality that it can be applied any constrained problem types such as timetabling or scheduling.

The Problem

The problem chosen for experimentation was the Bin Packing Problem. It is one of the classic optimisation problems and is conceptually very simple. The problem instances were acquired from the Operations Research Library (Beasley 2005). A series of packages must be assigned to a set of bins which have a fixed capacity. The objective is to fit all the differently sized packages into as few bins as possible. The problem has only one constraint to start with, namely a bin cannot contain more packages than it has capacity for. To make the problem more interesting we introduced three new constraints which exemplified the kind of constraints found both in other constrained problems and also real world problems. For our experiments the aim was not to try and pack the bins, simply to take a starting solution created by a greedy heuristic and attempt to remove all the constraint violations from it.

Clashing Packages Constraint In many problems there exists the notion of assignments which are incompatible with each other. In course timetabling classes which share students can never feasibly occur during the same period. For our problem we divided the packages into two categories based on whether they were oddly or evenly numbered and then enforced that within a bin these types could not be mixed.

Package Ordering Constraint This constraint states that within a bin the larger packages must be assigned lower positions than smaller ones. In a warehousing problem it would not be unreasonable to assume that the smaller crates must be placed atop the larger ones.

Bin Ordering Constraint To satisfy this constraint a bin must be as full or more so than any of the following bins. This constraint means the ideal solution will have the bins arranged from left to right in descending order of size. The neighbourhoods most suited to solving this constraint are ones which move all the contents of a bin as a single group.

Neighbourhoods

For this experiment we created twelve different neighbourhoods: moveBin, movePosition, moveBinAndPosition, swapBin, swapPosition, swapBinAndPosition, moveAllContentsBin, moveAllContentsPosition, moveAllContentsBinAndPosition, swapAllContentsBin, swapAllContentsPosition, swapAllContentsBinAndPosition. The design of these neighbourhoods was influenced by the work done on composing timetabling neighbourhoods by (Di Gaspero & Schaerf 2007). The rationale behind this decision was that in the future the neighbourhoods for Constraint Directed Variable Neighbourhood Search (CDVNS) could be inferred from a formal definition of the problem, at present though they are implemented manually. A recent technical paper by (Ågren, Flener, & Pearson 2007) uses conventional CSP notation to fully capture a problem akin to Constraint Programming.

Constraint Direction

Key to the operation of CDVNS is the neighbourhood selection matrix. This structure stores which neighbourhoods should be searched when attempting to solve any given constraint configuration. It is important that the neighbourhoods returned are as specific as possible. By specific we mean able to reduce the current constraint violations whilst leaving the other constraints unaffected. An example of this would be for the Bin Ordering Constraint where the swapAllContentsBin neighbourhood is the most appropriate and specific neighbourhood. This is because it exchanges the entire contents of bins and retains their original internal orderings. It cannot affect the other problem constraints. No packages are added or removed from the collections of packages being swapped so the amount of existing package clashes will remain constant. This also means that no Capacity Exceeded constraints can be added or removed. Finally since the orderings are left unaltered the amount of Package Ordering Constraints violations is maintained.

Algorithm 1 Constraint Directed VNS

```
1:  while violations > 0 and iterations < limit do
2:      constraint ← getDominantConstraint()
3:      neighbourhoods[] ← matrix[constraint]
4:      for all n in neighbourhoods[] do
5:          currentScore ← explore(n)
6:          if currentScore < bestScore then
7:              bestScore ← currentScore
8:          end if
9:      end for
10:     violations ← bestScore.violations
11: end while
```

Not all constraints can be tackled with such specific neighbourhoods and this introduces the new problem of what order to attempt to solve the constraint violations in. Our current solution is to artificially weight the Clashing Packages violations so that the search focuses on solving them first. Solving the constraints using a naive unweighted approach where the most pressing constraint was chosen simply based

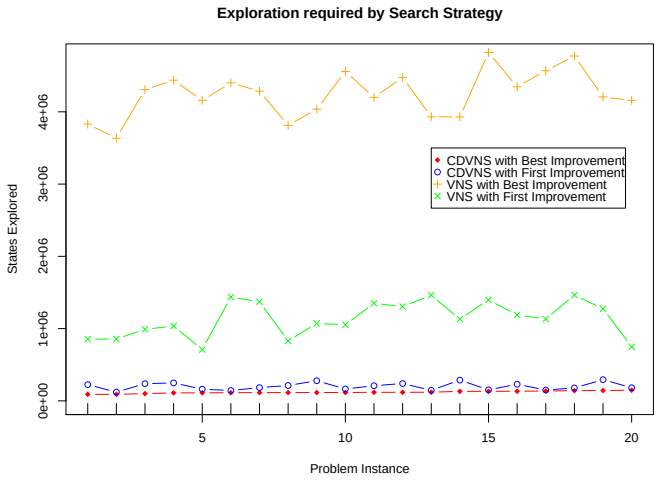

Figure 1: Results from the 120 Package Problem Instances

on the amount of violations led to the search having to perform extra unnecessary exploration. In our experimental runs the unweighted system would often require to explore more than twice the number of states used by the weighted run.

Initial Results

The initial results can be seen in Figure 1. In the figure there is a plot of the two different search strategies, CDVNS and an exhaustive VNS, each trying two different acceptance strategies, Best Improvement and First Improvement. The CDVNS with a Best Improvement uniformly outperforms the other strategies on all the problem instances. There are various interesting things to note. Firstly using the Best Improvement strategy with CDVNS always performs better than the First Improvement which seems counter-intuitive since with the Best Improvement strategy we are committing to fully explore each neighbourhood. The First Improvement scheme only needs to explore a neighbourhood until it finds an improving situation.

Another point to note is that the CDVNS scheme has a lot less variation in the amount of exploration required to solve each of the instances. For the plain exhaustive VNS there was far more variation, indeed some results differed by a larger amount than the CDVNS actually required to solve the problem.

Conclusions

By exploiting knowledge about the underlying problem constraints it is possible to reduce the amount of states explored by a VNS whilst not sacrificing search reach-ability. This reduction in search exploration is achieved by using the structure of the problem and should be applicable to any constrained problem. An added benefit is that the performance does not seem to fluctuate wildly across problem instances.

Future Work

There are many directions which we intend to explore with this work. Currently the precedence of constraints is manually specified by weighting the number of violations. It should be possible to assign this weighting automatically by creating a directed graph structure that captures the information about which neighbourhoods can affect the other constraints. Standard graph labelling techniques should then be able to produce an efficient ordering for tackling the constraints.

The current neighbourhoods were designed and implemented manually, by specifying potential problems in a more formal language akin to a CSP we intend to infer appropriate neighbourhoods and create the selection matrix automatically.

At present the algorithm is implemented in Python but in the future we aim to make use of the Comet language, (Van Hentenryck & Michel 2005). Comet provides many abstractions which ease the implementation of Local Search algorithms and should allow for more comprehensive experimentation to be performed. The language should also make it easier to combine the neighbourhood selection element of CDVNS with the acceptance strategies of other meta-heuristics.

References

Ågren, M.; Flener, P.; and Pearson, J. 2007. On constraint-oriented neighbours for local search. Technical Report 2007-009, Department of Information Technology, Uppsala University, Box 337, SE - 751 05 Uppsala, Sweden.

Beasley, J. E. 2005. Or-library http://people.brunel.ac.uk/ mastjjb/jeb/info.html.

Di Gaspero, L., and Schaerf, A. 2007. A composite-neighbourhood tabu search approach to the travelling tournament problem. *Journal of Heuristics* 13(2):189–207.

Mladenovic, N., and Hansen, P. 1997. Variable neighborhood search. *Computers and Operations Research* 24(11):1097–1000.

Van Hentenryck, P., and Michel, L. 2005. *Constraint-Based Local Search*. The MIT Press.

Viana, A.; de Sousa, J. P.; and Matos, M. A. 2005. Constraint oriented neighbourhoods - a new search strategy in metaheuristics. In Ibaraki, T.; Nonobe, K.; and Yagiura, M., eds., *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research / Computer Science Interfaces Series*. Springer. chapter 15.