Progress Control in Variable Neighbourhood Search

Tim Curtois[1], Laurens Fijn van Draat[2], Jan-Kees van Ommeren[3], and Gerhard Post[23]

[1] ASAP, School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, UK
[2] ORTEC bv, Groningenweg 6K, 2803 PV Gouda, The Netherlands, www.ortec.com
[3] Department of Applied Mathematics, University Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

1 Introduction

The methods of intensification and diversification are indispensable in successful meta heuristics for local search. Intensification corresponds in some sense to local optimisation; the neighbourhood of a solution is searched intensively for solutions which are better or have better opportunities. On the other hand, diversification tries to escape from (relatively small) neighbourhoods to solutions which might lead to better final results. A heuristic that is well aware of the intensification versus diversification problems, is the Variable Neighbourhood Search (VNS), see [2]. In this method, more than one neighbourhood structure is considered. After finishing intensification with respect to one neighbourhood, the heuristic diversifies to another neighbourhood. In this way one hopes to escape from poor local optima.

In this work we introduce a model to predict the quality of a neighbourhood. We use this model to identify 'bad' neighbourhoods and avoid searching them. We call this process 'Progress Control'. Computational results are presented to show that progress control helps us finding better solutions in the same amount of time.

2 Problem setting

2.1 Alternating Neighbourhood Search

Our object of research is a specific type of VNS, in which there is a 'small' neighbourhood, and a 'large' neighbourhood. After reaching a local optimum with respect to the small neighbourhood, we move our attention to another solution in the large neighbourhood. From here we again perform local optimisation with respect to the new small neighbourhood. If we reach a better local optimum, we take this new solution as the starting point for further investigation, otherwise we revert to the old local optimum, and continue from there. We call this method 'Alternating Neighbourhood Search' (ANS).

E. K. Burke, H. Rudová (Eds.): PATAT 2006, pp. 376–380. ISBN 80-210-3726-1.

In ANS it is normal that 99% of the calculation time is spent on local optimisation with respect to the small neighbourhood. A possibility to improve the performance of ANS is to detect, somehow, that it is useless to continue the current local optimisation. The most obvious way is to stop if, after some fixed amount of time, a certain quality has not been reached. In our experience, this method is not good enough: a bad solution that improves steadily might still be a promising one.

In this section we propose a model for the progress of the local optimisation process. With this model we follow the progress and try to predict which neighbourhoods are good and which are bad.

2.2 Modeling the progress

Suppose we follow the local optimisation and get an update of the result every τ milliseconds. With c_n we denote the cost after n periods. Furthermore we denote with $c = c_\infty$ the cost of the local optimum that eventually will be reached. Since we apply local optimisation, we know that $c_{n+1} \leq c_n$. The moves are selected randomly, hence it seems reasonable that the chance that a move at time t_n is successful is proportional to the difference $c_n - c$. Hence, for the expected value of the cost c_n we get

$$E(c_n - c) = \lambda(c_{n-1} - c).$$

Solving this recursion relation we obtain

$$E(c_n) = (c_0 - c)\lambda^n + c. \tag{1}$$

In continuous time we use $t_n = n \cdot \tau$ to get

$$c(t) = Ae^{Bt} + C, \tag{2}$$

which is more convenient to use. Here $A = c_0 - c$, $C = c$, and $e^{B\tau} = \lambda$.

Several assumptions underlie this model:

1. The number of good moves is proportional to the difference of the current score and the end score. In practice this is not correct: some violations of soft constraints can be removed by several moves and will probably found earlier. This implies that λ decreases with time (hence our early estimates are optimistic).
2. The cost change of a good move is always the same. In practice this is not correct: soft constraint violations with high costs are usually detected earlier, as a move can lift this high cost at introducing small costs for other soft constraints. Again early estimates are optimistic.
3. All moves take an equal calculation time. In practice this is not exactly the case, but if τ is relatively big, several hundreds of moves occur between t_n and t_{n+1}, and the number of these moves is with high probability close to average.

All these assumptions make that the predicted progress differs from the realised progress: the realised progress is much more irregular, especially in the beginning of the local optimisation. For this reason we use a rolling horizon for the progress controller, see section 2.3.

Although the model above is the most appealing model, we did not use it in our more extensive experiments. Preliminary experiments revealed that the approach in (2) is hard to scale: quite soon the exponential factor B is strongly negative, and C equals approximately the latest cost. To circumvent this problem we assume that the local optimisation will reach a perfect result of cost 0. Hence we try to approximate the progress with the curve

$$c(t) = Ae^{Bt}. \tag{3}$$

With this formula we can estimate t^*, the time needed before the local optimiser will reach the quality of the current best result. If this estimated time t^* is larger than a certain reference time T we stop the local optimisation. The assumption that the local optimisation will end with cost 0 is obviously not correct. But as before we take an optimistic point of view: if we would estimate the end cost to be higher, the time t^* would be larger, which implies that we would stop the process sooner.

2.3 Progress control

Using the (natural) logarithm in (3) leads to the linear function:

$$\log c(t) = \log A + Bt. \tag{4}$$

We use the least squares method to find the line $y = \alpha x + \beta$, approximating the points (x_k, y_k) where $x_k = t_k$ and $y_k = \log(c_k)$. A side effect of the least squares method is that big jumps in the costs have a relative large influence; since jumps are always downward, this will pull the curve (3) down. This seems good in our situation, where jumps correspond to an unstable situation, in which case some more patience seems appropriate.

We use our estimates of α and β to predict t^* each time we record a new value of c_n. For calculating t^* and taking the decision whether or not to stop we use three parameters p, f, and m:

(**p**) We use a *rolling horizon*: we forget about early points, and estimate the future progress only based on the last p points.

(**f**) We use a *start-up time*: for the first f points we record whether $t^* > T$ or not, but we never stop the process. If $f = \infty$ we perform ANS without progress control.

(**m**) We use *forgiveness*: we do not necessarily stop directly if some $t_n^* > T$, but only if this happened more than m times in the last p checks. Clearly we will never stop before t_m; hence it is useless to consider $f < m$. In particular if $m \geq p$, we do not stop at all.

We call a combination of these parameters (p, f, m) a *strategy*. Our main objective is to investigate the influence of a chosen strategy on the quality of the solution, assuming that we have a limited time at our disposal.

3 Results

The problem area we consider is personnel scheduling; the objects we try to schedule are shifts, which are fixed in time. The shifts have to be assigned to resources (employees), such that all hard constraints are satisfied, and the cost, resulting from trespassing soft constraints, is minimised. We use the Variable Neighbourhood Search as introduced in [1] as ANS. This algorithm is implemented in the advanced planning system HARMONY, developed by ORTEC for workforce management and scheduling.

We tested our progress control on five different datasets with different sizes. For each dataset we simulated 1000 runs and recorded the average values of some key figures for different strategies. In Table 1 we give per dataset the key figures for not applying any strategy (the lines with the '-') and for the best strategy found. The size of the problems is indicated by the column '# Em.'.

Name	# Em.	p^*	f^*	m^*	Bad stop	Bad cont.	Good stop	Good cont.	Tries	Time	Dec.	(Dec./ Time) *1000
DataA	12	-	-	-	0	318	0	682	1.5	1090	243	223
		14	3	0	315	3	586	96	10.1	948	426	449
Data2	8	-	-	-	0	476	0	524	1.9	3105	14	4.6
		33	3	1	470	6	478	46	22	2122	28	13.2
Data1	46	-	-	-	0	639	0	361	2.7	245	10.9	44
		3	2	0	639	0	239	122	8.2	112	14.6	130
ORTEC1	16	-	-	-	0	704	0	296	3.4	515	503	975
		46	3	2	646	58	229	67	14.7	459	605	1319
ORTEC2	16	-	-	-	0	800	0	200	4.8	952	324	341
		33	9	4	748	52	116	84	11.9	760	472	622

Table 1. Best found strategies for five different datasets and the average values of the key figures.

The columns 'Bad stop' and 'Bad cont.' show the number of 'bad' neighbourhoods that are stopped or continued by our strategy. A 'bad' neighbourhood is a neighbourhood where the local optimisation eventually does not reach a better solution than the best known so far. We see that the optimal strategies filter out almost all of the bad neighbourhoods, just as we hoped. However, in the colums 'Good stop' and 'Good cont.' we see that our strategies also stop a lot of good neighbourhoods. As a result we see in the column 'Tries' that our strategies need 3 to 10 times more neighbourhoods to find a better solution than not applying any strategy. But still, in the column 'Time' we can see that it takes our strategies *less* time to find a better solution. For Data1 we even see that we gain more than 50% in time! The column 'Dec.' shows the difference between the score to beat and the score of the solution found after the first improvement. Here we see that the improvement in score is 20-100% higher for our strategies compared

to not applying progress control. This is caused by the fact that we are likely to skip neighbourhoods that give only a small improvement.

We can conclude that our progress control finds *better* solutions and that it finds them *faster*. The combination of these effects is shown in the last column. Of course it will be very hard, if not impossible, to find the optimal strategy *before* the optimisation process. But we found out that most strategies are an improvement compared to not applying progress control at all, so finding a *good* strategy is not so hard.

4 Conclusions

We defined progress control as the process of estimating the progress of a local optimisation in a neighbourhood and using these estimates to stop searching 'bad' neighbourhoods. The results show that progress control is a good idea in our Shift Scheduling problem. Since we didn't use any knowledge of the problem or even of the VNS we used, we believe that it can be applicable to many versions of Iterated Local Search.

Finally, we like to stress that we do not pretend that our model predicts the expected value of c_∞; as can be noted in equation (3) we do not even try this. In fact, we are not even interested in the progress itself. We only try to predict whether or not a local optimisation will end up in a solution with a better score than the best known so far.

References

1. E. Burke, T. Curtois, G. Post, R. Qu, B. Veltman, *A hybrid heuristic and variable neighbourhoud search for the nurse rostering problem*, Proceeding of the 5th international conference on the Practice and Theory of Automated Timetabling (PATAT 2004), pp. 445-446 (2004).
2. N. Mladenović, P. Hansen, *Variable Neighborhood Search*, Computers & Operations Research **24**, pp. 1097–1100, (1997).