Author(s): Remde, S. M., Cowling, P. I., Dahal, K. P. and Colledge, N. J.

Title: Exact/heuristic hybrids using rVNS and hyperheuristics for workforce scheduling.

Publication year: 2007

Book title: Evolutionary Computation in Combinatorial Optimization.

ISBN: 978-3-540-71614-3

Publisher: Springer-Verlag.

Original publication is available at *http://www.springerlink.com*

Citation: Remde, S. M., Cowling, P. I., Dahal, K. P. and Colledge, N. J. (2007) Exact/heuristic hybrids using rVNS and hyperheuristics for workforce scheduling. In: Evolutionary computation in combinatorial optimization. Proceedings of the 7th European Conference (EvoCOP 2007) Valencia, Spain, April 11-13, 2007. pp 188-197.

Exact/Heuristic Hybrids Using rVNS and Hyperheuristics for Workforce Scheduling*

Stephen Remde, Peter Cowling, Keshav Dahal, and Nic Colledge

MOSAIC Research Group, University of Bradford, Great Horton Road
Bradford, BD7 1DP, United Kingdom
{s.m.remde, p.i.cowling, k.p.dahal, n.j.colledge}@bradford.ac.uk

Abstract. In this paper we study a complex real-world workforce scheduling problem. We propose a method of splitting the problem into smaller parts and solving each part using exhaustive search. These smaller parts comprise a combination of choosing a method to select a task to be scheduled and a method to allocate resources, including time, to the selected task. We use reduced Variable Neighbourhood Search (rVNS) and hyperheuristic approaches to decide which sub problems to tackle. The resulting methods are compared to local search and Genetic Algorithm approaches. Parallelisation is used to perform nearly one CPU-year of experiments. The results show that the new methods can produce results fitter than the Genetic Algorithm in less time and that they are far superior to any of their component techniques. The method used to split up the problem is generalisable and could be applied to a wide range of optimisation problems.

1 Introduction

In collaboration with an industrial partner we have studied a workforce scheduling problem which is a resource constrained scheduling problem similar to but more complex than many other well-studied scheduling problems such as the Resource Constrained Project Scheduling Problem (RCPSP) [1] and job shop scheduling problem [2]. The problem is based on our work with @Road Ltd. which develops scheduling solutions for very large, complex mobile workforce scheduling problems in a variety of industries. Our workforce scheduling problem is concerned with assigning people and other resources to geographically dispersed tasks while respecting time window constraints and skill requirements.

The workforce scheduling problem that we consider consists of four main components: Tasks, Resources, Skills and Locations. Unlike many RCPSP problems, the tasks have locations and a priority value (to indicate relative importance). Resources are engineers and large pieces of equipment. They are mobile, travelling at a variety of speeds to geographically dispersed tasks. Tasks and resources have time windows with different associated costs (to consider, for example, inconvenience to

* This work was funded by EPSRC and @Road Ltd under an EPSRC CASE studentship, which was made available through and facilitated by the Smith Institute for Industrial Mathematics and System Engineering.

C. Cotta and J. van Hemert (Eds.): EvoCOP 2007, LNCS 4446, pp. 188 – 197, 2007.

customers at certain times, the cost of overtime, etc.). Tasks require a specified amount of specified skills, and resources possess one or more of these skills at different competencies which affects the amount of time required. A major source of complexity of our problem comes from the fact that a task's duration is unknown until resources are assigned to it. In this paper, the fitness of a schedule is given by one of the single weighted objective functions used in [3], $f = SP - 4SC - 2TT$, where SP is the sum of the priority of scheduled tasks, SC is the sum of the time window costs in the schedule (both resource and task) and TT is the total amount of travel time. This objective is to maximise the total priority of tasks scheduled while minimising travel time and cost. [3] describes the problem in more detail and uses a Genetic Algorithm to solve it. In this paper we will compare the Genetic Algorithm method with a new reduced Variable Neighbourhood Search and hyperheuristic methods.

We propose a method to break down this "messy" problem by splitting it into smaller parts and solving each part using exact enumerative approaches. Hence each part consists of finding the optimal member of a local search neighbourhood. We then design ways to decide which part to tackle at each stage in the solution process. These smaller parts are the combination of a method to select a task and a method to select resources for the task. We will take these smaller parts and use reduced Variable Neighbourhood Search and hyperheuristics to decide the order in which to solve them.

This paper is structured as follows: we present related work in section 2 and propose reduced Variable Neighbourhood Search and hyperheuristic approaches in section 3. In section 4 we empirically investigate the new techniques and compare them to a genetic algorithm in terms of solution quality and computational time. We present conclusions in section 5.

2 Related Work

The RCPSP [1] involves a set of tasks which have to be scheduled under resource and precedence constraints. Precedence constraints require that a task may not start until all its preceding tasks have finished. Resource constraints require specified amounts of finite resources to be available when the task is scheduled. Scheduling an RCPSP involves assigning start times to each of the tasks. The RCPSP is a generalisation of many scheduling problems including job-shop, open-shop and flow-shop scheduling problems. The RCPSP has no notion of variable time dependant on skill or location of tasks and resources. The time line is also discrete and assumes resources are always available.

The Multimode Resource Constrained Resource Scheduling Problem (MRCPSP) extends the RCPSP [4]. In the MRCPSP, there is the option of having non-renewable resources and resources that are only available during certain periods. In addition, a task maybe executed in one of several execution modes. Each execution mode has different resource requirements and different task durations. Usually the number of these modes is small and hence exact methods can be used. In the workforce scheduling problem considered in this paper, we have a very large number of execution modes (as the task duration depends on the resource competency and the task skill requirement which are both real values). [5] uses a genetic algorithm as a solution to problems where using an exact method is intractable. [6] surveys heuristic solutions to the RCPSP and MRCPSP.

Solution methods such as Genetic Algorithms (GAs) were introduced by Bremermann [7] and the seminal work done by Holland [8]. Since then they have been developed extensively to tackle problems including the travelling salesman problem [9], bin packing problems [10] and scheduling problems [11]. A Genetic Algorithm tries to evolve a population into fitter ones by a process analogous to evolution in nature. Our previous work [3] compares a multi-objective genetic algorithm to a single weight objective genetic algorithm to study the trade-off between diversity and solution quality. The genetic algorithm is used to solve the dynamic workforce scheduling problem studied in this paper.

Variable Neighbourhood search (VNS) is a relatively new search technique and the seminal work was done by Mladenović and Hansen [12]. VNS is based on the idea of systematically changing the neighbourhood of a local search algorithm. Variable Neighbourhood Search enhances local search using a variety of neighbourhoods to "shake" the search into a new position after it reaches a local optimum. Several variants of VNS exist as extensions to the VNS framework [13].

Reduced Variable Neighbourhood search (rVNS) [13] is an attempt to improve the speed of variable neighbourhood search (with the possibility of a worse solution). Usually, the most time consuming part of VNS is the local search. rVNS picks solutions randomly from neighbourhoods which provide progressively larger moves. rVNS is targeted at large problems where computational time is more important than the quality of the result. In combinatorial optimisation problems, local search moves like "swap two elements" are frequently used, and [14] for RCPSP as well as others such as [15], apply VNS by having the neighbourhoods make an increasing number of consecutive local search moves. [16] however defines only two neighbourhoods for VNS applied to the Job Shop Scheduling Problem, a swap move and an insert move, which proves to be effective.

VNS can be seen as a form of hyperheuristic where the neighbourhoods and local search are low level heuristics. The term "hyperheuristic" was introduced in [17]. Hyperheuristics rely on low level heuristics and objective measures which are specific to the problem. The hyperheuristic uses feedback from the low level heuristics (CPU time taken, change in fitness, etc.) and determines which low level heuristics to use at each decision point. Earlier examples of hyperheuristics include [18] where a genetic algorithm evolves a chromosome which determined how jobs were scheduled in open shop scheduling. A variety of hyperheuristics have been developed including a learning approach based on the "choice function" [17], tabu search [19], simulated annealing [20] and Genetic Algorithms [21].

3 Heuristic Approaches

Our proposed framework splits the problem into (1) selecting a task to be scheduled and (2) selecting potential resources for that task. A task is randomly chosen from the top two tasks which we have not tried to schedule ranked by the task order, to make the search stochastic, to ensure that running it multiple times will produce different results. We have implemented 8 task selection methods given in table 1. Note that some of our task orders are deliberately counterintuitive to give us a basis for comparison.

Table 1. Task sorting methods

Method	Description
Random	Tasks are ordered at random.
PriorityDesc	Tasks are ordered by their priority in descending order
PriorityAsc	Tasks are ordered by their priority in ascending order
PrecedenceAsc	Tasks are ordered by their number of precedences ascending
PrecedenceDesc	Tasks are ordered by their number of precedences descending
PriOverReq	Tasks are ordered by their estimated priority per hour assuming the task will take as long as the total skill requirement
PriOverMaxReq	Tasks are ordered by their estimated priority per hour assuming the task will take as long as the maximum skill requirement
PriOverAvgReq	Tasks are ordered by their estimated priority per hour assuming the task will take as long as the average skill requirement

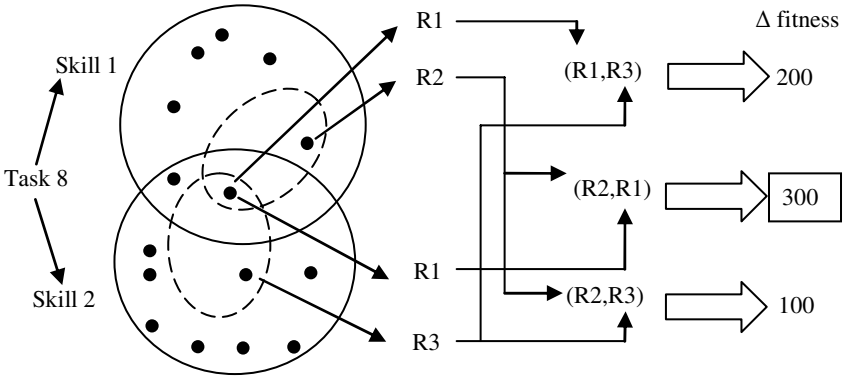

Fig. 1. Resource Selector. The dotted subset of resources possessing the required skill is chosen by a Resource Selector. The assignment (R2, R1) is chosen as the best insertion.

PriorityDesc, *PriOverReq*, *PriOverMaxReq* and *PriOverAvgReq* are attempts to identify the tasks which will give us the most reward and schedule them first. They estimate the task duration differently and use this estimate to calculate priority hour. *PrecedenceDesc* attempts to schedule those tasks with the largest number of succeeding tasks first. *PrecedenceAsc*, *PriorityAsc* and *Random* give us some indication of the effect of task orders since intuition would suggest that they should give poor results.

We then define Resource Selectors which select a set of potential resources for each skill required by the selected task. The Resource Selectors first sort the resources by their competencies at the skill required and then select a subset of them. This could be, for example, the top five or the top six to ten etc. The subsets of resources are then enumerated and exhaustive search used to find the insertion which will yield the lowest time window and travel penalties subject to precedence constraints. Figure 1 illustrates this.

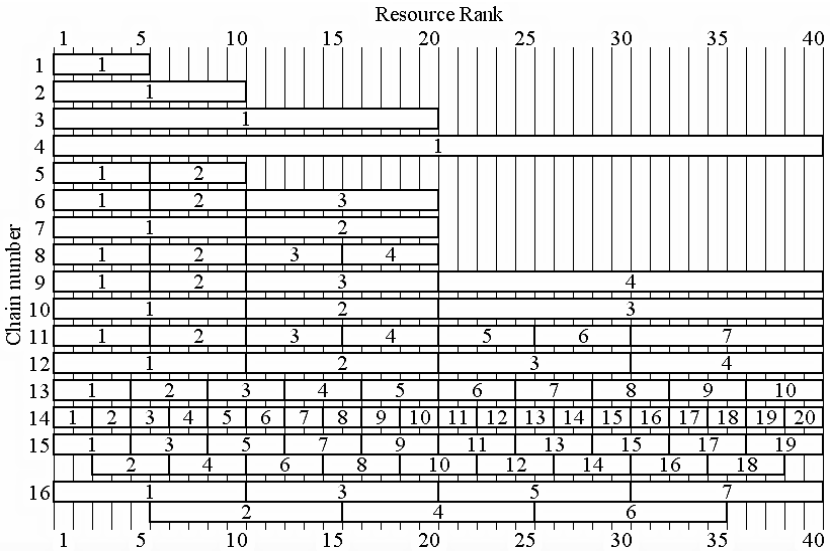

Fig. 2. Resource selection "chains" for the rVNS

```
k is the index of the resource selector in use
(N1, N2, … N_kmax ) is our chain of resource selectors
Sort tasks using the chosen task order
k:=1
while (k<kmax)
    for each Unscheduled Task T
        Select Sets of Resources Using Nk for Task T
        Exhaustively Search the selected sets of resources
                to find an optimal insertion I
        Insert task T into the schedule using I
    next
    if some tasks were inserted then
        k:=1
    else
        k:=k+1
    end if
end while
```

Fig. 3. Pseudo code for our rVNS method

The neighborhoods of our rVNS insert tasks selected by a task order using a given resource selector. If an insertion is not possible, because of resource or task constraints, we try the next resource selector and so on. We consider several sequences of resource selection neighborhoods, or "chains", as shown in figure 2. These neighborhoods show a progression of an increasing range of resources used and

smaller sets. Figure 3 shows the pseudo code for our rVNS method. Allowing search to restart at the start of the chain allows the search to retry insertions that may have failed before because of resource or task constraints. If s is the maximum number of skills and n is the number of tasks, then the algorithm has time complexity $O(nk|N_k|^s)$ for each chain. With the 16 resource selection chains and the 8 task orders we have defined, we have 128 different rVNS methods.

Our first hyperheuristic, *HyperRandom*, selects at random a Low Level Heuristic (i.e. a (task order, resource selector) pair) to use at each iteration and applies it if its application will result in a positive improvement. This continues until no improvement has been found for a certain number of iterations. The second, *HyperGreedy*, evaluates all the Low Level Heuristics at each iteration and applies the best if it makes an improvement. This continues until no improvement is found. The low level heuristics are the combination of a task selector and a resource selector.

The genetic algorithm we will use is that of [3]. The chromosome represents an order of tasks to be scheduled by a serial scheduler. The initial population is generated randomly and the task order is evolved. The way in which the tasks are inserted into the schedule is a fast naïve approach as schedule must be generated many times per generation. The serial scheduler takes the next task from the chromosome and allocates resources to it greedily skill by skill. A resource is selected by finding the resource which has the greatest amount of available time in common with the task's time windows and any other resources already selected. After each skill has been allocated a resource, it is inserted into the schedule as early as possible. We use a population size of 50, mutation rate of 1%, and a crossover rate of 25% using Uniform Crossover based on out previous experience [3]. The GA is run for 100 generations (or for a maximum of 2.5 hours) and the result is the fittest individual in the final population.

4 Computational Experiments

To compare the methods for solving the problem, we use each method (one Genetic Algorithm, 128 rVNS and two hyperheuristics) on five different problem instances. The five problem instances require the scheduling of 400 tasks using 100 resources over one day using five different skills. Tasks require between one and three skills and resources possess between one and five skills. The problems are made to reflect realistic problems @Road Ltd. have identified and are generated using the problem generator used in [3].

Each method is used for five runs of the five instances and an average taken of the 25 results. To ensure fairness, each method is also run for a 2.5 hour "long-run" where the 25 results are repeatedly generated and the best average over all there repeated runs is reported. As these experiments require nearly a CPU year to complete (five runs of five instances using 131 different methods lasting 2.5 hours each = 8187.5 CPU hours) they were run in parallel on 60 identical 3.0 GHz Pentium 4 machines. Implementation was in C# .NET under Windows XP.

Figure 4 shows the results of the 2.5 hour "long run" for each rVNS approach. Results for a single run of each approach are similar but 1-4% worse on average. The intuitively "bad" task orders, *PriorityAsc* and *Random* are clearly shown to be worse than the intuitively reasonable orders such as *PriorityDesc*. Measure based on

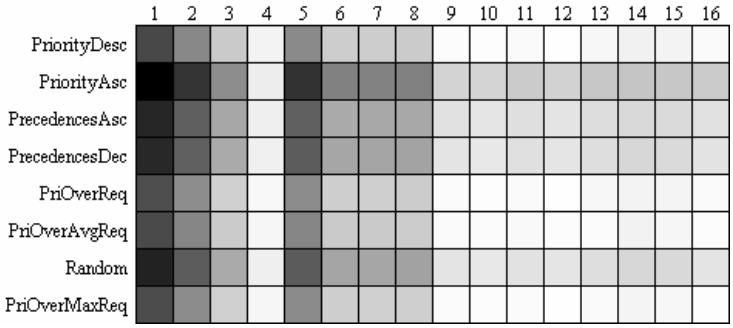

Fig. 4. Heat graph of the performance of rVNS methods for 2.5 hour "long run". Black = 4472, White =26525.

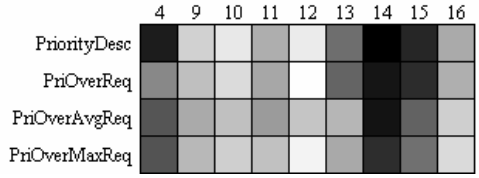

Fig. 4a. Heat graph of the performance of selected rVNS methods for 2.5 hour "long run". Black = 25398, White =26525.

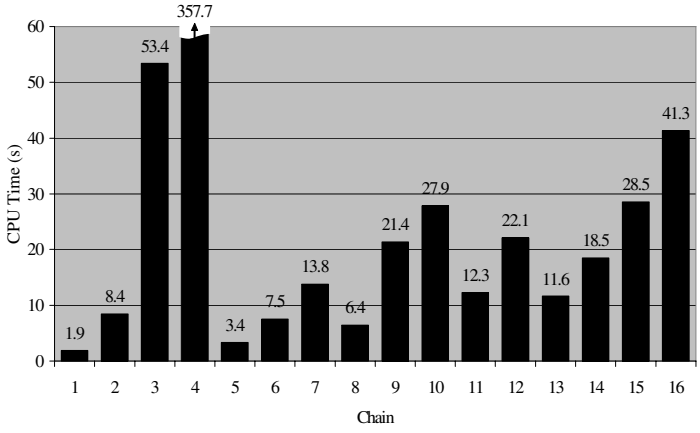

Fig. 5. Average CPU time taken by each chain used in the rVNS methods

decreasing priority or priority per hour (*PriorityDesc*, *PriOverReq*, *PriOverAvgReq*, *PriOverMaxReq*) are superior to other measures. Figure 4a compares the best approaches in detail. Chain 12 produces the best results for all task orders. It is clear to see the correlation between results with common chains or task orders. Chain 4 demonstrates that trying to estimate priority per hour is superior to *PriorityDesc*. This is probably because with a limited amount of free time in the schedule, using tasks that have lower priority but can be completed in a shorter time is more beneficial.

Figure 5 compares the CPU time for a single run of rVNS using each chain. It is clear that the approach would scale to very large problems using small resource selection sets such as for chains 1, 5, 6, 8, 11 and 13. Moreover, it appears from figure 4 that little solution quality is lost when covering the resources with small subsets rather than larger ones as in chain 4, but the CPU times are significantly reduced. Chain 12 yields the best results of the chains which take reasonable amounts of CPU time, and clearly outperform chain 2 and chain 7 which do not consider the whole set of resources. It seems that resources of poor competence must be considered to get the best possible results.

Table 2 shows the best result from the rVNS (Chain 12, Task Order *PriOverReq*) compared with GA and the hyperheuristic methods. They quite clearly show that *HyperGreedy* provided the fittest results on average while using more CPU time. The GA provided the worst result and in the slowest time. This may result from its insertion heuristic, however implementing a better one would make it even slower. The rVNS is the fastest method we have tested and provides results nearly 20% better than the GA in less than 1/350 of the CPU time required. Exactly solving small sub problems appears very effective in this case.

Table 2. GA, rVNS and Hyper-Heuristic Results for one run and long run

Method	Fitness (single run average)	CPU Time (s)	Fitness (after 2.5 hours)
GA	21401.3	9000.0	21401.3
rVNS (Best)	25662.5	25.1	26215.1
HyperRandom	24525.4	78.3	25645.4
HyperGreedy	26523.6	419.2	27103.1

HyperRandom performs poorly compared to the best rVNS method. rVNS task selectors and resource selectors are sensible guesses which significantly improve on the random approach of *HyperRandom*. The resource selectors of the rVNS tend to select resources which are of similar competence, so that a high competence resource is not combined with a low-competence resource (which might tie up the time of a high-competence resource).

The *HyperRandom*, and the *HyperGreedy* heuristics try significant numbers of bad low level heuristics which make local improvements which in the long run are far from optimal. In the case of the *HyperGreedy* method, the bad low level heuristics are evaluated every iteration which wastes CPU time. Analysis of the low level heuristics used in the *HyperGreedy* method was performed and show that 19 (26.4%) of the low level heuristics were never used and 56 (77.7%) of the low level heuristics were used less than one percent of the time. Figure 6 analyses the low level heuristics (LLHs) used. It shows the top 20 LLHs used together with when they are used in schedule generation. First third, middle third and last third show the usage at different stages in the scheduling process – from when the schedule is empty and unconstrained to when the schedule is almost full and inserting a task is more difficult. From these results it is clear that different LLHs contribute at different stages of the solution process, and that many different LLHs provide a contribution. For example, LLH 32 is more

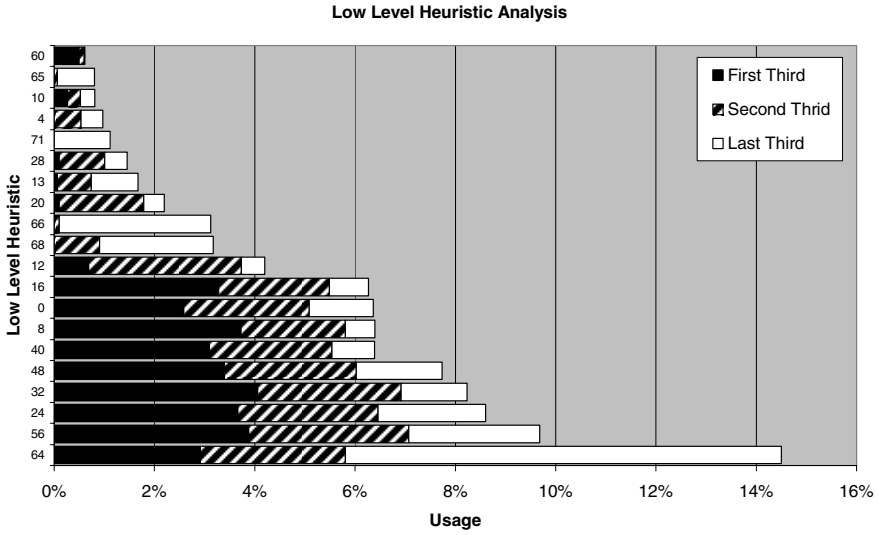

Fig. 6. Usage of low level heuristics throughout the HyperGreedy search

effective at the start of scheduling, LLH 12 is more effective in the middle and LLH 64 is more effective at the end. Without access to a large number of LLHs it seems that solution quality would be much reduced.

5 Conclusions

In this paper we have compared a large number (128) of reduced Variable Neighbourhood Search (rVNS) approaches to hyperheuristics and Genetic Algorithm approaches for workforce scheduling problem. We have demonstrated the effectiveness of heuristic/exact hybrids which find optimal subproblem solutions using an enumerative approach. Our rVNS method can produce good results to large problems in low CPU time. Our hyperheuristics produce even better results using more CPU time and we showed that the hyperheuristic uses a range of low level heuristics throughout the search process.

The hyperheuristics we used are simple and learning could potentially decrease CPU time and increase fitness. In future work we intend to implement learning mechanisms. We have seen from the analysis that many low level heuristics were never used and some used mainly at the beginning, middle or end. Learning the low level heuristics behaviour could potentially lead to better solutions in less time.

References

1. Hartmann, S.: Project Scheduling under Limited Resources: Model, methods and applications. Springer-Verlag, Berlin Heidelberg, New York (1999)
2. Pinedo, M. and Chao, X.: Operations scheduling with applications in manufacturing and services. McGraw-Hill, New York (1999)

3. Cowling, P., Colledge, N., Dahal, K. and Remde, S.: The Trade Off between Diversity and Quality for Multi-objective Workforce Scheduling. Evolutionary Computation in Combinatorial Optimization, Proc. Lecture Notes in Comp. Science 3906: 13-24 (2006)
4. Kolisch, R.: Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Oper. Res. 90 (2): 320-333 Apr 19 (1996)
5. Alcraz, J., Marotom R., and Ruiz, R.: Solving the Multi-mode Resource-Constrained Project Scheduling Problems with genetic algorithms. Journal of Operational Research Society (2004) 54, 614-626
6. Kolisch, R. and Hartmann, S.: Experimental Investigations of Heuristics for RCPSP: An Update. European Journal of Oper. Res. 174 (1): 23-37 (2006)
7. Bremermann, H.: The evolution of Intelligence. The Nervous System as a Model of it's environment. Technical Report No 1, contract No 477(17) Dept. of Math., Univ. of Washington, Seattle. (1958)
8. Holland, J. H.: Adaptation in Natural and Artificial Systems, Ann Abor, MI: University of Michigan Press, Michigan. (1975)
9. Whitley, D., Starkweather, T. and Shaner, D.: The travelling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In Handbook of Genetic Algorithms, New York: Van Nostrand Reinhold (1991)
10. Falkenauer, E.: A Hybrid Grouping Genetic Algorithm for Bin Packing. Journal of Heuristics, vol 2, No. 1, 5-30 (1996)
11. Ross, P., Hart E. and Corne, D.: Some observations about GA-based exam timetabling. Lecture Notes in Computer Science 1408: 115-129 (1998)
12. Mladenovic, N. and Hansen, P.: Variable neighborhood search. Computers & Operational Research 24 (11): 1097-1100 Nov (1997)
13. Hansen, P. and Mladenovic, N.: Variable neighborhood search: Principles and applications. European Journal of Oper. Res. 130 (3): 449-467 May 1 (2001)
14. Fleszar, K. and Hindi, K.S.: Solving the resource-constrained project problem by a variable neighbourhood scheduling search. European Journal of Oper. Res. 155 (2): 402-413 Jun 1 (2004)
15. Garcia, C.G., Perez-Brito, D., Campos, V. and Marti, R.: Variable neighborhood search for the linear ordering problem. Comp. & Oper. Research 33 (12): 3549-3565 Dec (2006)
16. Sevkli, M., Aydin, M.E.: A variable neighbourhood search algorithm for job shop scheduling problems. Lecture Notes in Computer Science 3906: 261-271 (2006)
17. Cowling, P., Kendall, G. and Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. PATAT III Springer LNCS 2079: 176-190 (2001)
18. Fang, H., Ross, P. and Corne, D.: A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. 11th European Conference on Artificial Intelligence, (1994)
19. Burke, E. K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics 9 (6): 451-470 Dec (2003)
20. Bai, R. and Kendall, G.: An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics. In proc. of The Fifth Metaheuristics Int. Conf. (2003)
21. Kendal, G., Han, L. and Cowling, P.: An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. CEC, IEEE Press (2002) 1185-1190