Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 13 (6): 883-888, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.883.888

Adaptive Guided Variable Neighborhood Search

Rafidah Abdul Aziz, Masri Ayob, Zalinda Othman and Hafiz Mohd Sarim
Data Mining and Optimization Research Group, Centre of Artificial Intelligence Technology,
Fakulti Teknologi dan Sains Maklumat,
University Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia

Abstract: The basic idea of a Variable Neighborhood Search (VNS) algorithm is to systematically explore a neighborhood of a solution using a set of predefined neighborhood structures. Since different problem instances have different landscapes and complexities whilst different neighborhood structures may lead to different solution spaces, the choice of which neighborhood structure to be applied is a challenging task. Therefore, this work proposes an Adaptive Guided Variables Neighborhood Search (AG-VNS). AG-VNS has two phases. First, is a learning phase which is used to memorize neighborhood structures that can effectively solve specific soft constraint violations by applying the neighborhood structures to the best solution. These steps are repeated until a stopping condition for the learning phase is met. Second, is an improvement phase which is used to enhance the quality of a current best solution by selecting the most suitable neighborhood structure from memory that will be applied to the current solution. Its effectiveness is illustrated by solving course time tabling problems. The performance of the AG-VNS is tested over the Socha course time tabling datasets. Results demonstrated that the performance of the AG-VNS is comparable with the results of the other VNS variants, while outperforming some variants in particular instances. This demonstrates the effectiveness of applying the adaptive learning mechanism to guide the selection of the neighborhood structures in the VNS algorithm.

Key words: Course time tabling, variable neighborhood search, adaptive search

INTRODUCTION

Variable Neighbourhood Search (VNS) is an improvement meta-heuristic that has been proposed by VNS deals with several neighbourhood structures and therefore, the decision to apply which neighbourhood at the current iteration in crucial (Thompson and Dowsland, 1995), due to the fact that different neighborhood structures generate different landscapes. Literature shows that VNS has been applied to solve challenging optimization problems such as course time tabling.

However, up to date, there has been no work undertaken to solve course time tabling problems by applying an adaptive search in VNS. Motivated by the above, this work proposes an adaptive learning mechanism in the VNS algorithm in order to guide the VNS algorithm to choose the right neighborhood structure during the search. The proposed approach is tested over the Socha course time tabling datasets (Socha *et al*., 2003). Results demonstrate that the adaptive learning mechanism in the VNS can enhance the performance of the AG-VNS to obtain better solutions (compared with other variants of VNS) for the course time tabling problem.

PROBLEM DESCRIPTION

University course time tabling problems can be defined as assigning a given number of courses to a given number of timeslots and rooms subject to a set of hard and soft constraints. This work used the Socha datasets (Socha *et al*., 2003) which are modeled as follows:

- A set of C courses c_i (i = 1,…,C)
- t_n represents the set of timeslots (n = 1,…,45)
- A set of R rooms r_j (j = 1,…, R)
- A set of F room features
- A set of M students

The course time tabling problem consists of assigning every course c_i to a timeslot t_n and room r_j so that the following hard constraints are satisfied:

- No student can be assigned to more than one course at the same time
- The room should satisfy the features required by the course

Corresponding Author: Masri Ayob, Data Mining and Optimization Research Group,
Centre of Artificial Intelligence Technology, Fakulti Teknologi dan Sains Maklumat,
University Kebangsaan, Malaysia, 43600 UKM Bangi, Selangor, Malaysia
Tel: +603 8921 6741 Fax: +603 8921 6184

- The number of students attending the course should be less than or equal to the capacity of the room
- No more than one course is allowed in a timeslot in each room

The objective is to satisfy all hard constraints while minimizing the number of students involved in the violation of soft constraints. The soft constraints are equally penalized (penalty cost = 1 for each violation per student). The soft constraints are:

- A student should not have a course scheduled in the last timeslot of the day
- A student should not have more than two consecutive courses
- A student should not have a single course on a day

Table 1: Eleven instances of socha datasets (Socha *et al*., 2003)

Instances	Course	Room	Feature	Student
Small 1	100	5	5	80
Small 2	100	5	5	80
Small 3	100	5	5	80
Small 4	100	5	5	80
Small 5	100	5	5	80
Medium1	400	10	5	200
Medium 2	400	10	5	200
Medium 3	400	10	5	200
Medium 4	400	10	5	200
Medium 5	400	10	5	200
Large	400	10	10	400

Table 1 presents the characteristics of the Socha datasets (Socha *et al*., 2003). The datasets consists of 11 problem instances which are categorized as small, medium and large.

VNS-BASIC FOR COURSE TIME TABLING PROBLEMS

Contrary to other local search methods, VNS does not follow a trajectory but explores increasingly distant neighborhoods of the current incumbent solution and jumps from this solution to a new one if and only if an improvement has been made (Hansen and Mladenovic, 2001). Figure 1 shows the pseudocode of the basic VNS (Hansen and Mladenovic, 2001).

Figure 2 shows the pseudocode of the VNS-Basic for solving course time tabling problems. In this work, the initial solution is produced using a Largest Degree Graph.

Coloring heuristic: The feasible timetable is obtained by adjusting appropriate courses in the schedule based on room availability and other hard constraint violations until all the hard constraints are satisfied without taking into account any of the soft constraint violations.

This study used the following neighborhood structures:

Initialization: Select the set of neighborhood structures N_k, k = 1,…,k_{max}, that will be used in the search; find an initial solution x; choose a stopping condition;

Repeat the following until the stopping condition is met:
(1) Set k←1; (2) Until k = k_{max},repeat the following steps
(a) **Shaking:** Generate a point x' at random from the kth neighborhood of x(x'∈ N_k(x));
(b) **Local search:** Apply some local search method with *x'* as an initial solution; denote with x" (the obtained local optimum);
(c) **Move or not:** If this local optimum is better than the incumbent, move there (x←x") and continue the search with N_1 (k←1); otherwise, set k←k+1;

Fig. 1: Steps of the basic VNS

Initialization:
- Let N_k be a set of neighborhood structures that will be used in the search, where k = 1,..., k_{max},
- Let N_p be a set of neighborhood structures that will be used in the local search, where p = 1,…,p_{max},
- Set the best solution (x_{best}) = given initial solution (x^0), where f(x_{best}) = f(x^0),
- Choose a stopping condition,

Repeat the following until the stopping condition is met:
(1) Set k←1; (2) Until k = k_{max},repeat the following steps:
(a) Shaking. Generate a solution x' at random from the k^{th} neighborhood of x_{best}(x'∈N_k(x_{best}));
(b) Local search. Apply VND with x' as initial solution;
(I) Set l←1; (ii) Until l = l_{max}, repeat the following steps:
(a) Find the best neighbor x" of x';
(b) Move or not:
If the solution thus obtained x" is better than the x', set x'←x" and continue the search with N_1 (p ← 1); otherwise, set p←p+1;
(c) Move or not: If the solution (x') is better than the x_{best}, move there (x_{best}←x') and continue the search with N_1 (k←1); otherwise, set k←k+1;

Fig. 2: Steps of the VNS-Basic for solving course time tabling problems

- **(N1) Swap:** Select two timeslots at random and simply swap all the courses in one timeslot with all the courses in the other timeslot while maintaining a feasible timetable
- **(N2) Swap:** Select any last timeslot of the day and another single timeslot (exclude the last timeslots of the day) and simply swap all the courses in the last timeslot with all the courses in the other timeslot while maintaining a feasible timetable
- **(N3) Move:** Select any 5 events from any last timeslot of the day and move it to a new free timeslot (exclude the last timeslots of the day) while maintaining a feasible timetable
- **(N4) Swap:** Select any 2 events at random and simply swap their timeslot while maintaining a feasible timetable
- **(N5) Move:** Select any 5 events from any timeslot and move it to a new free timeslot (exclude the last timeslots of the day) while maintaining a feasible timetable

AG-VNS ALGORITHM PROPOSED FOR COURSE TIME TABLING PROBLEMS

The basic idea of AG-VNS is to guide the VNS algorithm to choose the right neighborhood structure at the right time. Therefore, the adaptive learning mechanism is introduced. With this mechanism, the algorithm memorizes which neighborhood structure could effectively solve the specific soft constraint violations. Then, the algorithm uses the memory information as a guide for selecting the right neighborhood structure to enhance the quality of the best solution. For this purpose, AG-VNS is divided into two phases. Phase I is for learning while Phase II is for improvement.

Phase I: Learning: In this phase, the algorithm memorizes which neighborhood structure could effectively solve the specific soft constraint violations. For this purpose, the memory is divided into three parts (which represents the three soft constraints S_i where i = 1, 2, 3) and each part contains five spaces (representing the neighborhood structures) to keep a total of the reduction values of the soft constraint violations.

This phase begins when the algorithm generates a solution x' by applying N_k (where k = 1) to the initial solution x^0. If the solution x' is better, the algorithm will identify which soft constraint violations have reduced. The reduction value is calculated and updated in the memory. After that, the algorithm repeats the same steps by applying the neighborhood structure N_k (where k = 2 until k = k_{max}). The procedures are repeated until the stopping condition for the Phase I is met.

For example, given an initial solution x^0, where $f(x^0) = S^0_1+S^0_2+S^0_3$ and after applying neighborhood structure N_k (where k = 1), an incumbent solution x' is generated where $f(x') = S'_1 + S'_2 + S'_3$. If f(x') is less than $f(x^0)$, then the algorithm will identify which soft constraint violations was/were reduced. For example, if $S'_1<S^0_1$, the reduction value is calculated. Therefore, the reduction value of the soft constraint violations (d) = $S^0_1-S'_1$. Then, the value d is updated in the memory at the space 1 (which represents the first neighborhood structure) in part 1 (which represents the soft constraint 1 (S_1).

Table 2 illustrates an example of the memory contents after the stopping condition of learning phase (phase I) has been met. Based on the table, the highest total of the reduction value for the soft constraint 1 (S_1) is 123 has been obtained after applied neighborhood structure 1 (N_1) to the solution x^0. This means that N_1 is the most appropriate neighborhood structure to be selected to solve soft constraint 1 while simultaneously enhance the quality of the solution x" (in phase II). N_2, on the other hand, is the most appropriate neighborhood structure to be selected to solve soft constraint 2 (S_2) and soft constraint 3 (S_3). This is because the highest total reduction value for soft constraint 2 (S_2) is 88 has been obtained after applied the neighborhood structure 2 (N_2) to the solution x^0 and the highest total the reduction value for the soft constraint 3 (S_3) is 78 also has been obtained after applying neighborhood structure 2 (N_2) to the solution x^0. This information will be used in the phase II.

Phase II: Improvement: In this phase, the algorithm uses the memory (from Phase I) as a guide for selecting the most appropriate neighborhood structure to enhance the quality of a best solution. For the first iteration, the initial solution x^0 is set to be the best solution x_{best}.

This phase starts when the algorithm generates a solution x" by randomly applying any of the neighborhood structures (N_k) to the best solution x_{best} Then, the algorithm identifies the highest soft constraint violations in the solution x". Subsequently, the algorithm uses the memory information from Phase I to select the most appropriate neighborhood structure to solve the soft constraint violations to enhance the quality of the solution x". For example, if the highest soft constraint

Table 2: Example of the memory contents after the stopping condition of learning phase (phase i) has been met

	Soft constraints (S_i, where i = 1, 2, 3)		
N_k (where k = 1 until k = k_{max})	S_1	S_2	S_3
1	123	34	23
2	59	88	78
.	89	42	5
.			
.			
k_{max}	43	54	12

Initialization:
Let N_k be a set of neighborhood structures that will be used in the Phase I, where k = 1,..., k_{max};
Let Np be a set of neighborhood structures that will be used in the Phase II, where p = 1,…,p_{max};
Set the given initial solution x^0 as the best solution x_{best};
Set the stopping condition I and II;

Phase I: Learning
Repeat
For k = 1 to k_{max},
- **Shaking:** Randomly generate a solution x' by applying N_k to the initial solution x^0
  If the solution x' is better than the initial solution x^0
  i. Identify which soft constraint violation has decreased
  ii. Calculate the decreased value
  iii. Update the value in the memory

End if
Next k
Until stopping condition I is met

Phase II: Improvement
Repeat
- **Shaking:** Generate a solution x" by randomly applying any of the neighborhood structures Nk to the solution xbest
- Identify the soft constraint (Si) which has the highest violations in the solution x"
- Generate a solution x* by selecting the right neighborhood structure to be applied to the solution x".

If f(x*) is better than f(xbest) then
xbest←x*;
Update memory; // repeat step (i) to step (iii) as in phase I
Else:
For p= 1 to pmax
Generate solution x** by applying Np to the solution x"
If f(x**) is better than f(x") then
x"←x**;
Update memory; // repeat step (i) to step (iii) as in phase I
Exit For
Else : Set p ¬ p+1
End if
Next p
If f (x") is better than f(xbest) then
(xbest←x")
End if
End if
Until stopping condition II is met

Fig. 3: Pseudo code of the AG-VNS

violations in the solution x" is S_1, the algorithm will refer to the memory to select the most appropriate neighborhood structure to solve the soft constraint 1 while enhancing the quality of the solution x". Based on the Table 2, the highest total reduction value for S_1 is from N_1. Therefore, N_1 is used to enhance the quality of the solution x". If it turns out that the solution x* produced by that step is better than the best solution x_{best}, the best solution x_{best} is updated by the solution x* and the memory is updated by repeating step (i) to step (iii) as in the first phase.

In instances where solution x* is found to be worse than the best solution x_{best}, the algorithm will apply the neighborhood structure N_p to the solution x". If the solution x** produced from that step is better than the solution x", then the solution x" is updated by the solution x^{**} and the memory is updated by repeating step (i) to step (iii) as in the first phase. Otherwise, the algorithm continues the search with N_p (where p←p+1). These steps are repeated until the solution x^{**} produced is better than the solution x" or p = p_{max}. Finally, if the solution x" is better than the solution x_{best}, the solution x_{best} is updated by the solution x". The algorithm proceeds to the next iteration until the stopping condition for the phase 2 is met. Figure 3 shows the pseudocode of the AG-VNS.

RESULTS AND DISCUSSION

In order to assess the effectiveness of applying learning mechanism in the VNS, the same neighborhood structures are used to compare AG-VNS with VNS-Basic. The stopping conditions for the AG-VNS that are used in phase I and phase II are 2000 and 100,000 iterations, respectively. These values were determined based on preliminary experiment. Table 3 presents the average penalty cost for 20 runs.

It is clear from Table that AG-VNS outperforms VNS-Basic. This is due to the use of the adaptive learning mechanism strategy to guide the algorithm to choose the right neighborhood structure at the right time. The performance of AG-VNS has been compared with other approaches in the literature. Table 4 presents the comparison results. These are the works that has been compared in this experiment:

- M1: Variable neighborhood search with EMC (Abdullah, 2006)
- M2: Variable neighborhood search with Tabu (Abdullah, 2006)
- M3: Non-linear Decay Rate (Abdullah and Turabieh, 2008)
- M4: Genetic Algorithm with a sequential local search (Jat and Yang, 2009)

Table 3: Comparison results on course time tabling Problem between vns-basic and ag-vns

	Iteration : 100,000 (20 runs)	
Instances	VNS-Basic	AG-VNS
Small1	3	0
Small2	5	0
Small3	2	0
Small4	4	0
Small5	1	0
Medium1	438	130
Medium2	567	157
Medium3	478	162
Medium4	455	125
Medium5	507	123
Large	1074	784

Table 4: Comparison results on course time tabling problem between ag-vns and other approaches

Datasets	AG-VNS	M1	M2	M3	M4
Small 1	0	0	0	3	2
Small 2	0	0	0	4	4
Small 3	0	0	0	6	2
Small 4	0	0	0	6	0
Small 5	0	0	0	0	4
Medium 1	130	338	317	140	254
Medium 2	157	326	313	130	258
Medium 3	162	384	357	189	251
Medium 4	125	299	247	112	321
Medium 5	123	307	292	141	276
Large	784	945	926	876	1027

It can be seen that across all problem instances, AG-VNS has produced much better results when compared against other approaches in literature. For the five small instances, AG-VNS is able to solve them to optimality. AG-VNS results for medium1, medium3, medium5 and large instance are better than the results obtained by other approaches. These demonstrated that AG-VNS outperformed other approaches in the literature. These results demonstrated the effectiveness of applying the adaptive learning mechanism in VNS.

CONCLUSION

This study have proposed the AG-VNS algorithm for solving course time tabling problems. The basic idea of the AG-VNS is to guide the VNS algorithm to choose the right neighborhood structure during the search. Therefore, the adaptive learning mechanism was introduced to memorize which neighborhood structure can effectively solve specific soft constraint violations. The recommended neighborhood structure (from the learning phase) is used to enhance the quality of solutions in the improvement phase. Results have demonstrated that the AG-VNS produces better solutions for course time tabling problems and outperforms other stated approaches. This finding shows the effectiveness of the adaptive learning mechanism in the VNS algorithm in order to guide the VNS algorithm to choose the right neighborhood structure at the right time.

For future work, the AG-VNS will be hybridized with Honey Bee Mating Optimization (HBMO) to investigate the effectiveness of the AG-VNS in HBMO.

ACKNOWLEDGMENT

The authors wish to thank Ministry of Higher Education for supporting this work under the FRGS Research Grant Scheme (FRGS/1/2012/SG05/UKM/02/11).

REFERENCES

Abdullah, S., 2006. Heuristic approaches for university time tabling problems. Ph.D. Thesis, School of Computer Science and Information Technology, The University of Nottingham, United Kingdom.

Abdullah, S. and H. Turabieh, 2008. Generating university course timetable using genetic algorithms and local search. Proceedings of the 3rd International Conference on Convergence and Hybrid Information Technology, Volume 1, November 11-13, 2008, Busan, pp: 254-260.

Hansen, P. and N. Mladenovic, 2001. Variable neighborhood search: Principles and applications. Eur. J. Oper. Res., 130: 449-467.

Jat, S.N. and S. Yang, 2009. A guided search genetic algorithm for the university course time tabling problem. Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications, August 10-12, 2009, Dublin, Ireland, pp: 180-191.

Socha, K., M. Samples and M. Manfrin, 2003. Ant Algorithms for the University Course Time tabling Problem with Regard to the State-of-the-Art. In: Applications of Evolutionary Computing, Cagnoni, S., C.G. Johnson, J.J.R. Cardalda, E. Marchiori and D.W. Corne *et al*. (Eds.). Springer, Berlin, Germany, ISBN-13: 9783540009764, pp: 334-345.

Thompson, J. and K. Dowsland, 1995. General cooling schedules for a simulated annealing based time tabling system. Proceedings of the 1st International Conference on Practice and Theory of Automated Time tabling, August 29-September 1, 1995, Edinburgh, UK., pp: 345-363.