XLVII Symposium on Operational Research
SYM-OP-IS 2020
Belgrade, 20th-23rd September 2020.
UNIVERSITY OF BELGRADE
FACULTY OF TRANSPORT AND TRAFFIC ENGINEERING

MATHEURISTICS BASED ON VARIABLE NEIGHBORHOOD SEARCH

TATJANA DAVIDOVIĆ[1], LUKA MATIJEVIĆ[1]

[1] Mathematical Institute of the Serbian Academy of Science and Arts, Kneza Mihaila 36, 11000 Belgrade, Serbia, {tanjad,luka}@mi.sanu.ac.rs

Abstract: *Matheuristics represent heuristic optimization methods based on hybridizing exact solvers with metaheuristics. Usually, the exact solvers work on Mathematical Programming (more precisely Mixed Integer Linear Programming, MILP) formulation of the considered problem. Metaheuristic principles are used to define subproblems (e.g., by fixing values for a subset of variables) and exact solver is then invoked to determine values for the remaining variables. The main goal of this paper is to promote three matheuristics that explore Variable Neighborhood Search (VNS) as metaheuristic part. These are Variable Neighborhood Branching (VNB), Variable Neighborhood Decomposition Search for 0-1 MIP problems (VNDS-MIP), and Variable Intensity Neighborhood Search (VINS).*

Keywords: *Optimization problems, Mixed integer programs, Exact solvers, Metaheuristics, Hybrid methods*

1. INTRODUCTION

Matheuristics [7] become very popular in the last decade. Workshops devoted to the development and application of these general-purpose model-based optimization techniques are organized biannually since 2008 (`https://mh2018.sciencesconf.org/`). The generality of these methods means that matheuristics do not use *a priori* knowledge about the problem being optimized and that enables their application to the various range of problems. Matheuristics are actually heuristic methods obtained by hybridizing metaheuristics and exact solvers. The main idea behind this hybridization is to use metaheuristic rules for fixing values of some binary variables and creating subproblems for exact solvers. Sometimes, they can even be considered as the exact method, i.e., given enough resources (unlimited memory and running time) they can provide optimal solution.

In the recent literature, neighborhood based metaheuristics are usually combined with mathematical programming optimization techniques to design effective matheuristics. We review three state-of-the-art matheuristics based on Variable Neighborhood Search (VNS) [1, 3, 10]: Variable Neighborhood Branching (VNB) [4], Variable Neighborhood Decomposition Search for 0-1 MIP problems (VNDS-MIP) [6], and Variable Intensity Neighborhood Search (VINS) [5]. In order to describe these algorithms, we first introduce some notation and definitions.

Matheuristics rely on Mixed Integer Linear Programming, MILP formulation and usually operate exclusively on binary variables as they can take only two possible values, zero and one. On the other hand, it is straightforward to reformulate a model containing integer variables into a model with binary variables having an additional index.

In order to introduce neighborhood structures in the space of binary variables, we need to define the distance between two solutions $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_n)$. Hamming distance is obvious choice, i.e.,

$$\delta(x, x') = \sum_i | x_i - x'_i |, \tag{1}$$

where i refers to the subset of binary variables. If the original MIP is denoted by P, let $LP(P)$ represent its linear relaxation (obtained when in P the integer requirements on variables are released. The solution of $LP(P)$ is usually denoted by y. The distance (1) could be generalized to $\delta(x, y)$, where $| x_i - y_i |$ can take any value from the interval $[0, 1]$. $(P|C)$ is used to mark the subproblem of P obtained by adding the set of constraints C, while $P(k, x) = (P \mid \delta(x, x') \leq k)$, for $k = 1, 2, \ldots$ denotes the solution subspace of P containing all the solutions whose distance from a given solution x is at most k.$P(k, x)$ could be used to define neighborhoods:

$$\mathcal{N}_k(x) = \{x' \in X \mid \delta(x, x') \leq k\}. \tag{2}$$

More precisely, neighborhood $\mathcal{N}_k(x)$ contains all the solutions differing from x in the values of at most k binary variables. It is easy to see that $\mathcal{N}_k(x) \subset \mathcal{N}_{k+1}(x)$, and therefore, if neighborhood $\mathcal{N}_{k+1}(x)$ is completely explored, it is not necessary to search in neighborhood $\mathcal{N}_k(x)$.

An alternative way to define neighborhoods is using variable states. Each binary variable can be in one of the two possible states: fixed and relaxed. The fixed state means that the exact MIP solver is not allowed to change

its value during the optimization process. On the other hand, we want the optimizer to find the best value of the variables in relaxed state. Neighborhood structures are then defined with respect to the indices and number of relaxed variables. Local search in the space of binary variables is realized by invoking exact solver on a subproblem $(P|C)$ obtained by adding the set of neighborhood defining constraints C to the original problem P. The exact solver is allowed to run for a given time limit to avoid long executions in the cases when the defined neighborhoods are too large. The decision about the next step is then made based on the return status of the exact solver and is specific for each particular matheuristic.

All matheuristic methods described in this paper (VNB, VNDS-MIP, VINS) apply VNS rules for creating initial solutions and subproblems and CPLEX exact solver as a Local Search procedure to improve the current best solution. The paper is organized as follows. After the introduction, three sections to follow are devoted to the description of selected matheuristics. The paper concludes with Section 5.

2. VARIABLE NEIGHBORHOOD BRANCHING

Variable neighborhood branching (VNB) is proposed in [4]. The input parameters for VNB are 0-1 MIP problem P, VNS parameters k_{min}, k_{step}, k_{max}, maximum allowed CPU time T_{max}, and the time for subproblems t_{lim}. The pseudo-code of VNB is illustrated by Alg. 1. VNB adds constraints to the original problem P and generates subproblems explored in both Local Search (LS) and Shaking (SH) phase. The initial solution is obtained as the first feasible solution found by CPLEX. It becomes also the current best solution. Instead of standard LS, VNB performs search through multiple neighborhoods, i.e., Variable Neighborhood Descent (VND) [3]. Neighborhoods are defined by adding constraints (2) based on Hamming distance (1) to the original model. The value for k is updated properly with respect to the CPLEX return status. Actually, VNB uses two indices to count neighborhoods, k and k_1. Index k counts neighborhoods in VND, while in Shaking k_1 is used.

The diversification in VNB (Shaking) is performed by invoking CPLEX to find the first feasible solution within the disk defined by radii k_1 and $k_1 + k_{step}$. If it is necessary, the disk size is increased until a feasible solution is found. VND is realized by executing CPLEX on the defined subproblems until the pre-specified time limit t_{lim} is reached. Subproblems are controlled by the value for the current neighborhood index k. At the beginning of VND, $k = 1$ is set. If CPLEX proves that no feasible solution exists in the current neighborhood k, the value for k is increased by 1. In the case when in $\mathcal{N}_k(x)$ CPLEX finds an optimal solution or better feasible solution, an improvement is recorded and k is reset to 1. Finally, if no feasible solution is found within a t_{lim} time limit, VND procedure is terminated and the control is given to VNB. Move or Not step is realized in the standard way.

It should be noted here that the maximum number of neighborhoods in VND is not specified. Of course, it is pointless for k or k_1 to take values larger than the number of binary variables in P. In such a case, VNB acts as an exact algorithm as CPLEX is executed on the whole problem P. Another remark related to Alg. 1 is that Shaking and Local Search (VND) steps are reversed. This is due to the fact that the first feasible solution is considered as a result of Shaking in the whole search space.

3. VARIABLE NEIGHBORHOOD DECOMPOSITION SEARCH

Variable Neighborhood Decomposition Search (VNDS) was proposed in [2] to deal with hard optimization problems. It is a two-level optimization scheme based on VNS and on decomposition of the original problem into subproblems. Once the subproblems are solved, the solution of the whole problem is obtained by combining corresponding parts. VNDS for 0-1 MIPs appeared in [6]. It applies VNDS scheme on the set of binary variables to decompose the original problem. At each step, the original problem P is decomposed into two parts: in the first part, the variable values are fixed and this part should remain unchanged; the second part consists of relaxed variables whose values are to be determined by the exact MIP solver within the given time limit.

The first step of VNDS-MIP is to find an integer feasible solution x of the considered optimization problem P and an optimal solution y of $LP(P)$. The current best solution x_{best} is initialized by x. Using generalized definition for the distance $\delta(x,y)$, binary variables are sorted in a non-decreasing order with respect to $|x_i - y_i|$. More precisely, if $p = |\mathcal{B}|$ ($\mathcal{B}$ being the set of binary variables) and $\delta_j = |x_j - y_j|$, $j \in \mathcal{B}$, then x_j, $j = 1,2,\ldots,p$ are ordered such that $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_p$. Subproblems in VNDS-MIP are generated by changing states of binary variables, i.e., by fixing values for a subset of ordered binary variables to their values in the current best solution x_{best}. The variables to be fixed are the ones with values closest to their linear relaxation counterparts (see Fig. 1). The remaining variables are considered relaxed, and their number defines the size of the subproblem (neighborhood) to be solved by CPLEX within the given CPU time limit. In Fig. 1, variables on the left side (including shaded part) are fixed, while the remaining variables (towards the right side of the figure) are relaxed.

Algorithm 1 Pseudo-code for VNB

procedure VNB($P, T_{max}, t_{lim}, k_{min}, k_{max}, k_{step}$)
  $IntSolLim \leftarrow 1; TL \leftarrow T_{max}$
  $status \leftarrow MIPSolve(P, TL, IntSolLim, x)$
  if ($status = no\_feasible$) **then**
    Print("The problem cannot be solved within the given time limit.")
    Exit
  else
    $x_{best} \leftarrow x; f_{best} \leftarrow f(x); x' \leftarrow x; f' \leftarrow f(x)$
  end if
  $t \leftarrow 0$
  while ($t < T_{max}$) **do** ▷ Main VNB loop
    $cont \leftarrow True; k \leftarrow 1; k_1 \leftarrow k_{min}; IntSolLim \leftarrow MaxSol$
    while ($cont \wedge t < T_{max}$) **do** ▷ VND
      $TL \leftarrow \min\{t_{lim}, T_{max} - t\}$
      $AddConstraint(d(x', x) \leq k)$
      $status \leftarrow MIPSolve(P|C, TL, IntSolLim, x'')$
      $RemoveConstraint(d(x', x) \leq k)$
      switch ($status$) **do**
        case $no\_feasible$:
          $cont \leftarrow False$
        case opt_sol :
          $AddConstraint(d(x', x) \geq k + 1)$
          $x' \leftarrow x''; f' \leftarrow f(x''); k \leftarrow 1$
        case $feasible\_sol$:
          $AddConstraint(d(x', x) \geq 1)$
          $x' \leftarrow x''; f' \leftarrow f(x''); k \leftarrow 1$
        case $infeasible$:
          $k \leftarrow k + 1$
      end switch
      $t \leftarrow GetUserTime()$
    end while
    if ($f' < f_{best}$) **then** ▷ // Move or Not
      $x_{best} \leftarrow x'; f_{best} \leftarrow f(x'); k_1 \leftarrow 1$
    else
      $k_1 \leftarrow k_1 + k_{step}$; **if** $(k_1 > k_{max})$ **then** $k_1 \leftarrow k_{min}$
    end if
    $cont \leftarrow True; k_1 \leftarrow k_{min}; IntSolLim \leftarrow 1$
    while ($cont \wedge t < T_{max}$) **do** ▷ // Shaking
      $TL \leftarrow T_{max} - t$
      $AddConstraint(k_1 \leq d(x_{best}, x) \leq k_1 + k_{step})$
      $status \leftarrow MIPSolve(P|C, TL, IntSolLim, x')$
      if ($status = infeasible \vee status = no\_feasible$) **then**
        $k_1 = k_1 + k_{step}$
      else
        $cont \leftarrow False$
      end if
      $RemoveConstraint(k_1 \leq d(x_{best}, x) \leq k_1 + k_{step})$
      $t \leftarrow GetUserTime()$
    end while
  end while
  return (x_{best}, f_{best})
end procedure

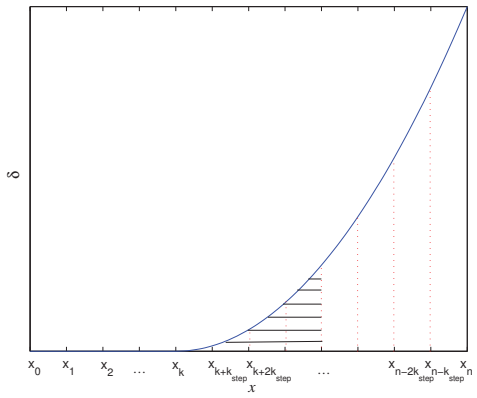

Figure 1 Illustration of subproblem generation in VNDS-MIP (Figure is provided by Dr. Jasmina Lazić)

After the binary variables are sorted, $q = |\{j \in \mathcal{B}, s.t.\ \delta_j = 0\}|$ is determined and it is used to calculate values for k_{min}, k_{max}, and k_{step} as it is shown in Alg. 2. Then the appropriate subsets of variables are fixed/released and time-limited CPLEX is invoked to solve the obtained subproblem. If a new best solution is obtained, an attempt of further improvement is made by applying VND (the same one used in VNB) to the x_{best}, otherwise, the subproblem is increased by properly changing neighborhood size for searching. In the case when the subproblem involves all variables with positive distance from the linear relaxation counterparts ($k + k_{step} > k_{max}$), the search is extended (half by half) to the remaining binary variables. If all binary variables were already relaxed during the search (which actually means that the whole problem was considered as a subproblem and treated by CPLEX), the execution of VNDS-MIP ends (flag execute is set to False). This means that either CPLEX was able to solve the original problem P (optimal solution has been found) or the maximum time limit was reached and the best feasible solution is reported to user. As VNDS-MIP is able to prove the optimality, given enough time and memory, we can consider it as an exact solution method.

4. VARIABLE INTENSITY NEIGHBORHOOD SEARCH

VINS [5] is inspired by the VNDS-MIP matheuristic [6] and Variable Intensity Local Search (VILS), a hybrid heuristic developed for multi-resource generalized assignment problem [9]. It generalizes the idea of fixing and relaxing subsets of binary variables in such a way that, contrary to VNDS-MIP which uses a single pattern, VINS exploits different patterns as it is the case in VILS.

As in VNDS-MIP, VINS solves the linear relaxation of the MIP first, and than finds the first feasible solution. The corresponding solutions are marked by y and x, respectively. Having these two sets of variable values, the distances between x and y based on the generalized form of Eq. (1) are calculated and the variables are sorted in the non-decreasing order according to these distance values. In VNDS-MIP, variables having similar values in x and y were considered to offer less space for improvement, however, this may not be true. Therefore, 10 different patterns (referred to as neighborhoods) for generating subproblems are defined in VINS as follows.

- N1: α% of the worst variables are released;
- N2: variable set is divided into 10 bins and $\alpha/10$% worst variables are released in each bin;
- N3: starting at random position α% variables are released;
- N4: at 10 random positions, $\alpha/10$% variables are released;
- N5: α% of the best variables are released;
- N6: in 10 equal bins $\alpha/10$% best variables are released;
- N7: $\alpha/2$% of best and $\alpha/2$% of worst variables are released;
- N8: in 10 equal bins the same pattern as in N7 is applied;
- N9: random α% variables are released;
- N10: in 10 equal bins, random $\alpha/10$% variables are released.

Terms "the best" and "the worst" are used here to mark variables with the lowest and the biggest distance between the values in current best solution and in the linear relaxation solution, respectively. The value of parameter α defines the neighborhood size, i.e., the percentage of variables that will be released and given to

Algorithm 2 Pseudo-code for VNDS-MIP

```
procedure VNDS-MIP(P, T_max, t_lim)
    IntSolLim ← 1; TL ← T_max
    lpstat ← LPSolve(P, TL, y)
    status ← MIPSolve(P, TL, IntSolLim, x)
    if (status = no_feasible) then
        Print("The problem cannot be solved within the given time limit.")
        Exit
    else
        x_best ← x; f_best ← f(x); x' ← x; f' ← f(x); p ← | B |
    end if
    t ← 0; execute ← True
    while (execute ∧ t < T_max) do                                   ▷ Main VNDS-MIP loop
        Calculate(δ); x ← Sort(x_best, y); q ← | {j ∈ B, s.t. δ_j = 0} |
        k_max ← p − q; k_min ← k_max/10; k_step ← k_min; k ← k_min
        cont ← True; IntSolLim ← MaxSol
        while (cont ∧ t < T_max) do
            TL ← min{t_lim, T_max − t}; improved ← True              ▷ Local Search
            FixAndRelease(P, k)
            AddConstraint(f(x), L, f_best)
            status ← MIPSolve(P|C, TL, IntSolLim, x'')
            ReleaseAll(P|C)
            if (status = no_feasible ∨ status = infeasible) then
                improved ← False
            end if
            if (improved) then                                       ▷ Move or Not
                VND(P, T_max, t_lim, k, x'', x', f')
                x_best ← x'; f_best ← f(x'); k ← k_min; cont ← False
            else
                if (k + k_step > k_max) then k_step ← max{[k/2], 1}
                if (k ≤ p) then
                    k ← k + k_step
                else
                    execute ← False
                end if
            end if
            t ← GetUserTime()
        end while
    end while
    return (x_best, f_best)
end procedure
```

time-limited CPLEX solver for improving. In addition to changing neighborhood types, VINS allows changes in the size of neighborhoods. As the search advances, the size of neighborhoods α increases, as well as the corresponding time t_{lim} to limit CPLEX engagement in the resulting subproblems. Therefore, with respect to VNDS-MIP, VINS has two new input parameters: an array of neighborhood sizes *alphas* and the corresponding array of times for subproblems *time_limits*.

The proposed VINS does not explore the "first improvement" VNS strategy that is adopted in VNB and VNDS-MIP. Instead of moving to N1 of the smallest size upon each improvement of the current best solution, the neighborhoods are searched in round robin fashion, starting from N1 till N10, in each of the available sizes. In the case an improvement is made, the current best solution is updated, the corresponding constraint limiting the objective function value is added to the model, the variables are re-sorted by the distance between values in current best solution and the linear relaxation solution, and the search continues in the next neighborhood of the same size. The size never decreases as it is the case in other two methods. When all neighborhoods are explored, the size is increased and the search continues in N1. If there is enough time and all sizes are explored, CPLEX is invoked to additionally improve the current best solution acting on the whole problem *P*. The proposed procedure is illustrated by Alg. 3.

The Linux versions of all methods are available at the Mathematical Institute of the Serbian Academy of Sciences and Arts and can be provided by the authors upon a request. For all of them, data about the problem to be solved should be provided in a form of .lp file which can be generated automatically in most of the exact solvers (CPLEX, Gurobi, etc.). The three described methods are compared for the first time in [5] on the miplib3 instances and on the container ships routing problem from [8]. Recently, a new comparison on allocating of passenger ferry fleet in maritime transport problem is reported in [11].

5. CONCLUSION

In order to promote three matheuristic methods based on Variable Neighborhood Search principles and developed by Serbian researches we presented their brief description. Successful applications confirming their usefulness can be find in numerous papers from the relevant literature and we mentioned the most recent few.

Algorithm 3 VINS

procedure VINS($P, T_{max}, alphas, time\_limits$)
  $IntSolLim \leftarrow 1; TL \leftarrow T_{max}$
  $lpstat \leftarrow LPSolve(P, TL, y)$
  $status \leftarrow MIPSolve(P, TL, IntSolLim, x)$
  if $(status = no\_feasible)$ **then**
    Print("The problem cannot be solved within the given time limit.")
    Exit
  else
    $x_{best} \leftarrow x; f_{best} \leftarrow f(x_{best})$
  end if
  $t \leftarrow 0; cont \leftarrow True$
  $N \leftarrow N1$
  $\alpha \leftarrow Alpha1$
  $TL \leftarrow TimeLimit1$
  $IntSolLim \leftarrow \infty$
  $x \leftarrow Sort(x_{best}, y)$
  while $(cont \wedge t < T_{max})$ **do** ▷ Main VINS loop
    $TL \leftarrow \min\{t_{lim}, T_{max} - t\}$
    $FixAndRelease(N, \alpha)$
    $status \leftarrow MIPSolve(P|C, TL, IntSolLim, x)$
    $improvement \leftarrow Improved(x)$
    if $(improvement)$ **then** ▷ Checking for Improvement
      $x_{best} \leftarrow x; f_{best} \leftarrow f(x)$
      AddConstraint$(f(x), L, f_{best})$
      $x \leftarrow Sort(x_{best}, y)$
    end if
    $N \leftarrow NextNeighborhood$ ▷ Move Step
    $t \leftarrow GetUserTime()$
    if $(N = null)$ **then**
      $N \leftarrow N1$
      $\alpha \leftarrow NextAlpha$
      $t_{lim} \leftarrow NextTimeLimit$
      if $(\alpha = null)$ **then** ▷ Final improvement of the whole problem
        $TL \leftarrow \min\{t_{lim}, T_{max} - t\}$
        $status \leftarrow MIPSolve(P, TL, IntSolLim, x)$
        $cont \leftarrow False$
      end if
    end if
  end while
  return (x_{best}, f_{best})
end procedure

Acknowledgement

This work has been supported by Serbian Ministry of Education, Science and Technological Development through the Mathematical Institute of Serbian Academy of Sciences and Arts.

REFERENCES

[**1**] Hansen, P., Mladenović, N., Brimberg, J., Moreno Pérez, J.A. (2019). Variable neighborhood search. In: Handbook of metaheuristics (57–97). Cham: Springer.

[**2**] Hansen, P., Mladenović, N., Perez-Britos, D. (2001). Variable Neighborhood Decomposition Search. Journal of Heuristics, 7(4), 335–350.

[**3**] Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S. (2017). Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization, 5(3), 423–454.

[**4**] Hansen, P., Mladenović, N., Urošević, D. (2006). Variable neighbourhood search and local branching. Comput. Oper. Res. 33(10), 3034–3045.

[**5**] Jovanović, P., Davidović, T., Lazić, J., Mitrović Minić, S. (2015). The variable intensity neighborhood search for 0-1 MIP. In: Proc. 42nd Symposium on Operations Research, SYM-OP-IS 2015, (229–232). Srebrno jezero, Serbia.

[**6**] Lazić, J., Hanafi, S., Mladenović, N., Urošević, D. (2010). Variable neighbourhood decomposition search for 0–1 mixed integer programs. Comput. Oper. Res. 37(6), 1055–1067.

[**7**] Maniezzo, V., Stèutzle, T., Voss, S. (eds.) (2009). Matheuristics: hybridizing metaheuristics and mathematical programming, vol. 10. New York: Springer.

[**8**] Maraš, V., Lazić, J., Davidović, T., Mladenović, N. (2013). Routing of barge container ships using MIP heuristics. Applied Soft Computing, 13(8), 3515–3528.

[**9**] Mitrović-Minić, S., Punnen, A.P. (2009). Local search intensified: Very large-scale variable neighborhood search for the multi-resource generalized assignment problem. Discrete Optimization, 6(4), 370–377.

[**10**] Mladenović, N., Hansen, P. (1997). Variable neighborhood search. Comput. & OR, 24(11), 1097–1100.

[**11**] Škurić, M., Maraš, V., Davidović, T., Radonjić, A. (2020). Optimal allocating and sizing of passenger ferry fleet in maritime transport. Research in Transportation Economics, (accepted).