Efficient heuristics for the workover rig routing problem with a heterogeneous fleet and a finite horizon

Journal of Heuristics manuscript No.
(will be inserted by the editor)

Efficient heuristics for the workover rig routing problem with a heterogeneous fleet and a finite horizon

Glaydston Mattos Ribeiro · Guy Desaulniers · Jacques Desrosiers · Thibaut Vidal · Bruno Salezze Vieira

Received: date / Accepted: date

Abstract Onshore oil fields may contain hundreds of wells that use sophisticated and complex equipments. These equipments need regular maintenance to keep the wells at maximum productivity. When the productivity of a well decreases, a specially-equipped vehicle called a workover rig must visit this well to restore its full productivity. Given a heterogeneous fleet of workover rigs and a set of wells requiring maintenance, the workover rig routing problem (WRRP) consists of finding rig routes that minimize the total production loss of the wells over a finite horizon. The wells have different loss rates, need different services, and may not be serviced within the horizon. On the other hand, the number of available workover rigs is limited, they have different initial positions, and they do not have the same equipments. This paper presents and compares four heuristics for the WRRP: an existing variable neighborhood

Glaydston Mattos Ribeiro
Transportation Engineering Program, Federal University of Rio de Janeiro, Brazil
Tel.: +55-21-25628132
Fax: +55-27-25628131
E-mail: glaydston@pet.coppe.ufrj.br

Guy Desaulniers
Department of Mathematics and Industrial Engineering and GERAD, École Polytechnique de Montréal, Canada
E-mail: guy.desaulniers@gerad.ca

Jacques Desrosiers
Department of Management Sciences and GERAD, HEC Montréal, Canada
E-mail: jacques.desrosiers@hec.ca

Thibaut Vidal
Massachusetts Institute of Technology and CIRRELT, MIT, USA
E-mail: thibaut.vidal@cirrelt.ca

Bruno Salezze Vieira
Department of Computer Engineering and Electronics, Federal University of Espírito Santo, Brazil
E-mail: brunosalezze@gmail.com

search heuristic, a branch-price-and-cut heuristic, an adaptive large neighborhood search heuristic, and a hybrid genetic algorithm. These heuristics are tested on practical-sized instances involving up to 300 wells, 10 rigs on a 350-period horizon. Our computational results indicate that the hybrid genetic algorithm outperforms the other heuristics on average and in most cases.

Keywords Workover rig routing · Branch-price-and-cut heuristic · Adaptive large neighborhood search · Hybrid genetic algorithm · Vehicle routing

1 Introduction

The workover rig routing problem (WRRP) arises on onshore oil fields which can contain hundreds of oil wells. These wells use artificial lift methods to make oil surface and this operation uses complex equipments. When such equipments present some malfunction, the productivity of the well is reduced and a request for maintenance describing the type of maintenance required and a production loss rate (e.g., $20m^3/day$) is issued. The maintenance is performed by a specially-equipped vehicle called a workover rig (in short, a rig) which is a compact mobile unit carrying external equipment for mud preparation and power generation. To avoid high production losses, maintenance should be completed as soon as possible.

Typically, in large oil fields such as those encountered in Brazil, many wells require maintenance at the same time and oil companies possess more than one rig to service the wells as soon as possible. However, due to the high costs of acquiring and operating rigs, they own relatively few of them compared with the number of wells to maintain. Consequently, an immediate allocation of a rig to a maintenance request is not possible, resulting in a queue of wells awaiting a rig for service. Moreover, the rigs are not necessarily equipped similarly and, depending on the maintenance type to perform, only a subset of them may be suitable to service a request. In this case, the fleet of rigs is said to be heterogeneous.

Given that the wells are located in various places in the oil field, the rigs must often travel between two maintenance services. Given their slow speed (approximately $12mph$), travel times must be taken into account when routing the rigs and scheduling the services. We assume that all rigs travel at the same speed but the algorithms that we propose in this paper can easily be adapted for the case with varying speeds.

Maintenance services also require time that must be considered when scheduling the rigs. The service time at a well depends on the maintenance type (e.g., completion, reinstatement, cleaning and stimulation), not on the rig performing the service. In Brazil, the completion service is considered strategic and takes about 15 days on average. The reinstatement service is also time-consuming and lasts between 5 to 15 days. The cleaning and stimulation services are much shorter and can be performed in less than a day or two.

The maintenance requests are issued dynamically but they are not directly assigned to a rig. The planning of the rig routes (sequences of wells to service) is performed in a rolling horizon fashion, that is, at every P time periods, where a period may correspond to, e.g., 2, 6, 12 or 24 hours. For a planning session, a finite horizon of H time periods with $H \geq P$ is considered for servicing the requests that are known at this time, including the portions of the routes previously planned that will not be completed before the beginning of the horizon. Given the limited number of rigs available, it might not be possible to plan the service of all the requests within the horizon. The unserved ones are postponed to the next planning session together with the requests to be issued in the next P periods, that is, until the next planning session. Because of this rolling horizon process, a rig can be located anywhere in the field at the beginning of the planning horizon, i.e., where it performed its last service or at a location en-route to another well. Also, it might not be available at the beginning of the horizon if it requires time to complete a maintenance that started before the horizon.

One particularity that distinguishes the WRRP from many other vehicle routing problems is that it involves no depot. To increase their productivity, the rigs are replenished in diesel in the field and their crews are also relieved where they are located. In consequence, the rig routes can finish anywhere at the end of the planning horizon.

In this context, the WRRP can be defined as follows. Given a set of wells requesting maintenance and a heterogeneous fleet of rigs, the WRRP aims at determining feasible routes for the rigs that minimize the sum of the production losses at the wells over the next H time periods. A rig route is deemed feasible if it starts at the rig's initial position, it lasts at most H periods, and it services only wells for which the rig is equipped. Multiple types of well services and rigs with well-to-rig compatibility constraints are thus considered. A route can end anywhere (typically, at the last visited well). Since the number of rigs is usually insufficient to service all wells in the planning horizon, we can assume that all rigs are used. Note also that the WRRP does not involve routing costs as they are considered negligible compared to the high costs incurred by the production losses.

The WRRP includes the traveling repairman problem (Tsitsiklis [26]) as special case in the presence of a single rig and with a large planning horizon. The traveling repairman problem is known to be NP-Hard from Sahni and Gonzalez [22], and so is the WRRP. Literature on the WRRP with a heterogeneous fleet is scarce. Aloise et al. [1] developed a variable neighborhood search (VNS) heuristic that produced solutions which could potentially reduce production losses worthing an estimated US \$2,568,000 per year. Their algorithm assumes, however, an infinite time horizon and assigns all requests to rigs using a greedy heuristic that is explained in Section 3. To evaluate a solution, the authors consider only the wells serviced within the horizon. Two exact algorithms based on column generation were also proposed recently by Duhamel et al. [8] and Ribeiro et al. [18]. Duhamel et al. [8] introduced three mixed-integer linear models. The first model improves the schedule-based formulation of Aloise et al. [1], the second one is based on an open vehicle routing model, and the last one is a set covering model for which a column generation algorithm is proposed. The best results were obtained with the latter model that allowed to solve to optimality instances with up to 4 rigs and 60 wells. Ribeiro et al. [18] devised a branch-price-and-cut (BPC) algorithm that relies on some of the most recent techniques proposed for the vehicle routing problem with time windows. The authors succeeded to solve practical-sized instances (with up to 10 rigs and 200 wells) in reasonable computational times (less than one hour).

Various heuristics were also developed for variants of the WRRP. A tabu search metaheuristic and an iterated local search heuristic were proposed by Neves [15] for the homogeneous fleet WRRP with mandatory service at all wells and deadlines. This variant was also studied by Ribeiro et al. [19] who designed a clustering search heuristic and an adaptive large neighborhood search heuristic, outperforming the results obtained by Neves [15].

For the homogeneous fleet WRRP with mandatory service at all wells and time windows but without travel times (only service times are considered), four heuristics were developed: a greedy randomized adaptive search procedure by Costa [4], a dynamic mounting heuristic by Costa and Ferreira Filho [5], a greedy randomized adaptive search procedure with path relinking by Pacheco et al. [16], and a simulating annealing metaheuristic by Ribeiro et al. [20].

Even if the BPC algorithm of Ribeiro et al. [18] can solve real-sized instances, it fails to produce good-quality feasible solutions within one hour of computational time for certain tested instances. Our goal is, thus, to identify a heuristic algorithm that can yield high-quality solutions in reasonable computational times for all instances. To do so, we first introduce three new heuristics for the WRRP that are based on state-of-the-art methodologies: a BPC heuristic, an adaptive large neighborhood search (ALNS) heuristic, and a hybrid genetic algorithm (HGA). Then, we compare the results obtained by these heuristics and the VNS heuristic of Aloise et al. [1] on the benchmark WRRP instances proposed by Ribeiro et al. [18].

The remainder of this paper is structured as follows. Section 2 presents a mathematical model for the WRRP. Section 3 summarizes the VNS heuristic of Aloise et al. [1] while Section 4 describes the proposed BPC, ALNS and HGA heuristics. Computational results are reported and discussed in Section 5, which also includes a brief conclusion.

2 Mathematical model

Ribeiro et al. [18] proposed two mathematical models for the WRRP, namely, an arc-flow model and a set packing model that can be obtained from the first model by applying the Dantzig-Wolfe decomposition principle. In this section, we present the latter model on which the proposed BPC heuristic relies.

Consider a planning horizon represented by a finite set of disjoint time periods of equal length (e.g., one hour) numbered from 1 to H. Let W be the set of wells requiring maintenance, ℓ_i the production loss rate per period at well $i \in W$, e.g., in $m^3/hour$, and τ_i its service duration.

Let K be the set of available rigs. Each rig $k \in K$ is associated with an equipment level $e_k \in \mathbb{N}$ and the service at well $i \in W$ requires an equipment with minimum level $e'_i \in \mathbb{N}$. A well-to-rig allocation is feasible if $e_k \geq e'_i$. We denote by $W^k \subseteq W$ the subset of wells that can be maintained by rig k. Let R^k be the set of feasible routes for rig k. With each route $r \in R^k$ and each well $i \in W^k$, we associate a binary parameter a^k_{ir} equal to 1 if route r visits well i and to 0 otherwise. With each route $r \in R^k$, we also define a parameter v^k_r that indicates the sum of the loss saved at each well serviced along the route, that is,

$$v^k_r := \sum_{i \in W^k} \ell_i (H a^k_{ir} - t^k_{ir}),$$

where t^k_{ir} is equal to the time period at which the service completes at well i in route r if it services it and to 0 otherwise. Finally, for each rig $k \in K$ and each route $r \in R^k$, define a binary variable Y^k_r indicating whether or not route r for rig k is selected in the solution.

With this notation, the WRRP can be formulated as the following set packing model:

$$\textit{Minimize} \quad H \sum_{i \in W} \ell_i - \sum_{k \in K} \sum_{r \in R^k} v^k_r Y^k_r \tag{1}$$

$$\text{subject to:} \quad \sum_{k \in K} \sum_{r \in R^k} a^k_{ir} Y^k_r \leq 1, \ \ \forall i \in W \tag{2}$$

$$\sum_{r \in R^k} Y^k_r = 1, \ \ \forall k \in K \tag{3}$$

$$Y^k_r \in \{0,1\}, \ \ \forall k \in K, \ r \in R^k. \tag{4}$$

The objective function (1) seeks to minimize the total production loss that is computed as the difference between the maximal loss (if no wells are maintained) and the sum of the losses saved by the rigs. The set packing constraints (2) guarantee that each well is serviced at most once in the planning horizon. The convexity constraints (3) ensure that one route is assigned to each available rig. Finally, the variables are subject to binary requirements (4).

In practice, model (1)–(4) contains a very large number of variables, one per feasible route for each rig. This difficulty can be overcome using column generation as in the exact BPC algorithm of Ribeiro et al. [18] or in the BPC heuristic developed in Section 4.1.

3 Variable neighborhood search

In this section we briefly describe the VNS heuristic of Aloise et al. [1] that we implemented and tested in our computational experiments.

A VNS heuristic explores a sequence of neighborhoods of increasing size (typically, nested) to search for improving solutions (Mladenović and Hansen

[14], Hansen and Mladenović [9]). When such a solution is found, it returns to the smallest neighborhood before continuing. Besides the neighborhoods, a VNS heuristic involves: a procedure to generate an initial solution, a perturbation strategy associated with the current neighborhood, and a local search algorithm. It proceeds as follows. First, an initial solution s is constructed and the best solution s_{best} found so far corresponds to s. Then, the main loop starts. While a stopping condition criterion is not met (in our case, a maximum computational time), the current solution s is perturbed randomly using a move allowed in the current neighborhood to generate a new solution s'. Starting from s', local search is applied to yield a solution $\bar{s}$. If $\bar{s}$ is not better than s_{best}, a new iteration is started from s but using the next neighborhood. Otherwise, the algorithm updates s_{best}, replaces s by $\bar{s}$, and returns to the smallest neighborhood before starting a new iteration.

We now review the key-elements implemented by Aloise et al. [1] for the WRRP.

Initial solution. An initial solution is constructed using a greedy algorithm that appends one well at a time to the current rig routes. More precisely, the algorithm loops continuously over the routes and adds one well at each route until all wells are inserted into a route without taking into account the horizon length. Aloise et al. [1] suggest to select the well to insert as the well (not yet scheduled) that yields the maximum production loss if inserted at the end of the rig route. This criterion favors long travel times between two wells serviced consecutively by the same rig. Preliminary computational experiments showed that it is rather preferable to choose the well yielding the minimum production loss. Consequently, we performed our tests using the latter criterion.

Local search. The local search procedure relies on the well swap moves within the same route and between two routes. Each iteration stops at the first improving solution found.

Perturbation. The algorithm uses nine different neighborhoods based on well-known moves such as route swap between two rigs, well swap within the same route or between two routes, and well reassignment from one rig to another.

Objective function. The objective function is computed as the total production loss within the horizon, considering that all wells scheduled for a maintenance after the end of the horizon are not serviced.

4 BPC, ALNS and HGA heuristics for the WRRP

In this section we present the new heuristics that we propose for the WRRP, namely, a BPC, an ALNS, and an HGA heuristic.

4.1 BPC heuristic

The BPC heuristic is an adaptation of the exact BPC algorithm developed by Ribeiro et al. [18]. A BPC algorithm (Barnhart et al. [3], Lübbecke and Desrosiers [13], Desrosiers and Lübbecke [7]) is a branch-and-bound algorithm in which the lower bounds are computed by column generation and cutting planes are added to tighten the linear relaxations. Column generation is a well-known mathematical programming technique able to solve linear programs that involve a very large number of variables associated with combinatorial objects such as paths in one or several networks. This iterative technique solves at each iteration a restricted master problem and one or several subproblems. The restricted master problem corresponds to the linear relaxation of the original model restricted to a relatively small subset of its variables. The subproblems allow the identification of new variables (columns) to add to the restricted master problem (variables with negative reduced costs) or they prove that none exist. In the latter case, the algorithm stops with an optimal linear relaxation solution. When this solution is fractional and the corresponding node is not pruned, violated valid inequalities can be added to the model or branching decisions can be imposed.

In the BPC algorithm of Ribeiro et al. [18], the master problem is given by the linear relaxation of model (1)–(4). There is one subproblem per rig. It corresponds to an elementary shortest path problem with an additional constraint to limit route duration and a time-dependent objective function. Given that this type of subproblem is NP-hard, the authors suggest to solve a relaxation of it, namely, its *ng*-path relaxation (Baldacci et al. [2]) that allows certain cycles. The shortest *ng*-path subproblems are solved using a labeling algorithm. Because these subproblems may still be computationally extensive, two heuristics are used to generate columns more rapidly. Indeed, at each iteration, a tabu search column generator is invoked first. For each route corresponding to a basic variable in the current restricted master problem solution, tabu search is applied starting with this route (that has a zero reduced cost) as an initial solution. Only two moves are considered: well insertion and well removal. A very limited number of tabu search iterations (predefined as a parameter I^{tabu}_{max}) are performed for each initial route. When the tabu search heuristic fails to find negative reduced cost columns, a heuristic labeling algorithm is invoked. This heuristic is the same as the exact labeling algorithm but it is applied on reduced-sized networks. These are obtained by keeping the arcs with the current smallest reduced costs, that is, at most A^{min} incoming and outgoing arcs at each node, see Ribeiro et al. [18] for details. When the heuristic labeling algorithm fails, the exact labeling algorithm is called to guarantee the exactness of the overall solution process.

To strengthen the linear relaxation at the root node of the search tree, Ribeiro et al. [18] add the following subset-row inequalities (Jepsen et al. [10]):

$$\sum_{k \in K} \sum_{r \in R^k_Q} Y^k_r \leq 1, \ \ \forall Q \subseteq W \text{ such that } |Q| = 3, \tag{5}$$

where Q is a subset of three wells and $R^k_Q \subseteq R^k$ is the subset of routes servicing at least two wells in Q. These inequalities are very efficient at tightening the linear relaxation. On the other hand, they require modifying the structure of the subproblems to handle their dual variables and may yield highly time-consuming subproblems. Whenever needed, Ribeiro et al. [18] impose branching decisions on the total flow on an arc linking two wells.

From this exact BPC algorithm, we devise a BPC heuristic by modifying the column generation procedure as well as the branching scheme. In the column generation algorithm, most of the time is spent solving the subproblems, especially when the exact labeling algorithm is invoked. Furthermore, this exact algorithm is often applied only to prove the optimality of the current restricted master problem solution, that is, most of the times it does not succeed to generate negative reduced cost columns. Consequently, we propose to use only the tabu search heuristic and the heuristic labeling algorithm to generate columns. Given that these heuristics are quite fast, we use them to solve the elementary shortest path version of the subproblems (rather than the ng-path relaxation). This allows to yield better linear relaxation solutions. For our computational tests, we use the following parameter setting: $I^{tabu}_{max} = 100$ and $A^{min} = 5$.

Instead of exploring a complete search tree to derive an integer solution, we use a diving strategy that alternates between solving a linear relaxation and imposing permanent decisions until finding an integer linear relaxation solution. Linear relaxations are solved heuristically by column generation as discussed in the previous paragraph. When the solution of a linear relaxation is fractional, two types of decisions can be imposed. First, if some Y^k_r variables take values above a given threshold (0.7 for our tests), all these variables are fixed to 1. Second, when there are no such variables, we fix to 1 the total flow on an arc between two wells. This arc is selected as the one with the highest total fractional flow.

Our BPC heuristic also uses subset-row inequalities but more scarcely than in the exact BPC algorithm to avoid increasing too much the time to solve the subproblems by the labeling heuristic. In fact, violated subset-row cuts are sought only when no Y^k_r variables can be fixed. Furthermore, a cut is added only if its violation exceeds a minimum threshold (set to 0.3 for our tests).

4.2 Adaptive large neighborhood search

Ropke and Pisinger [21] proposed the ALNS heuristic, which is an extension of the large neighborhood search heuristic introduced by Shaw [23]. It uses the ruin and recreate principle. The algorithm starts with an initial solution s which is modified iteratively. At each ALNS iteration, the algorithm destroys part of the current solution s and repairs it in a different way to generate a new solution s'. This solution is accepted as the current solution according to a criterion defined by a search paradigm such as simulated annealing. The algorithm terminates when it satisfies a stopping criterion, which in our case is a maximum number of iterations (50,000 for our tests). Several destroy and repair procedures can be considered throughout the algorithm. The pair of destroy and repair procedures to use at each iteration is selected through an adaptive probabilistic mechanism: the probability of selecting a given procedure depends on how well it performed previously.

As mentioned in Section 1, Ribeiro et al. [19] developed an ALNS heuristic for a variant of the WRRP that involves a homogeneous fleet of rigs and requires servicing each well before a given deadline specific to the well. For this paper, we tried to use a slightly modified version of this heuristic, considering a common deadline (the end of the horizon) for all wells. Two modifications of the repair phase were implemented to consider a heterogeneous fleet and to allow unassigned wells. First, wells can only be inserted into the routes of the rigs that are equipped to service them. Second, when attempting to insert a well into a compatible route, the duration of the resulting route is verified to ensure that it can be executed within the planning horizon. If this is not the case, the insertion is rejected and, when no feasible insertion can be found, the assignment of the well is left for the next ALNS iteration. This approach provided poor results for the WRRP instances considered in this paper. Therefore, we propose a new version of this heuristic that involves new destroy and repair procedures as well as a local search post-optimization algorithm. Besides, a different strategy is also adopted to define the simulated annealing cooling rate in the solution acceptance criterion.

Initial solution. To construct an initial solution, we use the same procedure as for the VNS heuristic (see Section 3). However, a well can be assigned to a rig if and only if its maintenance can be completed within the time horizon.

Selection of destroy and repair procedures. At each iteration of the ALNS algorithm, a destroy procedure is chosen to remove q wells from the routes and a repair procedure is then applied to insert them back into the current routes. The number q of wells to remove is a random variable that follows a discrete uniform distribution on the interval $[0.1|W|, 0.4|W|]$. The destroy and repair procedures, called hereafter removal and insertion procedures, are described in Sections 4.2.1 and 4.2.2.

The choice of the removal and the insertion procedure is based on the success of all the procedures in the previous iterations. More precisely, let $w_p > 0$ be a measure of how well a procedure p has performed in the past iterations. Then, given P procedures with weights w_p, procedure p is selected with probability $w_p / \sum\limits_{j=1}^{P} w_j$.

The removal and insertion procedures are weighted independently as follows. The overall search is divided into disjoint segments of φ consecutive ALNS iterations (in our tests, $\varphi = 100$). Initially, all procedures have the same weight, say $w_p = 25$ for $p = 1, \ldots, P$. Then, after φ iterations, the weight w_p of each procedure p is updated considering a score π_p that indicates how well the procedure has performed in this last segment. In our case, π_p is first set to 0 at the beginning of each segment. Then, when a pair of removal-insertion procedures finds a new best solution in the current segment, their scores are increased by δ_1. When it finds a solution better than the current one, their scores are increased by δ_2. Finally, when it finds a non-improving solution that is accepted, their scores are increased by δ_3. For our tests, we used $\delta_1 = 3$, $\delta_2 = 2$, and $\delta_3 = 1$.

Let ζ_p be the number of times that procedure p was chosen in the last segment. At the end of the segment, the weight w_p of procedure p is updated as follows. If $\zeta_p = 0$, then w_p does not change. Otherwise, $w_p := (1-\eta)\, w_p + \eta\pi_p/\zeta_p$, where η is the reaction factor defined by Ropke and Pisinger [21] that controls how quickly the weights are adjusted according to the effectiveness of the procedures. For our tests, the value of η was set to $1 - (4 \times 10^{-5})^{\varphi/I_{max}^{alns}} \approx 0.02$, where $I_{max}^{alns} =$ 50,000 is the maximum number of ALNS iterations.

Acceptance criterion. The acceptance criterion is defined according to a simulated annealing rule. At the ALNS iteration j, let $c(s)$ and $c(s')$ be the costs of the current solution s and the generated solution s', respectively. If $c(s') < c(s)$, then s' is accepted as the new current solution. Otherwise, s' is accepted with a probability $exp((c(s) - c(s'))/T_j)$, where T_j is the temperature at iteration j. The initial temperature T_1 is set to one third of the cost of the initial solution. Then, it cools down using the formula $T_j = \rho T_{j-1}$, where $\rho = (T_{end}/T_1)^{1/I_{max}^{alns}}$ is the cooling rate. This rate has been chosen such that, after I_{max}^{alns} iterations, the system reaches a frozen state corresponding to a target final temperature T_{end} that is equal to one twentieth of the initial solution cost.

Local search. When the ALNS heuristic stops, a post-optimization local search algorithm is applied on the final solution in the hope of improving it. This local search heuristic involves the traditional operators used for vehicle routing problems such as RELOCATE, SWAP, 2-OPT and 2-OPT*. See Vidal et al. [28] for more details. The local search is not highly time-consuming and improves the solution quality in some cases.

4.2.1 Removal procedures

In this section, we describe the removal procedures used in our ALNS heuristic. They are based on the ones proposed by Ropke and Pisinger [21] and Ribeiro et al. [19], but adapted to the WRRP. Where applicable, let D be a set of removed wells and let L be a set of wells not yet removed from the current solution. Note that, when a well i is removed in a route between locations i^- and i^+, then a modified route is obtained by connecting i^- to i^+.

Shaw removal. The general idea of a Shaw removal procedure (Shaw [23]) is to remove wells that are somewhat similar. The degree of similarity between two wells i and j is computed through a relatedness measure $R(i, j)$, where a lower value corresponds to more similar wells. We use the following two Shaw removal procedures.

The first procedure takes into account the absolute difference between the production loss of the wells i and j in the current solution. The relatedness measure is given by $R(i,j) = \left|\ell_i t_i^{k_1} - \ell_j t_j^{k_2}\right|$, where $t_i^{k_1}$ and $t_j^{k_2}$ are the end of service time periods at wells i and j, respectively, with $k_1, k_2 \in K$. Given a solution s, a set of removed wells D and a set of not yet removed wells L, the algorithm randomly selects a well $w \in D$, calculates the relatedness measure between it and each well $j \in L$, and then sorts L in increasing order of the relatedness measure. A well from L is removed and this process is repeated until $|D| = q$.

The well to remove is selected in L according to a random procedure introduced by Ropke and Pisinger [21] that we call the random well selection (RWS) procedure. In the RWS procedure, a uniform random number u is drawn in $[0,1]$ and the i-th well in L is selected where $i = \lceil u^3 |L| \rceil$. This procedure is used in several removal procedures.

The second Shaw removal procedure is based on travel times instead of service completion times. The relatedness measure between i and j is given by $R(i,j) = t_{ij}$, where t_{ij} is the travel time from well i to well j. The rest of the procedure is identical to the previous one.

Random removal. This removal procedure simply removes q wells at random from the routes in the current solution s. As mentioned by Ribeiro et al. [19], this procedure tends to generate a poor set of removed wells, but it helps diversifying the search.

Worst removal. This procedure removes wells with high production losses in the current solution s. Let $ProdLoss^-(i,s) = c(s) - c^-(i,s)$ be the production loss caused by well i in current solution s, where $c^-(i,s)$ represents the solution cost without well i in s. This procedure first sorts the wells according to $ProdLoss^-(i,s)$, chooses one well to remove using the RWS procedure, recomputes $ProdLoss^-(i,s)$ for the remaining wells, and repeats the process until removing q wells.

Cluster removal. Given a rig route, this procedure splits it into two clusters of wells where each cluster is defined by a criterion based on a connected component of the underlying network. We applied the approach proposed by Ropke and Pisinger [21] that defines the two clusters using the Kruskal's algorithm (Kruskal [12]) to find a minimal spanning tree. This algorithm starts with isolated nodes and connects them iteratively until obtaining a spanning tree. In our case, we stop the algorithm just before the last iteration when there are two connected components. One of the resulting two clusters is chosen at random and its wells are removed. With this procedure, the number of removed wells might exceed q depending on the number of wells in the last cluster.

Neighbor graph removal. This procedure uses historical information to remove wells. This information is stored in a complete directed and weighted graph, called the neighbor graph by Ropke and Pisinger [21]. The nodes in this graph represent the wells and the initial positions of the rigs. The weight of an arc (i, j) is the cost of the best solution found so far in which node i is visited just before node j. When a new solution is found, these weights are updated, if necessary. Given a current solution s, this removal procedure computes a score for each well i by summing up the weights in the neighbor graph of the arcs incident to i in solution s. The wells are then sorted in decreasing order of their score and the RWS procedure is applied to find a well to remove. The scores of the nodes adjacent to the removed node are recomputed before selecting another node to remove.

Request graph removal. This is another procedure based on historical information which is stored this time on a complete but undirected graph, called a request graph by Ropke and Pisinger [21]. The nodes are the same as in the neighbor graph. The weight of an edge (i, j) is the number of times that wells i and j have been served by the same rig in the B best solutions (top-B) observed so far in the search (for our tests, $B = 100$). When a new top-B solution is identified, the weights are adjusted according to the solutions entering and leaving the top-B solutions. Given a well i, well j is considered more related to i if it presents the largest edge weight in the current request graph. This relatedness measure is used as in the Shaw removal procedure described above.

4.2.2 Insertion procedures

Here, we describe the three insertion procedures used in the ALNS algorithm that are also derived from the ones proposed by Ropke and Pisinger [21] and Ribeiro et al. [19]. Let D be the set of removed wells to which we append all unserviced wells in the current solution s and consider the current destroyed solution that consists of the set of modified routes obtained after the well removals.

Basic greedy insertion. This greedy procedure inserts sequentially the wells in D according to the reverse order of their removals. Given the current destroyed solution, it inserts the current well in D into its best feasible position in any of the routes. Feasibility is checked with respect to the planning horizon H for every possible insertion.

Extensive greedy insertion. This greedy procedure also inserts sequentially the wells in D but, at each iteration, it finds the best insertion of all wells remaining in D and selects the one that yields the largest cost decrease while respecting the planning horizon H. This process is repeated until no more well can be inserted.

Regret insertion. This procedure tries to improve the myopic behavior of the greedy procedures. For each well $i \in D$, it computes a regret value equal to the difference between the cost of two solutions in which the well is inserted in its best or second best rig routes. The well i with the maximum regret value is chosen to be inserted in the current destroyed solution. Ties are broken by selecting the lowest cost insertion. This concept can be extended by considering not only the cost difference defined above, but by also considering the cost difference of inserting a well in its best, its 2^{nd}-best, its 3^{rd}-best, ..., or its κ^{th}-best route, where κ is a user-defined parameter. In our experiments, we used $\kappa = |K|$.

4.3 Hybrid genetic algorithm

This section introduces a hybrid genetic search with advanced diversity control for the WRRP. Contrasting with the most recent successful population-based methods for vehicle routing problems (Prins [17], Vidal et al. [27]), the proposed method does not rely on a giant-tour representation with a *Split* algorithm. Instead, it exploits a route representation as a permutation of well visits and delimiters representing specific rigs. One additional virtual rig is used to model unserviced wells. The size of the solution representation is thus $S = |K| + |W| + 1$. One such solution can be visualized on the top of Figure 1 ("Parent 1"). Rk corresponds to the start of the route for rig k and VR to the virtual rig. The other indices are customers. This solution contains three routes : rig R1 services wells 1 and 4, R2 services wells 3 and 6, and finally R3 services wells 8, 7 and 9. The other wells 2 and 5 are not serviced during the planning horizon.

Iterative generation of individuals. The algorithm starts from an initial population of μ solutions, generated by randomly assigning wells to rigs, arranging them in random order, and applying the local-improvement procedure described below. Then, the method iteratively selects two parents by binary tournament and applies an Ordered Crossover (OX) to generate a single offspring. This offspring undergoes the local-improvement procedure, a chromosome reorganization, and is added to the population.

The crossover OX is illustrated in Figure 1. Two integers α and β with $\alpha \leq \beta$ are randomly selected following a uniform probability distribution in $[1, S]$. The sequence of wells and rigs of Parent 1, located between α and β, is directly transcribed in the offspring. In the example, the sequence (R2,3,6,R3,8,7) is transmitted. Parent 2 is then swept onward, considering first the positions $(\beta{+}1, \ldots, S)$, and then $(1, \ldots, \beta)$. Any well or rig which is not already included in the offspring is transmitted. These new elements are positioned after β, coming back to the beginning of the offspring when the position S is reached.

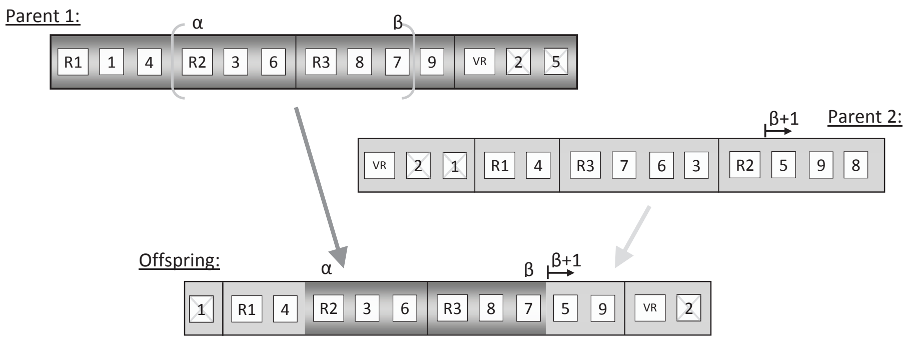
Parent 1:
α
β
R1 1 4 R2 3 6 R3 8 7 9 VR 2 5
β+1
Parent 2:
VR 2 1 R1 4 R3 7 6 3 R2 5 9 8
Offspring:
α
β
β+1
1 R1 4 R2 3 6 R3 8 7 5 9 VR 2

Fig. 1 Crossover OX on the complete solution representation

Finally, the wells located prior to the first rig, if any, are relocated at the end of the last route, and any well which cannot be serviced by its assigned rig is relocated in the virtual route after VR.

Solution cost. A solution s corresponds to a set of routes $\{r^0, r^1, \dots, r^{|K|}\}$, where r^0 is assigned to the virtual rig and route r^k to rig $k \in K$. The first element of route r^k, denoted $r^k(1)$, corresponds to the rig initial location whereas $r^k(i)$ denotes its i-th element. During the local search, the cost of a solution s is computed as follows:

$$c(s) = \sum_{k \in K} \sum_{i=2}^{|r^k|} \ell_{r^k(i)} t^k_{r^k(i)} + \sum_{i=2}^{|r^0|} \ell_{r^0(i)} H, \tag{6}$$

where t_i^k indicates the time period at which well i is serviced in the route for rig k. The first term in (6) represents the sum of the production loss at the serviced wells, whereas the second term is the penalty for the unserviced wells that are assigned to the virtual route.

This definition allows for any well i to be serviced at a time period $t_i^k > H$ (i.e., after the end of the planning horizon) for a cost of $\ell_i t_i^k$. Yet, relocating such a well to the virtual route leads to a strict cost improvement, from $\ell_i t_i^k$ to $\ell_i H$. This move is included in the proposed local-improvement procedure, and thus any local optimum is guaranteed to not include a service completion after the end of the planning horizon.

Local search. Any solution produced by the crossover is submitted to a local-improvement procedure, based on the standard neighborhoods Relocate, Swap, 2-opt, 2-opt*, and Cross limited to sequences of less than 2 wells. We refer to Vidal et al. [28] for a thorough description of these standard vehicle routing problem neighborhoods and pruning procedures. Moves are explored in random order, any improving move being directly applied.

To evaluate each move in amortized constant time, we rely as in Vidal et al. [29] on auxiliary data structures, pre-processed on subsequences of visits from the incumbent solution. Thus, for any sequence σ of a single or consecutive visits in the incumbent solution, the method keeps track of:

$T(\sigma)$: the duration to perform the visit sequence,
$\Lambda(\sigma)$: the sum of the production loss rates in the sequence,
$C(\sigma)$: the cost (production loss) of the sequence when starting at time 0,
$E(\sigma)$: the maximum equipment level to service the wells in the sequence.

For a sequence containing a single well $\sigma = (i)$, the duration equals the well service time $T(\sigma) = \tau_i$, the production loss parameters are given by $\Lambda(\sigma) = \ell_i$ and $C(\sigma) = \tau_i \ell_i$, and the service level is $E(\sigma) = e_i$. Equations (7)–(10), similar to Silva et al. [24], enable to obtain the same auxiliary data structures for any longer subsequence $\sigma \oplus \sigma'$ issued from the concatenation of subsequences $\sigma = (\sigma_u, \ldots, \sigma_v)$ and $\sigma' = (\sigma'_w, \ldots, \sigma'_x)$. These equations are used for data structure pre-processing and move evaluations. The considered local search moves are equivalent to a recombination of a bounded number of sequences of visits of the incumbent solution, such that these equations are applied a constant number of times for each move.

$$T(\sigma \oplus \sigma') = T(\sigma) + t_{\sigma_v \sigma'_w} + T(\sigma') \tag{7}$$
$$\Lambda(\sigma \oplus \sigma') = \Lambda(\sigma) + \Lambda(\sigma') \tag{8}$$
$$C(\sigma \oplus \sigma') = C(\sigma) + \Lambda(\sigma')(T(\sigma) + t_{\sigma_v \sigma'_w}) + C(\sigma') \tag{9}$$
$$E(\sigma \oplus \sigma') = \max\{E(\sigma), E(\sigma')\} \tag{10}$$

where $t_{\sigma_v \sigma'_w}$ denotes the travel time from σ_v to σ'_w.

The local-improvement procedure stops in a local minimum when all moves have been successively tried without success. Any move leading to an incompatible pair of rig and well is rejected. Finally, moves are only attempted between locations i and j (well or initial rig position) if j is among the $\Gamma = 30$ closest locations of i. The parameter Γ is usually called the granularity threshold (Toth and Vigo [25]).

Chromosome reorganization. To maximize the chances of connecting related parts of the solution during the crossover operation, any solution issued from the local search (and also during population initialization) is re-organized to produce the chromosome representation. For each route r^k, $k \in K$, the polar angle $\Theta(r^k)$ of the route barycenter is computed as

$$\Theta(r^k) = \arctan\left(\frac{\sum_{i=1}^{|r^k|}(y(i) - Y_0)}{\sum_{i=1}^{|r^k|}(x(i) - X_0)}\right), \tag{11}$$

where $(x(i), y(i))$ are the coordinates of the i-th location in route r^k, and (X_0, Y_0) is the barycenter of the set of wells and rig initial locations. For the virtual route, the computation is the same but the initial position of the virtual rig is not counted (that is, the sums start at $i = 2$). The routes are then ordered in the solution representation by increasing polar angle, starting from $\Theta = 0$.

Population management. We adopt the diversity management strategy of Vidal et al. [27], and thus integrate diversity measures directly in the individual fitness evaluation. The biased fitness $\phi_{\mathcal{P}}(s)$ of any individual s is a weighted sum between its rank in the population $\mathcal{P}$ in terms of contribution to the population diversity $\phi^{\mathrm{DIV}}(s)$, evaluated as a Hamming distance to the others, and its rank with respect to solution cost $\phi^{\mathrm{COST}}(s)$. It is given by:

$$\phi_{\mathcal{P}}(s) = \phi^{\mathrm{COST}}(s) + \left(1 - \frac{\mu^{\mathrm{ELITE}}}{|\mathcal{P}|}\right) \phi^{\mathrm{DIV}}(s), \tag{12}$$

where the parameter μ^{ELITE} governs the role of each criterion. This fitness measure is used for both parent and survivor selections.

All individuals issued from the crossover operation and the local search are directly included in the population. The population size is kept within a range $[\mu, \mu + \lambda]$ by operating a survivor selection phase when the maximum size of $\mu + \lambda$ is attained. During this phase, the λ worst individuals are iteratively selected out. For our tests, we use the same parameter setting $(\mu^{\mathrm{ELITE}}, \mu, \lambda) = (8, 25, 40)$ as in Vidal et al. [27].

Stopping criterion. The algorithm iterates until a maximum number $I^{hga}_{max} = 500$ consecutive iterations without improvement of the best solution is reached. The best solution is then reported.

5 Computational experiments

The VNS, BPC, ALNS and HGA heuristics were coded in C++ and run on a Linux PC equipped with an Intel Core i7-3770 processor clocked at 3.4 GHz. For our main results, the algorithms were tested on the WRRP instances introduced by Ribeiro et al. [18], i.e., a set of 80 instances of practical size involving 100 or 200 wells, 5 or 10 rigs, a horizon length of $H = 200$ or $H = 300$ time periods (10 instances for each possible parameter combination). The horizon lengths $H = 200$ and $H = 300$ correspond to approximately 14 and 21 days, respectively. Only short maintenance services (cleaning and stimulation) lasting between 2 and 10 periods are requested and the travel times are computed as the Euclidean distances between the well locations. To assess the scalability of the algorithms, additional instances involving different numbers of rigs and wells, and different horizon lengths were also generated in a similar fashion as Ribeiro et al. [18]. All instances are available at `http://www.gerad.ca/~guyd/wrrp.html`.

For all tests, all heuristics were executed 10 times with different random seeds for each instance, except the BPC heuristic that was run once because it is fully deterministic. In the following, we first report our main results before presenting scalability results as well as sensitivity analysis results.

5.1 Main results

For our main computational experiments, we solve the instances of Ribeiro et al. [18] using the four algorithms presented above. The detailed results of these experiments can be found in the Appendix. Tables 1–4 (one table per heuristic) summarize these results per group of instances with the same parameter combination. These tables are split in two parts: the upper one for instances with 100 wells and the bottom one with 200 wells. For each group of 10 instances and each heuristic, these tables report:

- the instances characteristics (No. Wells/No. Rigs/Horizon);
- the number of best solutions found out of these instances (Best No.);
- the average computational time per run in seconds (Time);
- the average percentage of deviation of the cost of every solution with respect to the best-known solution ($\text{Dev}_{\text{Every}}$);
- the average percentage of deviation of the cost of the best solution (out of 10 runs) with respect to the best-known (Dev_{Best}).

Because a single deterministic run was executed for BPC, both average deviations are the same and we only report $\text{Dev}_{\text{Every}}$.

For a given instance, the deviation of a solution cost is computed as

$$\text{Dev} = 100 \times (\text{BKUB - UB})/\text{BKUB},$$

where BKUB indicates the best-known solution value and UB is the considered solution cost. Note that these values are negative because the constant term in (1) (that is, the maximum total production loss if no maintenance was performed in the horizon) has been removed from the cost computation. The detailed solution values on each instance are provided in Tables 11–18 of the Appendix. Out of the 80 best-known solution values, 66 optimal ones were provided by the exact BPC algorithm of Ribeiro et al. [18] whereas 14 new best-known values (in bold face in Tables 11–18) were found by HGA and reasserted by ALNS (5) and BPC (4).

Table 1 Results for VNS

		VNS		
Combination	**Best (No.)**	**Time (s)**	**$\text{Dev}_{\text{Every}}$ (%)**	**Dev_{Best} (%)**
100/ 5/200	0	6.5	6.96	4.86
100/10/200	0	9.2	6.39	4.91
100/ 5/300	0	9.4	7.88	5.96
100/10/300	0	7.8	6.87	5.41
200/ 5/200	0	25.7	8.92	7.03
200/10/200	0	51.9	9.89	8.40
200/ 5/300	0	44.5	12.56	10.59
200/10/300	0	67.6	12.87	11.29
Average	**0.00**	**27.83**	**9.04**	**7.31**

Table 2 Results for BPC

	BPC		
Combination	**Best (No.)**	**Time (s)**	**Dev$_{\textbf{Every}}$ (%)**
100/ 5/200	10	0.5	0.00
100/10/200	9	2.3	0.00
100/ 5/300	4	30.7	0.08
100/10/300	5	23.9	0.02
200/ 5/200	9	14.1	0.01
200/10/200	8	27.9	0.03
200/ 5/300	8	296.7	0.00
200/10/300	2	815.6	0.28
Average	**6.88**	**151.46**	**0.05**

Table 3 Results for ALNS

		ALNS		
Combination	**Best (No.)**	**Time (s)**	**Dev$_{\textbf{Every}}$ (%)**	**Dev$_{\textbf{Best}}$ (%)**
100/ 5/200	10	6.8	0.01	0.00
100/10/200	9	9.3	0.04	0.00
100/ 5/300	9	9.5	0.04	0.01
100/10/300	10	7.5	0.02	0.00
200/ 5/200	7	26.6	0.23	0.02
200/10/200	7	52.4	0.22	0.01
200/ 5/300	2	45.4	0.52	0.11
200/10/300	3	66.2	0.46	0.12
Average	**7.13**	**27.96**	**0.19**	**0.03**

Table 4 Results for HGA

		HGA		
Combination	**Best (No.)**	**Time (s)**	**Dev$_{\textbf{Every}}$ (%)**	**Dev$_{\textbf{Best}}$ (%)**
100/ 5/200	10	5.7	0.00	0.00
100/10/200	10	4.8	0.01	0.00
100/ 5/300	10	5.5	0.00	0.00
100/10/300	10	4.3	0.00	0.00
200/ 5/200	10	26.5	0.01	0.00
200/10/200	10	26.7	0.02	0.00
200/ 5/300	10	26.7	0.06	0.00
200/10/300	10	21.1	0.06	0.00
Average	**10.0**	**15.16**	**0.02**	**0.00**

Regarding the quality of the solutions, HGA is the only heuristic that found all the best solution values. It is followed by ALNS and BPC with 57 and 55 best solution values, respectively. None were found by VNS for which the average deviation to the best solutions exceeds 9%, strongly dominated by the other heuristics: HGA (0.02%), BPC (0.05%), and ALNS (0.19%). Given this poor performance, the VNS heuristic is omitted from the subsequent analysis.

For instances with a horizon $H = 200$, the average deviation is the same for HGA and BPC at 0.01% but raises to 0.13% for ALNS. For instances with a horizon $H = 300$, HGA average deviation at 0.03% dominates BPC (0.09%) and ALNS (0.26%).

An analysis of the computational times provides a different angle as one can argue that BPC is executed only once per instance whereas the best solutions of HGA and ALNS required 10 runs. With this in mind, BPC is faster than 10 runs of HGA and ALNS on almost all instances except those with 200 wells and $H = 300$. Still, remark that even a single run of HGA already leads to solutions of higher average quality than those produced by the other methods. It is thus possible to reduce the number of random runs to 5 or less without compromising the quality of the best solutions. This would not be the case for ALNS that yields relatively large average deviation for instances with 200 wells. Finally, observe how stable are the HGA computational times: around 5 seconds for instances with $H = 200$, and 25 seconds when $H = 300$.

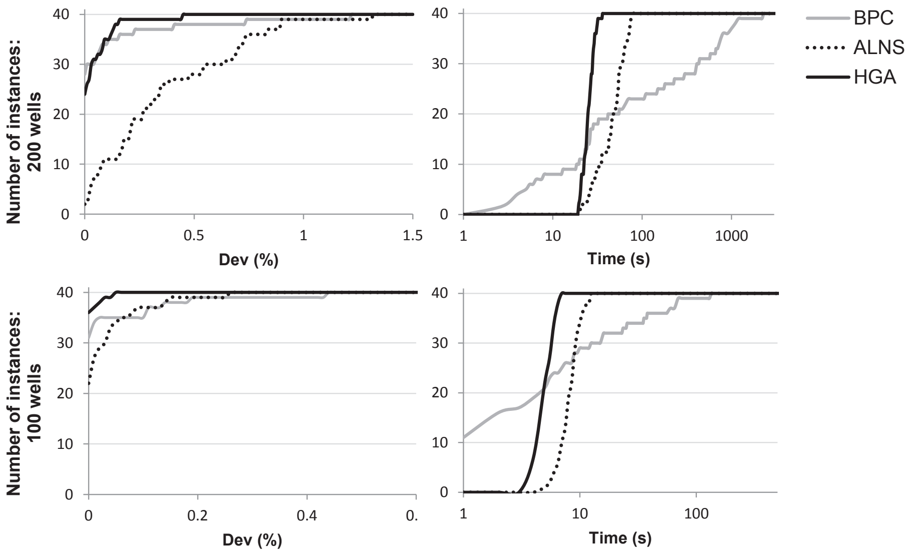

Fig. 2 Left: Number of instances whose computed solution cost is within a given deviation. Right: Number of instances solved within a computational time.

For the three best heuristics (BPC, ALNS, and HGA), Figure 2 reports the number of instances solved whose cost is within a given deviation (on the left) and the number of instances solved within a computational time (on the right). A logarithmic scale is used for time values. Instances with 100 or 200 wells are reported in different graphs, at the bottom and top, respectively. A point $(x, n(x))$ in a graph means that $n(x)$ solutions have been found by a given method with a deviation (or a time) less than or equal to x. This figure illustrates again that BPC and HGA find near-optimal solutions in most cases.

On instances with 200 wells, a deviation larger than 0.5% is obtained only on three runs for BPC, and never for HGA. From this figure, one can also remark that the variance of the computational time for HGA and ALNS is very small. For example, all instances with 200 wells are solved by HGA within 19 and 36 seconds. In comparison, the distribution of the computational times for BPC is more spread: 15 instances are solved in less than 25 seconds, but more than 750 seconds are needed for 5 instances out of 40. This increased variability is inherent in most mathematical-programming-based approaches because of the use of pseudo-polynomial sub-procedures and branching. In contrast, a classic metaheuristic based on local or large neighborhood search performs a polynomial – usually quadratic – number of operations for each new solution, and the number of solutions generated before reaching the termination criterion grows almost linearly with problem size. Promising perspectives of research involve blending these methods to circumvent the issue of the computational time variability while achieving higher quality solutions.

5.2 Scalability

To assess the scalability of our proposed heuristics, we performed tests on additional instances. In fact, we wanted to observe the impact of increasing separately the number of wells, the number of rigs, and the horizon length on the solution quality and the computational time. With the procedure of Ribeiro et al. [18], we generated new instances involving 50, 150, 250, and 300 wells (with 10 rigs and $H = 200$), involving 2, 4, 6, 8, and 12 rigs (with 200 wells and $H = 200$), and involving $H = 100, 150, 250, 350$ (with 200 wells and 10 rigs). For each combination (number of wells, number of rigs, and horizon length), 10 instances were generated. This set of instances was completed with instances used for the main tests above, namely, those in combinations 100/10/200, 200/10/200, and 200/10/300.

All these instances were solved using BPC, ALNS, and HGA (10 runs were executed by ALNS and HGA). The results of these tests are reported in Tables 5–7, where Table 5 is dedicated to a varying number of wells, Table 6 to a variable number of rigs, and Table 7 to a variable horizon length. For each instance set, we report the statistics Time, $\text{Dev}_{\text{Every}}$, and Dev_{Best} described in the previous section for each heuristic (except Dev_{Best} for BPC that is run once for each instance).

These results indicate that an increase in any dimension (number of wells, number of rigs, or horizon length) increases the average computational time of BPC and ALNS. The increase rate is, however, more important for BPC that has an exponential complexity, especially with respect to the horizon length. As for HGA, the average computational time does not really vary with the number of rigs and the horizon length. It increases though with the number of wells at a rate similar to that of ALNS. Figure 3 summarizes the empirical running time as a function of the number of rigs or wells, and of horizon length.

Table 5 Results with a variable number of wells (10 rigs and $H = 200$)

No. Wells	BPC		ALNS			HGA		
	$\text{Dev}_{\text{Every}}$ (%)	Time (s)	$\text{Dev}_{\text{Every}}$ (%)	Dev_{Best} (%)	Time (s)	$\text{Dev}_{\text{Every}}$ (%)	Dev_{Best} (%)	Time (s)
50	0.12	0.1	0.02	0.00	1.4	0.00	0.00	1.2
100	0.00	2.3	0.04	0.00	9.3	0.01	0.00	4.8
150	0.11	11.6	0.05	0.01	23.4	0.01	0.00	11.7
200	0.03	27.9	0.22	0.01	52.4	0.02	0.00	26.7
250	0.17	57.5	0.29	0.10	74.5	0.11	0.00	39.4
300	0.14	125.5	0.54	0.12	112.9	0.11	0.00	60.0

Table 6 Results with a variable number of rigs (200 wells and $H = 200$)

No. Rigs	BPC		ALNS			HGA		
	$\text{Dev}_{\text{Every}}$ (%)	Time (s)	$\text{Dev}_{\text{Every}}$ (%)	Dev_{Best} (%)	Time (s)	$\text{Dev}_{\text{Every}}$ (%)	Dev_{Best} (%)	Time (s)
2	0.10	0.5	0.05	0.00	6.7	0.00	0.00	16.5
4	0.07	1.2	0.09	0.00	16.8	0.00	0.00	19.4
6	0.11	6.4	0.11	0.01	28.6	0.02	0.00	19.3
8	0.16	18.5	0.20	0.03	39.4	0.02	0.00	21.3
10	0.03	27.9	0.22	0.01	52.4	0.02	0.00	26.7
12	0.15	92.4	0.26	0.09	57.1	0.02	0.00	20.4

Table 7 Results with a variable horizon length (200 wells and 10 rigs)

H	BPC		ALNS			HGA		
	$\text{Dev}_{\text{Every}}$ (%)	Time (s)	$\text{Dev}_{\text{Every}}$ (%)	Dev_{Best} (%)	Time (s)	$\text{Dev}_{\text{Every}}$ (%)	Dev_{Best} (%)	Time (s)
100	0.00	0.5	0.12	0.03	19.6	0.01	0.00	20.3
150	0.00	5.2	0.24	0.06	33.4	0.02	0.00	21.3
200	0.03	27.9	0.22	0.01	52.4	0.02	0.00	26.7
250	0.20	178.8	0.38	0.14	56.1	0.03	0.00	21.5
300	0.28	815.6	0.46	0.12	66.2	0.06	0.00	21.1
350	0.38	2568.4	0.44	0.16	57.4	0.02	0.00	18.3

Concerning solution quality, we observe that the performance of HGA remains outstanding compared to the other two heuristics. The average quality of the solutions it produces (column $\text{Dev}_{\text{Every}}$) seems to slightly deteriorate with an increase in the number of wells but does not really vary with the other two dimensions. At the opposite, the quality of the solutions obtained by BPC and ALNS clearly diminishes with an increase in any dimension. The deterioration is, however, more important for ALNS.

5.3 Sensitivity analysis

To assess some of the main settings and components of BPC, ALNS, and HGA, we also performed a sensitivity analysis on these settings and components. For this purpose, we selected five of the largest instance sets used in our previous tests, namely, those with combinations 200/10/200, 200/10/250, 200/10/300, 250/10/200, and 300/10/200, yielding a total of 50 instances. For each heuristic, we solved these 50 instances (10 runs per instance for ALNS and HGA)

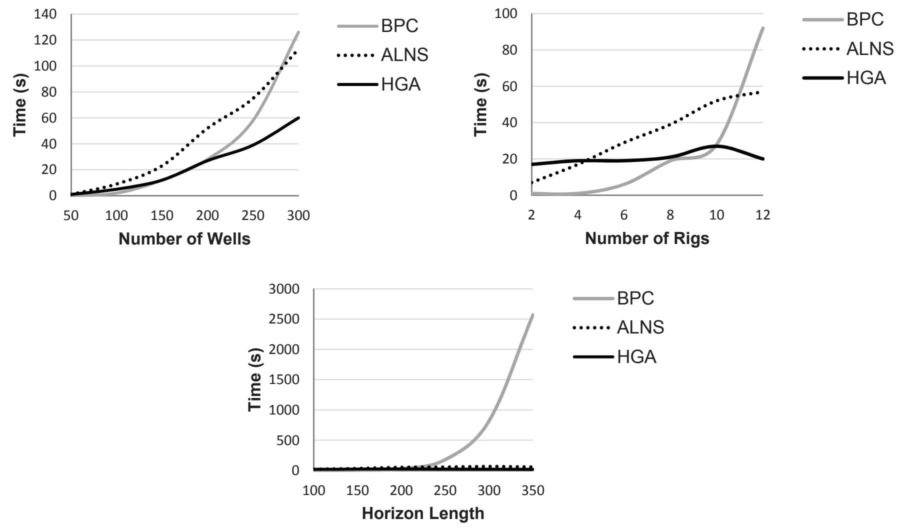

Fig. 3 Running time as a function of the number of rigs or wells, and of horizon length.

Table 8 Sensitivity analysis results for BPC

Algorithm configuration	**Dev$_{\mathbf{Every}}$ (%)**	**Time (s)**
Base ($I^{tabu}_{max} = 100$, $A^{min} = 5$)	0.16	241.1
No tabu ($I^{tabu}_{max} = 0$)	0.50	446.4
$I^{tabu}_{max} = 50$	0.11	212.4
$I^{tabu}_{max} = 150$	0.19	229.9
No heuristic labeling ($A^{min} = 0$)	0.77	39.0
$A^{min} = 10$	0.15	1005.1
No cuts	0.67	143.7

with different algorithm configurations including the one used for the previous tests (denoted Base). The results of these experiments are given in Tables 8 – 10 that are discussed separately below.

For BPC, we tested the impact of three components: the use of the tabu search column generator, the use of the heuristic labeling algorithm, and the use of cuts. For the tabu search component, we considered four levels of usage determined by I^{tabu}_{max}, the maximum number of iterations per route: $I^{tabu}_{max} = 0$ (No tabu), 50, 100 (Base), and 150. The results of Table 8 show that not using the tabu search column generator at all is very bad to the performance of BPC, both in terms of average computational time and solution quality. For these instances, the configuration with $I^{tabu}_{max} = 50$ outperforms the other on both criteria. This setting was not use for our main tests because the parameter values were adjusted using a different subset of instances. Finally, increasing the maximum number of iterations per route to 150 does not help improving the average solution quality; in fact, compared to the Base configuration

Table 9 Sensitivity analysis results for ALNS

Algorithm configuration	$\mathbf{Dev_{Every}}$ (%)	$\mathbf{Dev_{Best}}$ (%)	**Time (s)**
Base ($0.1\|W\| \leq q \leq 0.4\|W\|$, $I^{alns}_{max} = 50,000$)	0.38	0.10	72.8
$0.3\|W\| \leq q \leq 0.4\|W\|$	0.35	0.11	89.1
$0.1\|W\| \leq q \leq 0.6\|W\|$	0.40	0.12	87.7
$I^{alns}_{max} = 25,000$	0.53	0.15	35.3
$I^{alns}_{max} = 75,000$	0.30	0.08	96.5
No post-optimization local search	0.41	0.14	65.7

Table 10 Sensitivity analysis results for HGA

Algorithm configuration	$\mathbf{Dev_{Every}}$ (%)	$\mathbf{Dev_{Best}}$ (%)	**Time (s)**
Base ($I^{hga}_{max} = 500$, $(\mu^{ELITE}, \mu, \lambda) = (8, 25, 40)$)	0.06	0.00	32.3
$(\mu^{ELITE}, \mu, \lambda) = (4, 12, 20)$	0.10	0.00	25.0
$(\mu^{ELITE}, \mu, \lambda) = (16, 50, 80)$	0.05	0.00	45.5
$I^{hga}_{max} = 250$	0.09	0.00	25.1
$I^{hga}_{max} = 1000$	0.05	0.00	46.6
No advanced diversity control	0.25	0.02	21.7

($I^{tabu}_{max} = 100$), it deteriorates from $\text{Dev}_{\text{Every}} = 0.16\%$ to 0.19%. One reason that can explain this deterioration is a possible instability of the heuristic branching process (a better linear relaxation solution does not guarantee a better integer solution). Alternatively, with a higher number of tabu search iterations per route, the heuristic labeling algorithm is, in general, invoked less times. Given that heuristic labeling typically generates better columns (with smaller reduced costs) than tabu search, there are less opportunities to generate these good columns. For the heuristic labeling component, we tested three levels of usage determined by A^{min}, the minimum number of incoming and outgoing arcs to keep at each node of the networks: $A^{min} = 0$ (No heuristic labeling), 5 (Base), and 10. For this component, we observe that not using it yields solutions of very poor quality ($\text{Dev}_{\text{Every}} = 0.77\%$) while using it with networks whose sizes are not sufficiently reduced ($A^{min} = 10$ instead of 5) considerably increases the average computational time (from 241.1 seconds to 1005.1). Finally, the results clearly highlight that using cuts is necessary to achieve good quality solutions.

For ALNS, we also tested the impact of three algorithm features: the interval in which the number of wells q to remove at each iteration is selected, the total number of iterations to perform, and the use of the post-optimization local search heuristic. The average results are reported in Table 9. For the selection of the number of wells to remove, we observe that using the interval $[0.3|W|, 0.4|W|]$ (instead of $[0.1|W|, 0.4|W|]$) provides slightly better solutions on average ($\text{Dev}_{\text{Every}}$ drops from 0.38% to 0.35%) but less good best solutions. Indeed, this interval avoids using too small-sized neighborhoods that do not offer much potential for improvements. On the other hand, it requires more computational time than the base algorithm. Alternatively, using a wider interval ($[0.1|W|, 0.6|W|]$) deteriorates the quality of the average solution and of the best solution. This can be explained by the fact that the repair procedures loose efficiency as the size of neighborhood grow. Consequently, it becomes counterproductive to define too large neighborhoods beside being more time-consuming. The results also show that increasing the total number of iterations improves solution quality but requires more computational time. Clearly, executing only 50,000 iterations for the results presented in the two previous sections aimed at limiting computational time as better quality solutions could have been obtained if more iterations would have been allowed (for instance, setting $I_{max}^{alns} = 75,000$ instead of 50,000 in the base heuristic decreases $\text{Dev}_{\text{Every}}$ from 0.38% to 0.30% and Dev_{Best} from 0.10% to 0.08%). Finally, we observe that the post-optimization local search procedure increases the computational time by about 10% to decrease the percentage of deviation from the best solution cost ($\text{Dev}_{\text{Every}}$) by about 8% (from 0.41% to 0.38%). In fact, the procedure succeeded to improve the solution in only 15 instances (out of 50) but quite substantially in a few of them.

For HGA, we again tested the sensitivity of the algorithm to three component settings: the size of the population, the maximum number of iterations without improvement, and the use of an advanced diversity control utilized for both parent selection and survivors selection. The results in Table 10 show that decreasing population size (by halving the values of the parameters μ^{ELITE}, μ, and λ) the average computational time is reduced whereas the average solution quality is slightly deteriorated. Increasing it yields the inverse behavior. Similarly, reducing the maximum number of iterations without improvement I_{max}^{hga} from 500 to 250 obviously reduces average solution quality and computational time. Increasing it shows that average solution quality can be further improved. Finally, removing the advanced diversity control (that is, removing the second term in formula (12)) is highly detrimental to the performance of HGA: $\text{Dev}_{\text{Every}}$ increases from 0.06 to 0.25. Without this feature, it is the only time that HGA does not succeed to obtain the best-known solution over ten runs for some instances. This feature is thus a key element in HGA which enables a thorough exploration of a variety of high-quality and diverse solutions.

6 Conclusions

For the workover rig routing problem (WRRP) with a heterogenous fleet of rigs and a finite planning horizon, this paper introduced three heuristic solution algorithms: a branch-price-and-cut (BPC) heuristic, an adaptive large neighborhood search (ALNS), and a hybrid genetic algorithm (HGA). The solutions obtained by these heuristics were compared to those produced by an existing variable neighborhood search (VNS). Our computational results on existing instances from the literature indicate that HGA outperforms the other heuristics on average and in most cases. It yields the best quality solutions and the fastest computational times except for the smallest instances that can be solved more rapidly by BPC. BPC also yields high-quality solutions but with large computational times for the largest instances. For VNS and ALNS, the computational times are small (approximately one minute in the worst case). The solution quality is very good for ALNS but very poor for VNS. Furthermore, additional tests on newly generated instances show that HGA is the most scalable of these heuristics with respect to the number of wells, the number of rigs, and the horizon length.

As future research, our work suggests that combining HGA with BPC could yield a powerful matheuristic. Furthermore, HGA could be exploited within an exact BPC algorithm such as that of Ribeiro et al. [18] to compute proven optimal solutions in very reasonable computational times. Finally, another interesting research avenue is to study the dynamic version of the WRRP in which rig routes can be modified dynamically as new maintenance requests are issued. Indeed, dynamism is very important for oil companies as the production of highly productive wells may decrease suddenly, in which case it might be preferable to immediately revise the planned routes of the rigs in order to avoid high losses. This dynamic WRPP requires the development of new solution algorithms that would integrate stochastic components.

Acknowledgements Glaydston Mattos Ribeiro acknowledges the Espírito Santo Research Foundation and the National Council for Scientific and Technological Development for their financial support. Guy Desaulniers and Jacques Desrosiers acknowledge the National Science and Engineering Research Council of Canada for its financial support.

References

1. D.J. Aloise, D. Aloise, C.T.M. Rocha, C.C. Ribeiro, J.C. Ribeiro Filho and L.S.S. Moura, Scheduling workover rigs for onshore oil production, Discrete Applied Mathematics, 154(5), 695–702 (2006)
2. R. Baldacci, A. Mingozzi and R. Roberti, New route relaxation and pricing strategies for the vehicle routing problem, Operations Research, 59(5), 1263–1283 (2011)
3. C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P. H. Vance, Branch-and-price: Column generation for solving huge integer programs, Operations Research, 46(3), 316–329 (1998)
4. L.R. Costa, Solving the workover rigs routing problem, Master's thesis, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil (2005)
5. L.R. Costa and V.J.M Ferreira Filho, A heuristic of dynamic mounting for the workover rigs routing problem, In Proceedings of XXXVII SBPO Brazilian Symposium on Operations Research, 2176–2187 (2005)
6. G. Desaulniers, F. Lessard and A. Hadjar, Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows, Transportation Science, 42(3), 387–404 (2008)
7. J. Desrosiers and M.E. Lübbecke. Branch-price-and-cut algorithms. In: J.J. Cochran, L.A. Cox Jr., P. Keskinocak, J.P. Kharoufeh and J.C. Smith, editors, Wiley Encyclopedia of Operations Research and Management Science 8, Wiley, New York, NY (2010)
8. C. Duhamel, A.C. Santos and L.M. Gueguen, Models and hybrid methods for the onshore wells maintenance problem, Computers & Operations Research, 39(12), 2944–2953 (2012)
9. P. Hansen and N. Mladenović. Variable neighborhood search. In: F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, 145–184, Kluwer Academic Publishers (2003)
10. M. Jepsen, B. Petersen, S. Spoorendonk and D. Pisinger, Subset-row inequalities applied to the vehicle-routing problem with time windows, Operations Research, 56(2), 497–511 (2008)

11. A. Ken and M. Stewart, Surface operations in petroleum production, ISBN: 0444426779, Burlington: Elsevier, (1987)
12. J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem. In: Proceedings of the American Mathematical Society, American Mathematical Society, American Mathematical Society, 48–50 (1956)
13. M.E. Lübbecke and J. Desrosiers, Selected topics in column generation, Operations Research, 53(6), 1007–1023 (2005)
14. N. Mladenović and P. Hansen, Variable neighborhood search, Computers & Operations Research, 24(11), 1097–1100 (1997)
15. T.A. Neves. Heuristics with adaptive memory applied to workover rig routing and scheduling problem, Master's thesis, Fluminense Federal University, Niterói, Brazil, (2007)
16. A.V.F. Pacheco, G.M. Ribeiro and G.R. Mauri, A GRASP with path-relinking for the workover rig scheduling problem, International Journal of Natural Computing Research, 1(2), 1–14 (2010)
17. C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem, Computers & Operations Research, 31(12), 1985–2002 (2004)
18. G.M. Ribeiro, G. Desaulniers and J. Desrosiers, A branch-price-and-cut algorithm for the workover rig routing problem, Computers & Operations Research, 39(12), 3305–3315 (2012)
19. G.M. Ribeiro, G. Laporte and G.R. Mauri, A comparison of three metaheuristics for the workover rig routing problem, European Journal of Operational Research, 220(1), 28–38 (2012)
20. G.M. Ribeiro, G.R. Mauri and L.A.N. Lorena, A simple and robust simulated annealing algorithm for scheduling workover rigs on onshore oil fields, Computers & Industrial Engineering, 60(4), 519–526 (2011)
21. S. Ropke and D. Pisinger, A unified heuristic for a large class of vehicle routing problems with backhauls, European Journal of Operational Research, 171(3), 750–775 (2006)
22. S. Sahni and T. Gonzalez, P-complete approximation problems, Journal of the ACM, 23(3), 555–565 (1976)
23. P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems, Technical Report, University of Strathclyde, Glasgow (1997)
24. M.M. Silva, A. Subramanian, Vidal, T. and L.S Ochi, A simple and effective metaheuristic for the minimum latency problem, 221(3), 513–520 (2012)
25. P. Toth and D. Vigo, The granular tabu search and its application to the vehicle-routing problem, INFORMS Journal on Computing, 15(4), 333–346 (2003)
26. J.N. Tsitsiklis, Special cases of traveling salesman and repairman problems with time windows, Networks, 22(3), 263–282 (1992)
27. T. Vidal, T.G. Crainic, M. Gendreau, N. Lahrichi and W. Rei, A hybrid genetic algorithm for multidepot and periodic vehicle routing problems, Operations Research, 60(3), 611–624 (2012)
28. T. Vidal, T.G. Crainic, M. Gendreau and C. Prins, Heuristics for multi-attribute vehicle routing problems: A survey and synthesis, European Journal of Operational Research, 231(1), 1–21 (2014)
29. T. Vidal, T.G. Crainic, M. Gendreau and C. Prins, A unified solution framework for multi-attribute vehicle routing problems, European Journal of Operational Research, 234(3), 658–673 (2014)

A Detailed results

In this appendix, we report the detailed results of our computational experiments that were summarized in Tables 1–4. There is one table per parameter combination ($|W|$, $|K|$, H). In each table, the first column specifies the instance number (out of 10 instances). The next column indicates the best-known solution value (BKUB). Out of these 80 upper bounds, 66 correspond to optimal solutions provided by the exact BPC algorithm of Ribeiro et al. [18] and 14 are new best-known values (in bold face) obtained by HGA and reasserted by ALNS (5) and BPC (4). For each heuristic, the tables report four columns (only three for BPC): best solution value found over all runs (Best), average solution value (Avg), average computational time in seconds (Time), and the average solution value deviation with respect to BKUB in percentage (Dev) computed as Dev $= 100 \times$ (BKUB - Avg) /BKUB.

Table 11 Results for instances with 100 wells, 5 rigs and $H = 200$

Inst.	**BKUB**	**VNS**				**BPC**			**ALNS**				**HGA**			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-32275	-29991	-29472	6.5	8.68	-32275	0.6	0.00	-32275	-32275	6.8	0.00	-32275	-32275	5.6	0.00
2	-29068	-28202	-27936	6.4	3.89	-29068	0.5	0.00	-29068	-29065	6.0	0.01	-29068	-29068	5.8	0.00
3	-28466	-26717	-26001	6.5	8.66	-28466	1.2	0.00	-28466	-28466	7.9	0.00	-28466	-28466	6.6	0.00
4	-27929	-26464	-26017	6.4	6.85	-27929	0.2	0.00	-27929	-27929	6.9	0.00	-27929	-27929	6.3	0.00
5	-26398	-25377	-24555	6.4	6.98	-26398	0.4	0.00	-26398	-26388	6.2	0.04	-26398	-26398	4.8	0.00
6	-26661	-25628	-25261	6.6	5.25	-26661	0.5	0.00	-26661	-26661	8.3	0.00	-26661	-26661	5.6	0.00
7	-26128	-24765	-23723	6.5	9.20	-26128	0.1	0.00	-26128	-26128	4.4	0.00	-26128	-26128	5.8	0.00
8	-32912	-31073	-30101	6.6	8.54	-32912	0.9	0.00	-32912	-32904	8.9	0.02	-32912	-32912	4.8	0.00
9	-26704	-25207	-24889	6.5	6.80	-26704	0.2	0.00	-26704	-26704	6.3	0.00	-26704	-26704	5.4	0.00
10	-33521	-32500	-31939	6.5	4.72	-33521	0.5	0.00	-33521	-33521	6.8	0.00	-33521	-33520	6.1	0.00
Avg	-29006.2	-27592.4	-26989.4	6.5	6.96	-29006.2	0.5	0.00	-29006.2	-29004.1	6.8	0.01	-29006.2	-29006.1	5.7	0.00

Table 12 Results for instances with 100 wells, 10 rigs and $H = 200$

Inst.	**BKUB**	**VNS**				**BPC**			**ALNS**				**HGA**			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-52034	-50379	-48940	9.2	5.95	-52034	2.6	0.00	-52034	-52005	9.3	0.06	-52034	-52034	4.2	0.00
2	-46189	-44028	-43628	9.2	5.54	-46189	1.3	0.00	-46189	-46069	8.7	0.26	-46189	-46189	5.6	0.00
3	-47003	-43401	-42724	9.2	9.10	-47003	4.6	0.00	-47003	-47003	10.8	0.00	-47003	-47003	3.9	0.00
4	-46513	-44558	-44002	9.2	5.40	-46513	1.1	0.00	-46513	-46513	8.6	0.00	-46513	-46513	4.9	0.00
5	-47734	-45831	-44787	9.2	6.17	-47734	0.6	0.00	-47734	-47734	9.0	0.00	-47734	-47734	5.2	0.00
6	-41834	-39756	-39251	9.2	6.17	-41834	4.3	0.00	-41834	-41800	10.3	0.08	-41834	-41813	5.2	0.05
7	-44703	-43223	-42570	9.2	4.77	-44703	0.4	0.00	-44703	-44703	6.9	0.00	-44703	-44702	4.4	0.00
8	-51208	-48344	-47915	9.3	6.43	-51208	1.3	0.00	-51208	-51208	9.5	0.00	-51208	-51208	4.9	0.00
9	-45796	-43476	-42836	9.2	6.46	-45795	5.3	0.00	-45795	-45792	9.3	0.01	-45796	-45796	5.0	0.00
10	-51279	-47995	-47212	9.4	7.93	-51279	1.2	0.00	-51279	-51279	10.3	0.00	-51279	-51279	4.5	0.00
Avg	-47429.3	-45099.1	-44386.5	9.2	6.39	-47429.2	2.3	0.00	-47429.2	-47410.6	9.3	0.04	-47429.3	-47427.1	4.8	0.01

Table 13 Results for instances with 100 wells, 5 rigs and $H = 300$

Inst.	BKUB	VNS				BPC			ALNS				HGA			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-71487	-67008	-66505	9.3	6.97	-71487	3.2	0.00	-71487	-71477	9.8	0.01	-71487	-71487	5.4	0.00
2	-63737	-60143	-58393	9.3	8.38	-63735	8.6	0.00	-63737	-63736	8.1	0.00	-63737	-63737	7.0	0.00
3	**-63641**	-59602	-57926	9.4	8.98	-63519	130.8	0.19	**-63641**	-63619	11.2	0.03	**-63641**	-63641	6.5	0.00
4	-60144	-55426	-54605	9.4	9.21	-60131	16.0	0.02	-60144	-60112	9.7	0.05	-60144	-60136	5.9	0.01
5	-60598	-57873	-57441	9.3	5.21	-60334	15.1	0.44	-60598	-60505	9.0	0.15	-60598	-60597	6.4	0.00
6	-60081	-55871	-54801	9.5	8.79	-59997	65.4	0.14	-60029	-60029	12.1	0.09	-60081	-60065	5.8	0.03
7	-59372	-56743	-54127	9.3	8.83	-59369	5.4	0.01	-59372	-59372	5.4	0.00	-59372	-59372	4.0	0.00
8	-74611	-70163	-69301	9.5	7.12	-74611	23.1	0.00	-74611	-74581	12.3	0.04	-74611	-74611	5.0	0.00
9	-61860	-57035	-55745	9.3	9.89	-61860	4.0	0.00	-61860	-61860	8.9	0.00	-61860	-61860	3.8	0.00
10	-72805	-70061	-68821	9.4	5.47	-72805	35.2	0.00	-72805	-72805	9.2	0.00	-72805	-72805	5.0	0.00
Avg	-64833.6	-60992.5	-59766.5	9.4	7.88	-64784.8	30.7	0.08	-64828.4	-64809.6	9.5	0.04	-64833.6	-64831.2	5.5	0.00

Table 14 Results for instances with 100 wells, 10 rigs and $H = 300$

Inst.	BKUB	VNS				BPC			ALNS				HGA			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-111649	-105263	-102799	7.8	7.93	-111649	9.7	0.00	-111649	-111649	7.7	0.00	-111649	-111649	3.7	0.00
2	-97802	-91016	-89987	7.8	7.99	-97789	12.5	0.01	-97802	-97793	7.2	0.01	-97802	-97800.7	4.4	0.00
3	-100913	-96408	-95002	7.8	5.86	-100898	58.0	0.01	-100913	-100913	7.8	0.00	-100913	-100913	4.5	0.00
4	-96886	-91514	-90444	7.8	6.65	-96886	8.9	0.00	-96886	-96871	7.7	0.02	-96886	-96886	4.5	0.00
5	-102631	-99087	-98146	7.8	4.37	-102631	5.7	0.00	-102631	-102626	7.4	0.00	-102631	-102631	3.9	0.00
6	**-89949**	-83927	-82181	8.0	8.64	-89948	67.5	0.00	**-89949**	-89949	8.4	0.00	**-89949**	-89949	4.5	0.00
7	-96842	-92404	-90208	7.7	6.85	-96737	7.1	0.11	-96842	-96804	6.0	0.04	-96842	-96842	4.0	0.00
8	-108731	-101492	-100140	7.8	7.90	-108613	37.8	0.11	-108731	-108715	7.3	0.01	-108731	-108731	4.5	0.00
9	-98843	-93986	-93154	7.7	5.76	-98843	25.5	0.00	-98843	-98702	8.0	0.14	-98843	-98821.8	5.1	0.02
10	-109304	-103685	-101881	7.8	6.79	-109304	6.6	0.00	-109304	-109299	7.7	0.00	-109304	-109304	3.9	0.00
Avg	-101355.0	-95878.2	-94394.2	7.8	6.87	-101329.8	23.9	0.02	-101355.0	-101332.1	7.5	0.02	-101355.0	-101352.8	4.3	0.00

Table 15 Results for instances with 200 wells, 5 rigs and $H = 200$

Inst.	**BKUB**	**VNS**				**BPC**			**ALNS**				**HGA**			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-40257	-38214	-36976	25.7	8.15	-40257	60.8	0.00	-40257	-40174	26.0	0.21	-40257	-40257	26.6	0.00
2	-35084	-29801	-29437	25.0	16.10	-35084	2.6	0.00	-35084	-34769	20.2	0.90	-35084	-35084	35.9	0.00
3	-40195	-38295	-37286	25.9	7.24	-40195	19.7	0.00	-40131	-40122	29.9	0.18	-40195	-40195	28.3	0.00
4	-40523	-38054	-37260	25.7	8.05	-40492	4.6	0.08	-40519	-40455	26.9	0.17	-40523	-40523	24.5	0.00
5	-40194	-36328	-35396	25.6	11.94	-40194	5.2	0.00	-40194	-40187	24.5	0.02	-40194	-40194	24.2	0.00
6	-42335	-39977	-39617	25.8	6.42	-42335	12.7	0.00	-42322	-42317	35.4	0.04	-42335	-42320	25.8	0.03
7	-33070	-31559	-30749	25.5	7.02	-33070	26.2	0.00	-33070	-33062	20.2	0.02	-33070	-33063	24.8	0.02
8	-39517	-36798	-36409	26.0	7.86	-39517	3.4	0.00	-39517	-39229	30.6	0.73	-39517	-39517	25.4	0.00
9	-45683	-42140	-41522	25.7	9.11	-45683	3.7	0.00	-45683	-45647	27.7	0.08	-45683	-45664	23.2	0.04
10	-38036	-36159	-35243	25.8	7.34	-38036	1.9	0.00	-38036	-38036	25.3	0.00	-38036	-38036	25.9	0.00
Avg	-39489.4	-36732.5	-35989.5	25.7	8.92	-39486.3	14.1	0.01	-39481.3	-39399.8	26.6	0.23	-39489.4	-39485.4	26.5	0.01

Table 16 Results for instances with 200 wells, 10 rigs and $H = 200$

Inst.	**BKUB**	**VNS**				**BPC**			**ALNS**				**HGA**			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-67670	-61802	-60404	51.8	10.74	-67670	6.1	0.00	-67670	-67455	50.3	0.32	-67670	-67670	26.4	0.00
2	-61377	-55339	-54265	51.3	11.59	-61377	26.2	0.00	-61366	-61319	45.3	0.09	-61377	-61376	22.4	0.00
3	-66032	-59862	-59051	51.7	10.57	-66032	25.6	0.00	-65968	-65772	56.9	0.39	-66032	-65942	28.2	0.14
4	-67445	-61184	-60407	52.2	10.44	-67399	69.8	0.07	-67445	-67299	52.8	0.22	-67445	-67436	29.4	0.01
5	-72658	-68561	-67471	52.2	7.14	-72658	18.1	0.00	-72658	-72658	52.4	0.00	-72658	-72658	20.9	0.00
6	**-67122**	-61986	-61603	52.0	8.22	**-67122**	55.5	0.00	-67111	-66654	55.7	0.70	**-67122**	-67119	31.6	0.00
7	-60036	-54355	-52925	51.5	11.84	-60036	7.7	0.00	-60036	-59905	39.7	0.22	-60036	-60036	23.8	0.00
8	-60350	-57002	-55935	51.9	7.32	-60350	22.6	0.00	-60350	-60316	54.2	0.06	-60350	-60350	27.0	0.00
9	-79301	-72789	-71915	52.6	9.31	-79301	24.2	0.00	-79301	-79172	62.0	0.16	-79301	-79301	29.0	0.00
10	-61968	-55564	-54712	51.8	11.71	-61823	22.7	0.23	-61968	-61951	54.8	0.03	-61968	-61968	28.0	0.00
Avg	-66395.9	-60844.4	-59868.8	51.9	9.89	-66376.8	27.9	0.03	-66387.3	-66250.1	52.4	0.22	-66395.9	-66385.5	26.7	0.02

Table 17 Results for instances with 200 wells, 5 rigs and $H = 300$

Inst.	**BKUB**	**VNS**				**BPC**			**ALNS**				**HGA**			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	-90317	-82385	-81650	44.5	9.60	-90317	41.1	0.00	-90301	-89678	44.6	0.71	-90317	-90317	26.5	0.00
2	-83684	-71222	-68499	44.1	18.15	-83684	28.4	0.00	-83640	-83066	35.5	0.74	-83684	-83308	31.5	0.45
3	-91981	-80689	-79531	44.7	13.54	-91981	32.4	0.00	-91806	-91482	50.6	0.54	-91981	-91981	27.6	0.00
4	-91917	-82239	-79066	44.7	13.98	-91911	147.0	0.01	-91901	-91652	45.9	0.29	-91917	-91917	23.7	0.00
5	-91757	-84210	-81643	44.6	11.02	-91757	296.9	0.00	-91757	-91434	42.3	0.35	-91757	-91757	24.2	0.00
6	−**96136**	-84740	-82257	44.8	14.44	−**96136**	1185.3	0.00	-95941	-95329	60.5	0.84	−**96136**	-96056	29.4	0.08
7	-76049	-68103	-67107	44.3	11.76	-76049	718.7	0.00	-76046	-76026	32.9	0.03	-76049	-75984	27.7	0.09
8	-89421	-80513	-79293	44.5	11.33	-89421	177.6	0.00	-89421	-89355	53.5	0.07	-89421	-89421	20.7	0.00
9	-99962	-89006	-87665	45.1	12.30	-99953	109.9	0.01	-99459	-98643	46.3	1.32	-99962	-99962	24.8	0.00
10	-86564	-79675	-78303	43.8	9.54	-86564	229.8	0.00	-86431	-86332	42.1	0.27	-86564	-86562	30.7	0.00
Avg	-89778.8	-80278.2	-78501.4	44.5	12.56	-89777.3	296.7	0.00	-89670.3	-89299.7	45.4	0.52	-89778.8	-89726.5	26.7	0.06

Table 18 Results for instances with 200 wells, 10 rigs and $H = 300$

Inst.	**BKUB**	**VNS**				**BPC**			**ALNS**				**HGA**			
		Best	Avg	Time (s)	Dev (%)	Best	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)	Best	Avg	Time (s)	Dev (%)
1	**-148427**	-130721	-128608	68.6	13.35	-148346	706.0	0.05	**-148427**	-147902	65.0	0.35	**-148427**	-148380.6	19.5	0.03
2	**-138040**	-124751	-122179	67.9	11.49	-137013	571.7	0.74	-137923	-137794	54.3	0.18	**-138040**	-137996.9	23.8	0.03
3	**-149890**	-132437	-130005	67.2	13.27	-149278	1016.0	0.41	-149651	-148910	73.8	0.65	**-149890**	-149652.5	22.2	0.16
4	**-151996**	-135637	-134469	67.3	11.53	**-151996**	400.5	0.00	**-151996**	-151477	69.5	0.34	**-151996**	-151912.4	20.8	0.06
5	**-159150**	-143746	-140367	67.7	11.80	-158972	436.1	0.11	-159020	-158813	63.7	0.21	**-159150**	-159150	20.5	0.00
6	**-153497**	-138154	-134175	67.2	12.59	-153496	700.4	0.00	-153496	-152108	71.9	0.90	**-153497**	-153497	20.1	0.00
7	**-134005**	-119723	-117532	67.9	12.29	-133930	393.1	0.06	**-134005**	-133648	44.5	0.27	**-134005**	-133990.3	19.1	0.01
8	**-139587**	-122282	-120769	67.4	13.48	-137883	995.4	1.22	-139112	-138815	70.1	0.55	**-139587**	-139424.9	22.9	0.12
9	**-169910**	-148248	-145555	68.3	14.33	**-169910**	2272.6	0.00	-169543	-169091	75.6	0.48	**-169910**	-169686.3	19.9	0.13
10	**-143778**	-124489	-122783	67.1	14.60	-143554	664.4	0.16	-143305	-142783	73.6	0.69	**-143778**	-143649.9	22.5	0.09
Avg	-148828.0	-132018.8	-129644.2	67.6	12.87	-148437.8	815.6	0.28	-148647.8	-148134.1	66.2	0.46	-148828.0	-148734.1	21.1	0.06