



An Efficient Matheuristic for the Inventory Routing Problem

Pedro Diniz¹, Rafael Martinelli²(✉), and Marcus Poggi¹

¹ Departamento de Informática, PUC-Rio, Rio de Janeiro, Brazil
{pfonseca,poggi}@inf.puc-rio.br

² Departamento de Engenharia Industrial, PUC-Rio, Rio de Janeiro, Brazil
martinelli@puc-rio.br

Abstract. We consider the general multi-vehicle and multi-period Inventory Routing Problem (IRP). A challenging aspect of solving IRPs is how to capture the relationship among the periods where the routing takes place. Once the routes are defined, computing the optimal inventory at each customer on each period amounts to solving a network flow problem. We investigate the impact of efficiently solving this recurring network problem on the solutions found by the devised algorithm. A very significant impact is observed when solving 638 instances in a classical benchmark set, improving 113 upper bounds through assembling the network optimization into an ILS-RVND algorithm. In particular, the results suggested this approach performs better for larger instances with more periods, obtaining speed-ups of about ten times. A detailed comparison against nine of the most prominent exact and heuristic methods favors the proposed approach.

Keywords: Inventory routing Problem · Vendor-Managed Inventory · Matheuristics · Network Simplex · Iterated Local Search

1 Introduction

Logistics decisions are recognized for having a significant impact on every organization's strategic planning. With the advent of the internet and the ever-increasing globalization, this impact is growing and fostering companies in a continuous search to reduce costs and increase logistics efficiency. Global optimization of the supply chain is one of the efforts that have been employed in the late years and is gaining in popularity due to improved results when compared to traditional models. Vendor-Managed Inventory (VMI) systems are a big step towards this objective. It centralizes the decisions on the suppliers allowing them to reduce both production and distribution costs at the same time by combining and coordinating transportation and demand for multiple customers. The Inventory Routing Problem [10] is then an application of VMI to define routes to one or more vehicles to service a set of customers during a planning horizon. In every period, all customers' demands should be met with products from the

limited customers' inventories and/or from the production of the single depot, where all vehicles are located. Therefore, besides the customers' visits to each customer, the amount of product delivered must also be decided for each vehicle and period. The problem then asks for the minimum overall cost, considering the vehicles' routing costs, and inventory costs from the customers and depot.

This problem models three simultaneous supply chain decisions into one single (global) optimization problem: (i) decide the periods in which a customer should be visited, (ii) decide how much should be delivered for each customer, and (iii) decide the best delivery routes. Considering also variations of the problem, usually, one out of these two inventory policies are applied regarding customer visits: (i) order-up-to level policy (OU), where customer inventory is filled to its maximum capacity at every visit; and (ii) maximum level policy (ML), where customer inventory may be below but never above its limit at every visit.

The IRP is gaining much attention recently due to its importance, complexity, many variations, and the lack of an exact algorithm capable of solving instances of reasonable size in a short time. Still, considering it belongs to the class of the NP-hard problems, which can be proved by a reduction to the Traveling Salesman Problem, advances in the literature over the last ten years are very promising. Over the years, different methods were proposed for the single and, more recently, multi-vehicle versions of the problem. For the single-vehicle version, heuristic methods include a two-step heuristic algorithm [9], an Adaptive Large Neighborhood Search (ALNS) [13], a hybrid tabu search method [4], Simulated Annealing [2,20] and Iterated Local Search (ILS) [2,20]. The first exact method for the single-vehicle IRP was a branch-and-cut algorithm [5]. Methods for the multi-vehicle version are more recent but are increasing in a fast-pace. Heuristic methods include an ALNS [12], a hybrid matheuristic [7], a kernel search matheuristic from [18], an ILS [22], a Simulated Annealing [2] and unified matheuristic [11]. Exact methods for this variant include branch-and-cut [1,14], a branch-cut-and-price [16] and, more recently, a single-period cutting planes [8].

This paper considers a class of the multi-vehicle and multi-period Inventory Routing (IRP). The main result is a matheuristic composed of an Iterated Local Search, with Random Variable Neighborhood Descent, that explores a modification on a Network Flow algorithm to efficiently find the optimal inventory flow and costs, given the routes to be performed in each period. The basic idea was already explored in [12], where the authors report the computation time to solve the Network Flow problem as an issue. It limited the approach performance significantly in terms of solution quality versus computation time. In the resolution of IRPs, capturing the distribution of goods relation over the periods is a challenge. This research shows that efficiently solving this Network Flow problem pays off. A very significant impact is observed. When solving 638 instances in the small benchmark set from [5], for the ML policy, we improve 113 upper bounds through plugging the efficient network optimization into an ILS-RVND algorithm. In particular, the results suggest this approach performs better on larger instances, with more periods, customers, and vehicles. The proposed algorithm achieves about ten times faster execution time by updating and reusing

the underlying structure used by the Network Simplex algorithm. As the final algorithm intensively uses an enhanced Network Simplex method, we classify this heuristic algorithm as a *Matheuristic*.

This paper is organized as follows. The next section presents the metaheuristic used. Section 3 explains the changes in the Network Simplex algorithm to speed up the solution. In Sect. 4, we show the computational experiments and analyses. Section 5 concludes the work and lists future research.

2 Iterated Local Search

We start by describing the basic Iterated Local Search (ILS) algorithm. We address the fundamental constructs of the heuristic and the main components that build up the algorithm: the neighborhoods and the local search. The search-space used during the algorithm execution comprehends both feasible and infeasible solutions. A collection of neighborhoods is defined to explore this search-space. These neighborhoods are commonly used on several classes of vehicle routing problems, except the insert and remove neighborhoods, which are particular for the IRP since a customer may or may not be visited in a given period of the planning horizon. The neighborhoods are the following:

- **insert**(c, v, p): Insert customer c into route v in period p .
- **remove**(c, v, p): Remove customer c from route v in period p .
- **relocate**($c_1, v_1, p_1, c_2, v_2, p_2$): Remove c_1 from route v_1 and period p_1 , and insert into route v_2 in period p_2 before customer c_2 .
- **swap**(v, c_1, c_2): Swap customer at position c_1 with customer at position c_2 in rout v .
- **shift**(v, c, k): Move customer at position c to position $c + k$ in route v .
- **reverse-subtour**(s): Reverse the subtour s . This movement is equivalent to the well-known 2-opt move.

The ILS uses a Randomized Variable Neighborhood Descent (RVND) [20], which chooses, at each iteration, among the above neighborhoods. Given the current solution, represented by a set of customers tours for each vehicle and period, it randomly selects a neighborhood and performs a local search using the best improvement strategy. The new solution cost is then obtained by the sum of the routing and inventory costs.

The routing cost can be calculated in constant time, as for most of VRPs, given the changes performed by a move on the routes. Regarding the inventory, the modified routes may impose different deliveries and may forbid fulfilling all customers' demands. Therefore, changes in the amount delivered for each customer and period may be required. Our algorithm finds the optimal inventory or detects no feasible inventory exists by solving a Network Flow problem. The inventory cost either corresponds to the optimal inventory cost or a penalty cost in case of infeasibility. The next section presents the Network Flow problem that determines the optimal delivery amounts.

An outline of the ILS is presented in Algorithm 1. It starts with an empty initial solution as the current solution, i.e., no customer is visited in any period by any vehicle. This solution is infeasible, therefore penalized inventory costs will be associated. Then, in each iteration, the current solution is perturbed by applying *max_perturb* random moves followed by a complete run of the local search. During the perturbation, after each random move, the new solution is accepted if it is an improving solution or if it passes an acceptance criterion.

Algorithm 1. Outline of the Iterated Local Search

```

1: function ILS
2:   sol ← localSearch(empty_sol)
3:   best ← cur ← sol
4:   no_imp ← 0
5:   for i ← 1 to max_iter do
6:     for j ← 1 to max_perturb do
7:       new ← randMove(sol)
8:       if new.cost < sol.cost or testAccept(new, sol) then
9:         sol ← new
10:      end if
11:    end for
12:    sol ← localSearch(sol)
13:    if sol.cost < cur.cost or testAccept(sol, cur) then
14:      cur ← sol
15:      if cur.cost < best.cost then
16:        best ← cur
17:        no_imp ← 0
18:      end if
19:    else
20:      sol ← cur
21:    end if
22:    no_imp ← no_imp + 1
23:    if no_imp > max_no_imp or sol.cost > (1 + max_perc)*best.cost then
24:      sol ← cur ← best
25:    end if
26:  end for
27: end function

```

After the local search, this resulted solution is tested against the current best solution. It will be accepted if it is an improving solution or if it passes the acceptance criterion. If accepted, it will be tested for improvement against the global best solution and replace it in case of improvement. If no improvement was observed for a given number of iterations or the current solution value is higher than a given percentage of the global best, the current solution and current best solution are replaced with the global best solution.

The acceptance criterion is implemented based on the solution value obtained after the first call to the local search. In the first iteration, it will accept a solution

20% worse with a probability of 50%. This chance is then further decreased based on the number of iterations *max_iter* to accept a solution 10% worse with a probability of 10% on the last iteration [21].

3 Network Simplex

While the reduction to Minimum Cost Flow [19] is not something new to the literature, other studies that tried this same decomposition [12] reported significant running times where approximately 65% of the total time was spent solving only the subproblem. Our approach achieves, on average, ten times faster running times by reusing the underlying Network Simplex structure. Every optimal inventory is obtained starting the Network Simplex algorithm from the optimal solution of the previous iteration. The model and the update procedure are now described.

3.1 Formulation

The subproblem is defined on a directed acyclic graph $G = (V, A)$. In this graph, the vertex set is composed of $|T|$ copies of the supplier, customers, and vehicles, plus an artificial vertex representing the excess of product that may exist at the end of the time horizon for some customer or the supplier. Every supplier vertex in this graph is a source of flow equal to its production on the given period. Similarly, every customer demands a flow equal to its demand in the given period. The artificial vertex demands the difference between the total supply and the total demand.

The arc set has four different types of arcs. The first type contains arcs from the supplier vertex to vehicle vertices. The capacity of each arc is the vehicle's capacity, and the cost is zero. The second type contains arcs from the vehicle vertices to customers vertices. Each arc has unlimited capacity and cost zero. The third type contains arcs between consecutive periods for the supplier and the customers. They represent the inventory that may wait from one period to the next. Their cost and capacity are the inventory cost and capacity of each customer or supply. Finally, the last type contains the arcs from the customers or supplier in the last period to the excess vertex. For each arc, its capacity is the supplier or customer inventory capacity, but in this case, its cost is zero. An example of this graph, with three customers, two vehicles, and two periods, is presented in Fig. 1.

If a vehicle visits a customer in a given period, we keep the original cost of the corresponding vehicle-customer arc. On the other hand, if the vehicle does not visit the customer in the period, we set the arc cost to infinity. This approach is more efficient than to rebuild graph G or to remove and add arcs before every call to the Network Flow algorithm.

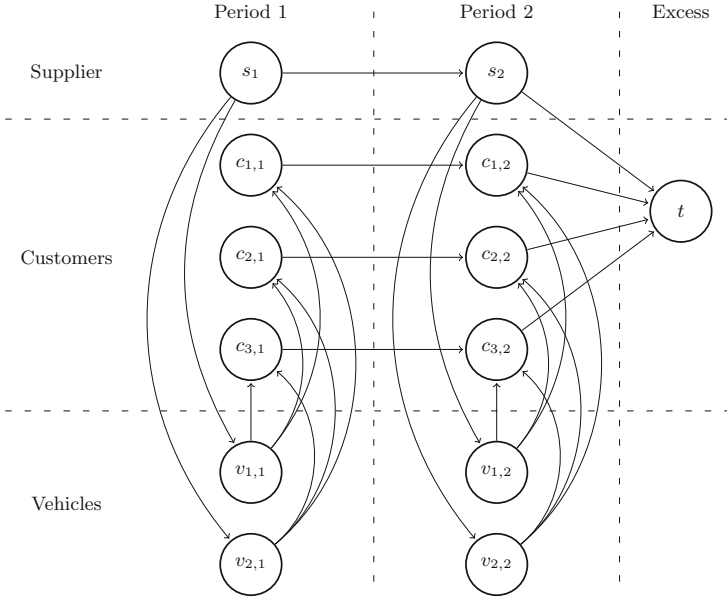


Fig. 1. Proposed network flow model for 2 customers, vehicles and periods.

3.2 Fast Flow Calculation

Once the graph is built, we use the Network Simplex algorithm to calculate the total inventory cost for a given matheuristic solution. During the local search, several move evaluations are performed in sequence by simple changes in the previous solution. By analyzing the neighborhoods presented in Sect. 2, it is possible to control the impact of each one in the Network Flow graph. For example, when an insert is evaluated, we have to add an arc from the corresponding vehicle to the customer (i.e., setting its cost to zero). For a remove evaluation, we have to remove the vehicle-customer arc (i.e., setting its cost to infinity). The modifications required by all vehicle moves can be represented by changes in costs on arcs.

The Network Simplex works using a spanning tree structure. It defines three sets, T , L and U . T contains the set of arcs that compose the spanning tree, i.e., the basic variables. L contains the set of arcs where the flow is zero, and U contains the set of arcs where flow equals the edge capacity. Both sets contain the non-basic variables. A Network Simplex iteration starts by calculating the reduced cost of every arc in L and U based on the dual information of the vertices, represented by the dual variables π_i . The reduced cost is then calculated as $\bar{c}_{ij} = c_{ij} + \pi_i - \pi_j$. If any non-basic arc has a negative reduced cost, the procedure moves this arc to set T , and removes another arc from T following the Simplex rules. At the end of execution, the optimal solution is found by joining the arc sets with positive flow, i.e., the arc sets T and U . We refer the reader to [15] for a complete description of the method.

Usually, any change in the solution would require executing the entire algorithm again because (i) some arcs may have been introduced (or removed), (ii) arcs' capacities may have changed, (iii) node supplies may have been modified, (iv) the maximum flow may have changed, and (v) arcs costs may have changed. Taking advantage of the IRP structure, the model we propose addresses most of these issues, and reduces changes to be the equivalent of deciding which arcs must be moved between T , L and U . In this model, change (i) does not happen because arcs are never introduced or removed, only their costs can change between zero or infinity. Changes (ii) and (iii) are also not present because the vehicle and inventory constraints are the same, and the production and demands values remain unchanged. All previous reasons implicate that change (iv) cannot occur too. The only change left is (v), and we show how to deal with it.

In Network Simplex, the arc cost is used to calculate the arc's dual cost and the dual variables π_i of the nodes below this arc if it is in T . These changes are sufficient to avoid having to execute the entire Network Simplex again and let the algorithm continue from where it stopped on the previous solution. The dual cost of each edge can be recalculated while verifying if the edge is a candidate to enter the tree. Otherwise, there is no need to check its reduced cost. Updating node potentials, on the other hand, requires more effort because they must be updated on a specific order. While other strategies may be used, we propose a procedure that iterates over all nodes starting from the root and continues node by node following the order used for tree construction (commonly referred to as the "thread order"). Algorithm 2 illustrates the procedure.

Algorithm 2. Update π values

```

1: function UPDATE_PI( )
2:   node  $\leftarrow$  thread[root]
3:   while node  $\neq$  root do
4:     arc  $\leftarrow$  pred[node]
5:     tgt  $\leftarrow$  arc.target
6:     src  $\leftarrow$  arc.source
7:     if tgt = node then
8:        $\pi$ [node]  $\leftarrow$   $\pi$ [src] + cost[arc]
9:     else
10:       $\pi$ [node]  $\leftarrow$   $\pi$ [tgt] - cost[arc]
11:    end if
12:  end while
13: end function

```

4 Computational Experiments

The NSIRP was implemented in C++ on Ubuntu Linux. For the inventory cost calculation (Minimum Cost Flow), we modified and used the Network Simplex

algorithm from the LEMON C++ library [17], adding the proposed modifications. Computational experiments were performed with a single thread on an Intel Core i7-8700K 3.7 GHz with 64 GB RAM. The algorithm was tested over the benchmark instances proposed in [5]. They are composed of 160 files organized into two classes of instances (*low* and *high* inventory cost) and cover scenarios where the horizon H can be equal to 3 or 6 periods. The number of clients n is $n = 5t$, with $t = 1, \dots, 10$ when $H = 3$ and $t = 1, \dots, 6$ when $H = 6$. All instances were tested from two up to five identical vehicles, dividing the vehicle capacity by the number of vehicles (and rounding down), and resulting in a total of 640 instances (but two are known to be infeasible). For each instance, we run the matheuristic algorithm ten times.

We first present the results on our main contribution, the Fast Flow Calculation. We performed a complete run on all instances with the regular Network Simplex algorithm (NSA) and with the Fast Flow Calculation (FFC). Figure 2 shows the average fraction of the original time that our modification obtains, calculated for each instance as ub_{FFC}/ub_{NSA} , for each of the four types of instance while the number of vehicle grows: low inventory cost with three periods (L3), high inventory cost with three periods (H3), low inventory cost with six periods (L6) and high inventory cost with six periods (H6). The smallest improvement was found for instances with two vehicles, three periods, five customers, and high cost. Our method runs in 21.1% of the original time. From the figure, we can notice a clear tendency. As the number of vehicles or the number of customers, or even the number of periods goes up, the improvement is more significant. The best improvement is when the method obtains a running time of 6.0% from the original one. It is noteworthy the correlation between the same characteristics, but different cost types (high and low). The results show that they have no impact on the proposed method.

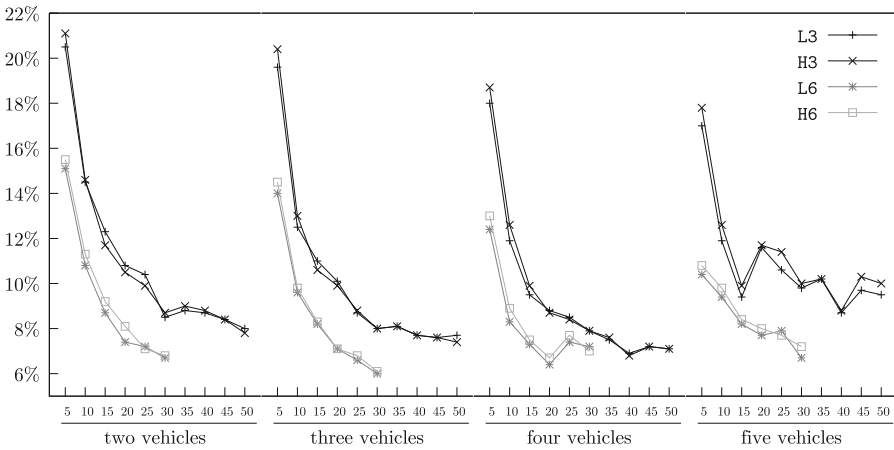


Fig. 2. Time fraction of fast flow, for each size, vehicles, periods and cost type.

Using fast flow, we now compare the results of NSIRP against the best known upper bounds and most prominent methods from the literature comprising nine methods: four exact, three matheuristics, and two metaheuristics. The exact methods are ABIS (Branch-and-Bound) from Archetti et al. (2007) [6], CL (Branch-and-Cut) from Coelho and Laporte (2014) [14], DRC (Branch-Cut-and-Price) from Desaulniers et al. (2016) [16], and ABW (Branch-and-Cut) from Avella et al. (2018) [8]. The matheuristic methods comprehend ABS Archetti et al. (2017) [7], the unified decomposition CCJ from Chitsaz et al. (2019) [11], and a kernel search AGMS from Archetti et al. (2019) [3, 18]. The metaheuristics are an Iterated Local Search SOSG from Santos et al. (2016) [22], and a Simulated Annealing AMM from Alvarez et al. (2018) [2]. The parameters used in the experiments were $max_iter = 500$, $maxperturb = 15$, $maxnoimp = 25$ and $maxperc = 1.2$.

Figure 3 shows the average gap for each instance type when the number of vehicles grows. Each gap is calculated against the best-known upper bound from the literature (LIT) as $(ub_{FFC} - ub_{LIT})/ub_{LIT}$. It becomes negative as the number of periods goes up with the worst, never exceeding 4.0%. The gap growth for each series was minimal reinforcing the correlation between same characteristics, except for L3. Table 1 shows the detailed results. The overall average gap was 0.77%, 0.01% on the best, and 1.84% on the worst. Of the 638 instances, NSIRP improved or at least matched 463 (72.0%) of the best-known upper bounds, with 113 improvements (51.0% of the 221 open instances) and 350 matches.

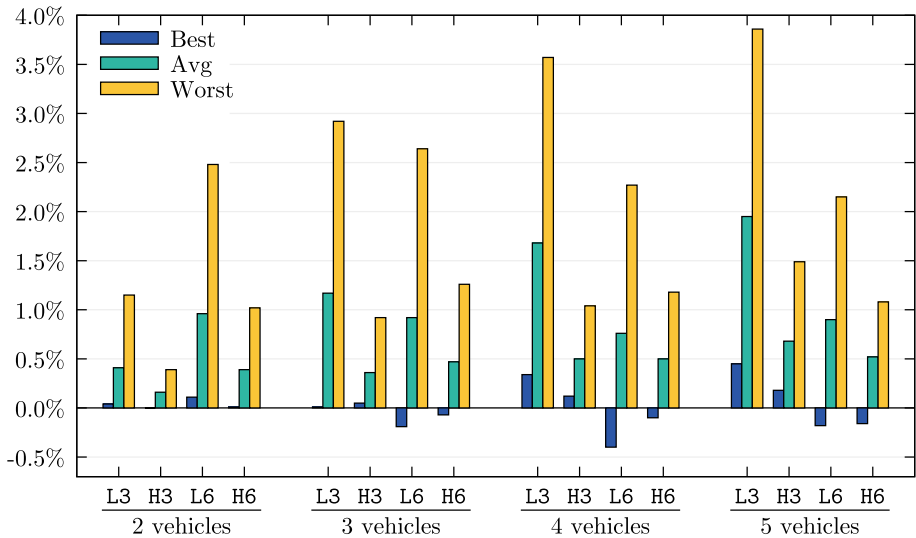


Fig. 3. Average UB gap for each group and number of vehicles.

Table 1. Summary results for classical IRP instances

Type	Veh	Best	Avg	Worst	Time	Better	Equal	Worse
L3	2	0.04%	0.41%	1.15%	47.88	0	45	5
H3	2	0.00%	0.16%	0.39%	58.85	2	41	7
L6	2	0.11%	0.96%	2.48%	75.17	3	18	9
H6	2	0.01%	0.39%	1.02%	89.78	7	14	9
Avg/Total		0.04%	0.48%	1.26%	67.92	12	118	30
L3	3	0.01%	1.17%	2.92%	74.74	2	44	4
H3	3	0.05%	0.36%	0.92%	92.79	3	35	12
L6	3	-0.19%	0.92%	2.64%	107.85	14	11	5
H6	3	-0.07%	0.47%	1.26%	133.40	14	9	7
Avg/Total		-0.05%	0.73%	1.94%	102.20	33	99	28
L3	4	0.34%	1.68%	3.57%	101.56	6	30	14
H3	4	0.12%	0.50%	1.04%	127.87	10	25	15
L6	4	-0.40%	0.76%	2.27%	140.04	15	5	10
H6	4	-0.10%	0.50%	1.18%	176.22	13	6	11
Avg/Total		-0.01%	0.86%	2.01%	136.42	44	66	50
L3	5	0.45%	1.95%	3.86%	129.59	0	31	19
H3	5	0.18%	0.68%	1.49%	165.29	3	26	21
L6	5	-0.18%	0.90%	2.15%	156.41	8	6	15
H6	5	-0.16%	0.52%	1.08%	221.56	13	4	12
Avg/Total		0.07%	1.01%	2.15%	168.21	24	67	67
Overall		0.01%	0.77%	1.84%	118.69	113	350	175

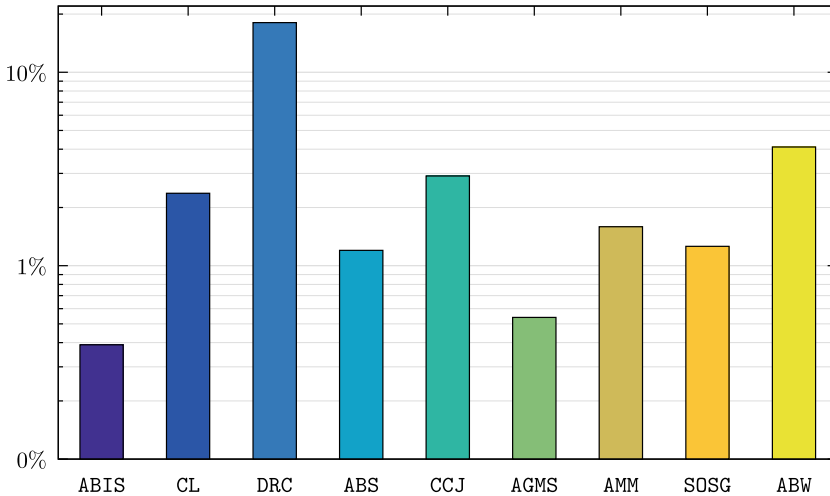


Fig. 4. Gap comparison from each work of the literature.

A best upper bound gap comparison against each method from the literature is presented in Fig. 4. For each method (MET), this gap is calculated against our best upper bound for each instance as $(ub_{MET} - ub_{FFC})/ub_{FFC}$. This comparison comprehends only the instances where the method provided an upper bound. From the figure, we can notice that, on average, NSIRP outperforms all nine methods we compare from the literature.

5 Conclusion

We have defined a new model to solve the multi-vehicle Inventory Routing. We have demonstrated how this model enables the development of inventory-exact solutions with more than ten times faster running times by extending current state-of-the-art Network Simplex implementation and proposing a fast flow procedure. We have implemented an ILS-RVND based matheuristic, entitled NSIRP, to assess the performance of the model on well-known instances from the literature and reported the results.

Computational experiments using well-known instances demonstrated that NSIRP could be widely applied between different instances, with an average upper bound gap of 0.77%. On the 638 tested instances, the method improved or at least matched 463 (72.0%) of the best-known upper bounds, improving 113 (51.0%) of the 221 open instances. Running time results indicates that there is no correlation between fast flow calculation times and the number of vehicles, customers, or periods. It is demonstrated to be a scalable method, well suited for the cases when one of these variables grows. Compared to other methods from the literature NSIRP was capable of decreasing the average upper bound gap for all of them, in the worst case by at least 0.4%.

Considering we are capable of solving the inventory subproblem using an exact algorithm that implements the simplex algorithm with significantly reduced running times, further work would be to use dual information to create more problem-specific neighborhoods. Use this information, such as the nodes π_i or the arcs' reduced costs, to reduce the search space, and navigate between solutions that otherwise would be composed of multiple moves.

Acknowledgements. This research was partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 140084/2017-7, 313521/2017-4, 425962/2016-4 and 311954/2017-0, and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), Financing Code 001. All support is gratefully acknowledged.

References

1. Adulyasak, Y., Cordeau, J.F., Jans, R.: Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS J. Comput.* **26**(1), 103–120 (2013)
2. Alvarez, A., Munari, P., Morabito, R.: Iterated local search and simulated annealing algorithms for the inventory routing problem. *Int. Trans. Oper. Res.* **25**(6), 1785–1809 (2018)
3. Archetti, C., Guastaroba, G., Huerta-Muñoz, D., Speranza, M.: A kernel search heuristic for the multi-vehicle inventory routing problem (2019). <http://or-brescia.unibs.it/instances>
4. Archetti, C., Bertazzi, L., Hertz, A., Speranza, M.G.: A hybrid heuristic for an inventory routing problem. *INFORMS J. Comput.* **24**(1), 101–116 (2012)
5. Archetti, C., Bertazzi, L., Laporte, G., Speranza, M.G.: A branch-and-cut algorithm for a vendor-managed inventory-routing problem. *Transp. Sci.* **41**(3), 382–391 (2007)
6. Archetti, C., Bianchessi, N., Irnich, S., Speranza, M.G.: Formulations for an inventory routing problem. *Int. Trans. Oper. Res.* **21**(3), 353–374 (2014)
7. Archetti, C., Boland, N., Speranza, M.G.: A matheuristic for the multivehicle inventory routing problem. *INFORMS J. Comput.* **29**(3), 377–387 (2017)
8. Avella, P., Boccia, M., Wolsey, L.A.: Single-period cutting planes for inventory routing problems. *Transp. Sci.* **52**(3), 497–508 (2018)
9. Bertazzi, L., Paletta, G., Speranza, M.G.: Deterministic order-up-to level policies in an inventory routing problem. *Transp. Sci.* **36**(1), 119–132 (2002)
10. Campbell, A., Clarke, L., Kleywegt, A., Savelsbergh, M.: The inventory routing problem. In: Crainic, T.G., Laporte, G. (eds.) *Fleet Management and Logistics*. CRT, pp. 95–113. Springer, Boston, MA (1998). https://doi.org/10.1007/978-1-4615-5755-5_4
11. Chitsaz, M., Cordeau, J.F., Jans, R.: A unified decomposition matheuristic for assembly, production, and inventory routing. *INFORMS J. Comput.* **31**(1), 134–152 (2019)
12. Coelho, L.C., Cordeau, J.F., Laporte, G.: Consistency in multi-vehicle inventory-routing. *Transp. Res. Part C: Emerg. Technol.* **24**, 270–287 (2012)
13. Coelho, L.C., Cordeau, J.F., Laporte, G.: The inventory-routing problem with transshipment. *Comput. Oper. Res.* **39**(11), 2537–2548 (2012)
14. Coelho, L.C., Laporte, G.: Improved solutions for inventory-routing problems through valid inequalities and input ordering. *Int. J. Prod. Econ.* **155**, 391–397 (2014)
15. Cunningham, W.H.: A network simplex method. *Math. Program.* **11**(1), 105–116 (1976)
16. Desaulniers, G., Rakke, J.G., Coelho, L.C.: A branch-price-and-cut algorithm for the inventory-routing problem. *Transp. Sci.* **50**(3), 1060–1076 (2015)
17. Dezsó, B., Jüttner, A., Kovács, P.: LEMON - an open source C++ graph template library. *Electron. Notes Theor. Comput. Sci.* **264**(5), 23–45 (2011)
18. Huerta-Muñoz, D., Archetti, C., Guastaroba, G., Speranza, M.: A kernel search for the inventory routing problem (2019). http://redloca.ulpgc.es/images/workshop/2019/Slides_2019/Huerta_Munoz.pdf
19. Orlin, J.B.: A polynomial-time parametric simplex algorithm for the minimum cost network flow problem. Working papers 1484-83. Massachusetts Institute of Technology (MIT), Sloan School of Management (1983)

20. Peres, I.T., Repolho, H.M., Martinelli, R., Monteiro, N.J.: Optimization in inventory-routing problem with planned transshipment: a case study in the retail industry. *Int. J. Prod. Econ.* **193**, 748–756 (2017)
21. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transp. Sci.* **40**(4), 455–472 (2006)
22. Santos, E., Ochi, L.S., Simonetti, L., González, P.H.: A hybrid heuristic based on iterated local search for multivehicle inventory routing problem. *Electron. Notes Discret. Math.* **52**, 197–204 (2016)