Metaheuristic solver for problems with permutative representation

David Woller [1,2,(✉)], Jan Hrazdíra[1], and Miroslav Kulich [1]

[1] Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague
Jugoslávských partyzánů 1580/3, Praha 6, 160 00, Czech Republic
[2] Department of Cybernetics, Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13, Praha 2, 121 35, Czech Republic
wolledav@cvut.cz

Abstract. Today, a large proportion of combinatorial optimization problems can be efficiently formulated as a mixed-integer program and solved with an exact solver. However, exact solvers do not scale well and thus custom metaheuristic algorithms are being designed to provide better scalability at the cost of no optimality guarantees and time-consuming development. This paper proposes a novel formalism for a large class of problems with permutative representation, together with a metaheuristic solver addressing these problems. This approach combines the advantages of both exact and metaheuristic solvers: straightforward problem formulation, scalability, low design time, and ability to find high quality solutions. Three different problems are formulated in the proposed formalism and solved with the proposed solver. The solver is benchmarked against the Gurobi Optimizer and significantly outperforms it in experiments with a fixed computational budget.

Keywords: Metaheuristics, Iterated Local Search, Variable Neighborhood Search, Permutative Representation

1 Introduction

Despite its relatively short history, combinatorial optimization is now a mature field with robust theoretical foundations and a broad portfolio of powerful methods. New and challenging applications are constantly emerging in a wide range of diverse fields, such as artificial intelligence, machine learning, supply chain management, or financial engineering. However, the underlying optimization problems are often intractable. Therefore, selecting the most suitable approach typically requires a trade-off between some of several criteria: optimality or approximation guarantee, design time, runtime, scalability, and versatility.

When the solution optimality is required, the available design time is limited, and the instances are of moderate size, the most suitable choice is to use Integer Programming (IP). Today, multiple solvers are available, such as Gurobi,

CPLEX, or Xpress, that provide highly efficient implementations of state of the art exact algorithms. The only user requirement is to formalize the problem and create a sensible model, which makes these solvers very attractive and widely used in practice. However, an explicit IP formulation of a high-dimensional problem may result in an immensely large model, which can be difficult to work with because of memory limitations. Furthermore, the computational complexity of an exact solver is inherently exponential when solving an $\mathcal{NP}$-hard problem, making it intractable. For these reasons, scalability remains a major limitation of problem-nonspecific IP solvers.

Computationally challenging applications are often tackled by metaheuristics, high-level algorithmic frameworks that can be adapted to problem-specific algorithms. Metaheuristic algorithms typically do not provide any guarantees about the quality of the solution, but are frequently used in applications where no other approach is computationally feasible. Most applications of metaheuristics require considerable design time and the implementations are not versatile. The reason for this is that essential components, such as repair heuristics, destroy heuristics, or local search operators, need to be tailored to a particular problem.

This paper presents a generic metaheuristic solver that is capable of solving a set of problems that share the same solution representation. The main goal is to combine the scalability of custom metaheuristic algorithms with the versatility of modeling paradigms such as IP. The representation of interest encodes a solution as an ordered sequence of nodes which can have arbitrary length and frequency of individual nodes. Any additional constraints and specifics are incorporated into an aggregated fitness function through penalties. This representation allows for the addressing of a broad portfolio of routing, scheduling, or sequencing problems. The proposed solver is benchmarked against the Gurobi optimizer on three $\mathcal{NP}$-hard problems. The main contributions of this paper are:

- proposing a unifying formalism for a wide class of permutative problems,
- proposing a generic solver working with the proposed formalism,
- providing a fair and extensive benchmark against the Gurobi optimizer.

The remainder of this paper is structured as follows. Section 2 discusses related works. Section 3.1 provides the definition of the proposed formalism, as well as the definitions of three classical problems in this formalism. The proposed solver is described in Section 3.2. The experimental results are presented in Section 4 and the conclusions in Section 5.

2 Related Works

First, several metaheuristic solvers capable of addressing a wider portfolio of related problems exist. For example, the LKH3 solver [9], is able to solve a large number of variants of Vehicle Routing Problem, Sequential Ordering Problem, Travelling Repairman Problem, Travelling Salesman Problem, and others by

transforming these problems into a standard TSP. Another example is the Unified Hybrid Genetic Search for Multiattribute Vehicle Routing Problems [21], addressing a broad range of VRP problems. Other solvers are designed as black-box solvers for any problem with a solution representable by a fixed-length permutation. These are, for example, the genetic algorithm (GA) combined with Branch & Bound technique proposed in [14], or the Bayesian optimization approach in [5]. However, the solution representation used in this paper is more general as it allows the solution sequence to have arbitrary length and node frequency.

Second, various metaheuristic frameworks exist. The purpose of these frameworks is to provide problem-independent building blocks that can be used to efficiently assemble a specialized solver. The common drawback is that the use of a custom solution representation typically requires the implementation of custom operators. According to a recent survey [17], most widely used frameworks are the Evolutionary Computation Research System [18] and the ParadiseEO framework [6]. Both implement a large number of mostly evolutionary algorithms and support basic representations such as integer vectors, binary vectors, or fixed-length permutations. Some frameworks are more specialized. For example, the MOEA framework [8] focuses on multi-objective evolutionary algorithms, and the JAMES framework [4] specializes in local search metaheuristics.

Third, the proposed solver contains a plethora of alternative heuristics, operators, and parameters that should be chosen appropriately for the problem at hand. Therefore, the automated design and configuration of heuristic algorithms is also a relevant domain. For this purpose, an external heuristic tuning algorithm is often used [19]. Some successful approaches are: the ParamILS tool [2] based on the Iterated Local Search metaheuristic [13], the Sequential Model-Based Algorithm Configuration tool [10], or the Iterated Racing tool [12], which was actually used for configuration of the proposed solver.

3 Methodology

This section details the main contributions. The common formalism used by the proposed solver is introduced in Section 3.1 and its usage is demonstrated on three problems: Capacitated Vehicle Routing Problem (CVRP), Non-Permutation Flowshop Scheduling Problem (NPFS) and Quadratic Assignment Problem (QAP). The proposed metaheuristic solver is described in Section 3.2.

3.1 Problem definitions

We newly propose the following generic problem definition that serves as an interface between the proposed solver and the specific problem to be solved. It defines only a few properties common to a large number of problems with permutative representation, which are known to the solver. Problem-specific properties are intended to be translated into penalty functions, which are then aggregated with the problem fitness function.

Generic problem definition is given as follows. Let A be a set of n unique nodes (Eq. 1), which can be present in a solution vector X of length m (Eq. 5). A node $a_i \in A$ can be present in X multiple times and the number of its occurrences is given by frequency f_i (Eq. 6-7). Here, the logical expression $[\![.]\!]$ evaluates to 1 if true, 0 otherwise. Each element $a_i \in A$ has defined lower and upper bounds l_i, u_i in the vectors L, U (Eq. 2-3), which limit the frequency of a_i (Eq. 8). The goal is to minimize the augmented fitness function $g(X)$ (Eq. 4).

Given set of nodes	$A = \{a_1, a_2, ..., a_n\},$	$A \subset \mathbb{N}$	(1)
and vectors of bounds	$L = [l_1, l_2, ..., l_n],$	$L \in \mathbb{N}_0^n$	(2)
	$U = [u_1, u_2, ..., u_n],$	$U \in \mathbb{N}_0^n$	(3)
minimize fitness function	$g(X) : A^m \to \mathbb{R}$		(4)
where	$X = [x_1, x_2, ..., x_m],$	$X \in A^m$	(5)
	$F = [f_1, f_2, ..., f_n],$	$F \in \mathbb{N}_0^n$	(6)
	$\forall a_i \in A : f_i = \sum_{j=1}^{m} [\![x_j = a_i]\!]$		(7)
	$\forall a_i \in A : l_i \leq f_i \leq u_i$		(8)

Note that X can have variable length if $L \neq U$. Also, the fitness function $g(X)$ can be completely arbitrary. In the following applications, it is expressed as $g(X) = \hat{g}(X) + p(X)$, where $\hat{g}(X)$ is the actual fitness function of the problem currently solved and $p(X)$ is a penalty function enforcing additional problem-specific properties in X.

Capacitated Vehicle Routing Problem (CVRP) is a classic routing problem. The CVRP's objective is to serve a set of customers with a fleet of k vehicles, while satisfying the demands of customers on cargo and respecting the limited load capacity of the vehicles. An equivalent interpretation used here is that a single vehicle is required to carry out at most k trips, while reloading the cargo at the central depot. The CVRP can be defined in the proposed formalism as follows.

$$D = a_1 \tag{9}$$
$$L = [2, 1, ..., 1] \tag{10}$$
$$U = [k + 1, 1, ..., 1] \tag{11}$$
$$\hat{g}(X) = \sum_{i=1}^{m-1} cost(x_i, x_{i+1}) \tag{12}$$
$$p(X) = M([\![x_1 \neq D]\!] + [\![x_m \neq D]\!]) \tag{13}$$
$$+ M \sum_{i=1}^{m} \max(0, dem(x_i)[\![load(x_i) < dem(x_i)]\!]) \tag{14}$$
$$\text{where } load(x_i) = \begin{cases} 0, & \text{if } i = 1 \\ Q, & \text{if } x_{i-1} = D \\ load(x_{i-1}) - dem(x_{i-1}), & \text{otherwise} \end{cases} \tag{15}$$

The set of nodes A contains $n-1$ customers to visit and a single depot D, which is fixed at a_1 (Eq. 9). The solution vector X contains individual trips separated by depot visits: $X = [D, x_2, ..., D, ..., x_{m-1}, D]$. The bounds L, U ensure that each customer is visited exactly once and the depot is visited at most $k+1$ times (Eqs.10-11). The CVRP fitness function $\hat{g}(X)$ corresponds to the total distance traveled by the vehicle (Eq. 12), where $cost(x_i, x_{i+1})$ is the distance between the nodes x_i and x_{i+1}. Two additional penalties must be added by the penalty function $p(X)$. The vehicle must start and end at the depot (Eq. 13) and the vehicle must have sufficient cargo load to fully satisfy each visited customer (Eq. 14). Here, $load(x_i)$ is the vehicle cargo load before entering node x_i and $dem(x_i)$ is the demand of node x_i. The vehicle is empty before entering the first node in X and loaded to its maximum load capacity Q in the depot (Eq. 15). Finally, M is a large constant.

Non-Permutation Flowshop Scheduling Problem (NPFS) is a variant of a more common Flowshop Scheduling Problem (FSP). In NPFS, n jobs from A have to be scheduled on k machines. The solution vector $X = [x_1, x_2, ..., x_m] = [\pi_1, \pi_2, ..., \pi_k]$ consists of k permutations π_j of length n, where each of these permutations determines the order of processing of the jobs on the machine $j \in \{1, 2, ..., k\}$. Therefore, $m = nk$. The bounds L, U reflect that each job has to be processed on each machine exactly once (Eqs. 16-17). The order in which individual machines are visited by each job is fixed and is given by the index k of the machine. Unlike in the FSP, the processing order within individual machines can differ in NPFS; therefore, generally $\pi_i \neq \pi_j$. In the following equations, $start(a_i, \pi_j)$ is the current start time of the processing of the job a_i in permutation π_j, $proc(a_i, j)$ is the processing time of the job a_i in machine j, and $f_{a_i}^{\pi_j}$ is the frequency of the node a_i in permutation π_j.

$$L = [k, k, ..., k] \tag{16}$$
$$U = [k, k, ..., k] \tag{17}$$
$$\hat{g}(X) = \max_{a_i \in A}(start(a_i, \pi_k) + proc(a_i, k)) \tag{18}$$
$$p(X) = M \sum_{i=1}^{n} \sum_{j=1}^{k} |1 - f_{a_i}^{\pi_j}| \tag{19}$$
$$+ M \sum_{i=1}^{n} \sum_{j=1}^{k-1} [\![start(a_i, \pi_{j+1}) \geq start(a_i, \pi_j) + proc(a_i, j)]\!] \tag{20}$$

The goal is to minimize the total makespan, i.e., the last job on the k-th machine should finish as early as possible (Eq. 18). Two penalties must be defined in $p(X)$. First, each job has to be scheduled exactly once on each machine (Eq. 19). Second, a job a_i can start to be processed on machine $j+1$ only after it finishes on machine j (Eq. 20).

Quadratic Assignment Problem (QAP) is a generalization of the following facility location problem: n distinct facilities from $\{1, 2, ..., n\}$ must be assigned to n distinct locations from A. Flow $f(i, j)$ is given for all pairs of facilities i, j and distance $d(a_k, a_l)$ is given for all pairs of locations a_k, a_l. The solution vector X contains a single permutation, where x_i determines the location assigned to the facility i. The goal is to minimize the sum of all flows weighted by the corresponding distances according to the assignment X (Eq. 23).

$$L = [1, 1, ..., 1] \tag{21}$$
$$U = [1, 1, ..., 1] \tag{22}$$
$$\hat{g}(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} flow(i, j) dist(x_i, x_j) \tag{23}$$

QAP formulation in the proposed formalism is exceptionally simple and does not require any penalties. By contrast, formulating the problem as MILP is complicated by the non-linearity of the problem, and a linearization technique needs to be employed.

3.2 Proposed solver

The proposed metaheuristic solver addresses the generic problem defined in Section 3.1. The user is required to specify the set of nodes A, vectors L, B containing their bounds and the augmented fitness function $g(X) = \hat{g}(X) + p(X)$. The solver minimizes $g(X)$ over all possible sequences $X \in A^m$ and guarantees that the condition on bounds defined in Eq. 8 is satisfied. All remaining

Double Bridge(X, k) generates k distinct random indices between 1 and m. The solution X is then divided into $k+1$ segments determined by these indices, and all of these segments are reversed.
Random Double Bridge(X, k) operates similarly to Double Bridge. However, each of the $k+1$ segments of X is reversed with a probability of 0.5.
Reinsert(X, k) selects uniformly randomly k distinct nodes from A. All occurrences of these nodes are removed from X and reinserted in randomly selected positions.
Random Swap(X, k) randomly selects two nodes in X and swaps their positions. This operation is performed k times.
Random Move(X, k) randomly selects a node in X and moves it to a random location. This operation is performed k times.
Random Move All(X, k) randomly selects a node a from A. All occurrences of a are randomly moved in X up to a maximal distance k from their original locations. This operation is performed k times.

Table 1. Perturbations

problem-specific constraints are hidden from the solver and must be enforced by the penalty function $p(X)$.

The solver contains four sets of alternative components: metaheuristics (MH), local search heuristics (LS), perturbations (P) and construction procedures (C). When solving a specific problem, one component from each of these categories must be set. The solver also contains a list of local search operators (O), at least one of which must always be used. The best configuration for a given problem is automatically identified using the Iterated Racing tool (irace) [12]. The remainder of this section provides a brief description of individual components, some of which were adapted from the literature and are described only briefly.

Metaheuristics are top-level procedures that control the entire optimization process. Two metaheuristics are currently implemented in the proposed solver: Iterated Local Search **ILS**(C, LS, P, O, k) [13] and Variable Neighborhood Search **VNS**(C, LS, P, O, k_{min}, k_{max}) [15]. Both metaheuristics construct an initial solution and then alternate local search and perturbation, until the timeout is reached. The parameter k is a numerical parameter of the perturbation, which is fixed in the ILS and variable in the VNS.

Local search heuristics control the application of individual local search operators in O to a current solution X, with the aim of reaching a new local optimum. Five variants of the Variable Neighborhood Descent (VND) heuristic adapted from [7] are implemented in the solver: Basic **BVND**(X, O), Pipe **PVND**(X, O), Cyclic **CVND**(X, O), Random **RVND**(X, O) and Random Pipe **RPVND**(X, O). All VND variants terminate and return a possibly improved solution X, when none of the operators in O succeeds in improving X. The only difference is the rule for iterating through O.

Perturbations are intended to randomly modify a current solution X in order to escape a local optimum. Six perturbations are implemented in the proposed solver and described in Table 1. All perturbations take a current solution X and a numerical parameter k as input and return a modified solution regardless of its fitness. Generally, the value of k determines the intensity of a perturbation.

Construction procedures are used to create an initial solution. Three constructions described in Table 2 are implemented in the proposed solver. All constructions return a solution X valid w.r.t. Eq. 8.

Local search operators define different neighborhoods of a solution X. When applying a local search operator, the entire neighborhood is searched, and the most improving modification of X w.r.t. $g(X)$ is applied. All operators perform improving moves only and are not allowed to violate the constraint given by Eq. 8. Some operators take additional parameters which are fixed in the solver. Thus, multiple variants of some operators are instantiated. The ranges of these parameters were set empirically and are given in the operator description below. The 11 operators described in Table 3 are implemented in the solver.

For some of these operators, the corresponding neighborhood is a subspace of another operator's neighborhood (e.g. Exchange and Reverse Exchange, Centered Exchange and Two-opt). The reason for this is that smaller neighborhoods can be searched faster and still lead to new local optima. On the other hand, a larger neighborhood may be beneficial when the search stagnates.

4 Results

The proposed solver described in Section 3.2 was implemented in C++. Both the implementation and the solution files are publicly available at [22]. The solver was benchmarked against the exact Gurobi Optimizer. The solver was parallelized using the OpenMP library, and its configuration for each problem was tuned by the irace package with a budget of 4000 experiments per problem. Both solvers were allowed to use eight threads. All experiments were carried out on dedicated machines with Ubuntu 18.04 OS, Intel Core i7-7700 CPU, 32 GB RAM memory, and 34 GB swap memory.

For each problem, a data set of 10 commonly used test instances of variable size and difficulty was selected. A different dataset of 20 training instances was used for automatic configuration of the proposed solver. The tuned configurations for each problem are shown in Table 4. Both solvers were given the same computing time, which was set empirically according to the size of the instance. Each instance was solved once by the Gurobi and fifty times by the proposed solver, as it is stochastic.

The following data are presented for each instance: the best known solution from the literature (BKS), the computing time budget (*time*), the lower bound and the gap value of the solution fitness found by the Gurobi (LB, *gap*), the

Greedy() repeatedly applies the local search operator Insert(X). The construction terminates, when $\forall a_i \in A : f_i = l_i$.
Random() repeatedly inserts nodes into random locations in X, until $\forall a_i \in A : f_i = l_i$.
Random-replicate() first generates a random permutation of all nodes in A. A copy of this permutation is repeatedly appended to itself, until $\forall a_i \in A : f_i \geq l_i$. If $f_i = u_i$, a_i is removed from the appended copy.

Table 2. Construction procedures

Insert(X) attempts to insert a node $a_i \in A$ to a position $j \in \{1, 2, ..., m\}$. The most improving combination of i, j w.r.t. $g(X)$ is used. As the operator is used in greedy construction, it internally extends the penalty function as $p(X) = p(X) + M^2 \sum_{i=1}^{n} \max(l_i - f_i, 0)$.
Remove(X) attempts to remove a node $x_i \in X$.
Two-opt(X) attempts to reverse a substring of X given by indices $i, j \in \{1, 2, ..., m\}$.
Exchange Nodes(X) attempts to exchange all occurrences of nodes $a_i, a_j \in A$ in X.
Exchange First Nodes(X) attempts to exchange first k occurrences of nodes $a_i, a_j \in A$ in X, where $k \in \{1, 2, ..., \max(f_i, f_j)\}$.
Exchange(X, p, q) attempts to exchange substrings of fixed lengths p, q in X. Variants: $(p, q) \in \{(1, 1), (1, 2), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)\}$.
Reverse Exchange$(X,\ p,\ q)$ operates similarly to the Exchange operator, but it also attempts to reverse one or both substrings before exchanging.
Centered Exchange$(X,\ p)$ reverts the substring of length $2p + 1$, starting at position $i \in \{0, m - p + 1\}$. Variants: $p \in \{1, 2, 3, 4, 5\}$.
Move$(X,\ p)$ attempts to move a substring of X with fixed length p from a position $i \in \{1, 2, ..., m - p + 1\}$ to a position $j \in \{1, 2, ..., m\}$. Variants: $p \in \{1, 2, 3, 4, 5\}$.
Reverse Move$(X,\ p)$ operates similarly to the Move operator, but it also attempts to reverse the substring. Variants: $p \in \{2, 3, 4, 5\}$.
Move all$(X,\ p)$ attempts to move all occurrences of a node $a_i \in A$ in X. All occurrences are shifted by the same distance distance $p \in \{-p, ..., 0, ..., p\}$ from their original positions. Variants: $p \in \{1, 2, 3, 4, 10\}$.

Table 3. Local search operators

same values obtained in ten times the computing budget (LB_{10t}, gap_{10t}), the best gap, the mean gap, and the standard deviation achieved by the proposed solver ($gap^*, \overline{gap}$, $stdev$). The values of gap are calculated as $gap(\%) = 100(\frac{\hat{g}(X)}{BKS} - 1)$. BKSs that are known to be optimal are written in bold.

Problem	Configuration
CVRP	MH: VNS($k_{min} = 3$, $k_{max} = 8$); C: Random; LS: RVND; P: Random Double Bridge; O: Insert, Remove, Two-opt, Exchange Nodes, Exchange($(p, q) \in \{(1, 1), (1, 2), (2, 3), (2, 4), (3, 3), (3, 4)\}$), Reverse Exchange($(p, q) \in \{(1, 2)\}$), Centered Exchange($p \in \{1, 2, 3, 4\}$), Move($p \in \{5\}$), Reverse Move($p \in \{2, 3, 5\}$), Move All($p \in \{2, 3, 4\}$)
NPFS	MH: ILS($k = 1$); C: Random-replicate; LS: BVND; P: Random Swap; O: Exchange Nodes, Exchange($(p, q) \in \{(1, 1), (2, 2), (2, 4), (3, 4), (4, 4)\}$). Reverse Exchange($(p, q) \in \{(2, 3)\}$), Centered Exchange($p \in \{2, 4\}$), Move($p \in \{1, 2, 5\}$), Reverse Move($p \in \{4\}$), Move All($p \in \{1, 3, 4\}$)
QAP	MH: ILS($k = 6$); C: Random; LS: PVND; P: Random Move; O: Two-opt, Exchange Nodes; Exchange($(p, q) \in \{(1, 1), (2, 2), (2, 3), (2, 4), (3, 4)\}$), Reverse Exchange($(p, q) \in \{(2, 2), (2, 4), (4, 4)\}$), Centered Exchange($p \in \{1, 2, 3, 5\}$), Move($p \in \{3\}$), Reverse Move($p \in \{3\}$), Move All($p \in \{1, 2, 3\}$).

Table 4. Tuned configurations

CVRP results are presented in Table 5. The problem was solved in Gurobi using the Miller-Tucker-Zemlin ILP formulation [16]. Testing instances were selected from CVRPLIB [11]. The proposed solver was able to find the optimum for the smallest three instances and for most of the instances, both gap^* and $\overline{gap}$ are within 5%. The proposed solver found a valid solution every time. On the other hand, Gurobi generated a valid solution only for the three smallest instances and with significantly larger gap. For the remaining seven instances, Gurobi returned only a lower bound, which did not improve much even in the longer runs with ten times the computing time.

NPFS results are presented in Table 6. The ILP formulation was adapted from [1] and the testing instances were selected from the dataset [20]. The BKS

			Gurobi				Proposed solver		
Instance	$time[s]$	BKS	LB	LB_{10t}	gap	gap_{10t}	gap^*	$\overline{gap}$	$stdev$
P-n050-k10	600	**696**	567	579	2.73%	1.01%	**0.00%**	0.18%	1.07
A-n65-k09	600	**1174**	822	831	29.05%	2.47%	**0.00%**	0.22%	2.03
P-n076-k05	600	**627**	567	571	13.88%	6.86%	**0.00%**	0.16%	1.50
X-n148-k46	1800	**43448**	23072	23705	-	-	0.12%	0.51%	89.94
X-n204-k19	1800	**19565**	14105	14118	-	-	0.57%	1.19%	60.48
X-n251-k28	1800	**38684**	20779	20779	-	-	1.04%	1.47%	95.73
X-n351-k40	3600	25896	14061	14061	-	-	3.84%	5.02%	174.04
X-n561-k42	3600	42717	25226	25768	-	-	2.58%	3.47%	200.30
X-n749-k98	3600	77269	30465	30805	-	-	4.93%	6.10%	388.35
X-n1001-k43	3600	72355	29862	30419	-	-	7.62%	8.90%	395.64

Table 5. CVRP results

values shown were generated for the Permutation Flowshop Problem (PFS), which requires the same job order on each machine. These are valid for the NPFS as well, but may be suboptimal. The proposed solver found a valid solution every time, and both gap^* and $\overline{gap}$ are within 7% for most instances. BKS was found twice, and in one case, a solution of better quality than PFS BKS was found. Gurobi found a valid solution for eight instances. For the remaining two, Gurobi failed to build the problem model due to excessive memory requirements. The gap and gap_{10t} values found by the Gurobi do not differ significantly from each other and are several times higher than those of the proposed solver.

			Gurobi				Proposed solver		
Instance	$time[s]$	BKS	LB	LB_{10t}	gap	gap_{10t}	gap^*	$\overline{gap}$	$stdev$
VFR20_5_1	600	1192	810	814	6.29%	3.61%	0.00%	0.16%	1.15
VFR20_15_1	600	1936	1409	1437	7.95%	4.86%	-0.41%	0.12%	3.83
VFR40_5_1	600	2396	1452	1460	6.59%	6.01%	0.00%	0.00%	0.00
VFR100_20_1	1800	6121	3482	3518	28.41%	27.46%	4.20%	5.28%	29.11
VFR100_20_2	1800	6119	3590	3622	29.14%	27.46%	5.77%	6.81%	25.53
VFR200_20_1	1800	11181	3684	5806	22.81%	22.73%	4.52%	5.52%	62.15
VFR400_40_1	3600	23085	2482	11997	22.57%	22.57%	6.14%	7.40%	165.19
VFR400_60_1	3600	25214	-	-	-	-	5.93%	6.97%	144.22
VFR600_40_1	3600	33337	2555	2555	18.86%	18.86%	4.80%	5.88%	146.23
VFR600_60_1	3600	35450	-	-	-	-	5.83%	6.89%	172.28

Table 6. NPFS results

QAP results are presented in Table 7. The Xia Yuan linearization [23] was used in the MILP formulation and the testing instances were selected from QAPLIB [3]. For this problem, both Gurobi and the proposed solver found a valid solution every time. For this reason and because of space limitations, the values of LB and LB_{10t} are omitted. The proposed solver found the BKS for six instances. Gurobi performed significantly worse than the proposed solver in both standard and long runs, although the values of gap_{10t} improved noticeably compared to gap.

5 Conclusions

In this paper, a unifying formalism for combinatorial optimization problems with permutative representation is proposed, together with a metaheuristic solver capable of addressing this class of problems. The goal is to provide a versatile solver that requires the user only to model the problem, similarly to various

			Gurobi		Proposed solver		
Instance	$time[s]$	BKS	gap	gap_{10t}	gap^*	$\overline{gap}$	$stdev$
tai20b	600	**122455319**	0.45%	0.45%	**0.00%**	**0.00%**	0.00
tai25a	600	**1167256**	4.88%	3.26%	**0.00%**	0.37%	2764.26
tai30b	600	**637117113**	4.53%	4.47%	**0.00%**	**0.00%**	0.00
tai40a	1800	3139370	5.17%	4.28%	0.31%	1.07%	9752.18
tai50b	1800	458821517	7.67%	4.30%	0.00%	0.09%	726353.31
tai60a	1800	7205962	5.64%	5.53%	1.07%	1.80%	18626.86
tai80a	3600	13499184	6.39%	5.56%	1.36%	1.82%	33320.47
tai80b	3600	818415043	12.94%	7.72%	0.00%	1.06%	5389291.60
tai100a	3600	21052466	11.75%	10.11%	1.36%	1.77%	52545.36
tai100b	3600	1185996137	19.98%	10.20%	0.00%	0.87%	11497905.21

Table 7. QAP results

MILP solvers, but offers better scalability, otherwise typical for problem-specific metaheuristic solvers.

Three different $\mathcal{NP}$-hard problems (CVRP, NPFS, and QAP) are formulated in the proposed formalism. The proposed solver is automatically configured separately for each problem and benchmarked against the Gurobi Optimizer. The experiments show that the proposed solver consistently reaches solutions of significantly better quality than Gurobi, given a fixed time limit. Moreover, it is capable of finding good quality solutions to such problems, for which Gurobi either finds no solution at all in the given time or fails to build the problem model due to memory limitations.

In future work, other metaheuristics and low-level components will be implemented to extend the solver capabilities. The solver will also be adapted to solve richer problems in terms of number of constraints, as it relies on transforming these into penalty functions. Finally, the solver will be compared with problem-specific metaheuristic algorithms and further improved toward offering similar scalability and solution quality.

Acknowledgements

This work has been supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000470). The work of David Woller has been also supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS21/185/OH K3/3T/37 and by the Czech Science Foundation (GACR) under Grant Agreement 19-26143X. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Benavides, A.J., Ritt, M.: Fast heuristics for minimizing the makespan in non-permutation flow shops. Computers & Operations Research **100**, 230–243 (2018)
2. Blot, A., Hoos, H.H., Jourdan, L., Kessaci-Marmion, M.É., Trautmann, H.: MO-ParamILS: A multi-objective automatic algorithm configuration framework. In: Learning and Intelligent Optimization, pp. 32–47. Springer Int. Pub., Cham (2016)

3. Burkard, R., Çela, E., Karisch, S., Rendl, F.: QAPLIB (2012). URL https://coral.ise.lehigh.edu/data-sets/qaplib/
4. De Beukelaer, H., Davenport, G.F., De Meyer, G., Fack, V.: JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics. Software - Practice and Experience **47**(6), 921–938 (2017)
5. Deshwal, A., Belakaria, S., Doppa, J.R., Kim, D.H.: Bayesian optimization over permutation spaces. Proceedings of the AAAI Conf **36**(6), 6515–6523 (2022)
6. Dreo, J., Liefooghe, A., Verel, S., Schoenauer, M., Merelo, J.J., Quemy, A., Bouvier, B., Gmys, J.: Paradiseo: From a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of Paradiseo. In: GECCO 2021 Companion, pp. 1522–1530. Association for Computing Machinery, Inc (2021)
7. Duarte, A., Mladenović, N., Sánchez-Oro, J., Todosijević, R.: Variable neighborhood descent. In: Handbook of Heuristics, vol. 1-2, pp. 341–367. Springer Int. Pub. (2018)
8. Hadka, D., Reed, P.M., Simpson, T.W.: Diagnostic assessment of the borg MOEA for many-objective product family design problems. In: 2012 IEEE Congress on Evolutionary Computation, CEC 2012 (2012)
9. Helsgaun, K.: An Extension of the LKH TSP Solver for Constrained TSP and VRP. Tech. rep., Roskilde University (2017)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization, pp. 507–523. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
11. Lima, I.: CVRPLIB (2014). URL http://vrp.galgos.inf.puc-rio.br/
12. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
13. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: Framework and applications. In: Handbook of Metaheuristics, pp. 129–168. Springer Int. Pub., Cham (2019)
14. Mehdi, M.: Parallel Hybrid Optimization Methods for Permutation Based Problems. Ph.D. thesis, Université des Sciences et Technologie de Lille (2011)
15. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **24**(11), 1097–1100 (1997)
16. Moghadam, B.F., Sadjadi, S.J., Seyedhosseini, S.M.: Comparing mathematical and heuristic methods for robust VRP. IJRRAS **2**(2), 108–116 (2010)
17. Parejo, J.A., Ruiz-Cortés, A., Lozano, S., Fernandez, P.: Metaheuristic optimization frameworks: A survey and benchmarking. Soft Comput. **16**(3), 527–561 (2012)
18. Scott, E.O., Luke, S.: ECJ at 20: Toward a general metaheuristics toolkit. In: GECCO 2019 Companion, pp. 1391–1398. ACM (2019)
19. Stutzle, T., López-Ibáñez, M.: Automated design of metaheuristic algorithms. In: Handbook of Metaheuristics, pp. 541–579. Springer Int. Pub., Cham (2019)
20. Vallada, E., Ruiz, R., Framinan, J.M.: New hard benchmark for flowshop scheduling problems minimising makespan. European Journal of Operational Research **240**(3), 666–677 (2015)
21. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research **234**(3), 658–673 (2014)
22. Woller, D.: Permutator github repository (2022). URL https://github.com/wolledav/permutator
23. Xia, Y., Yuan, Y.X.: A new linearization method for quadratic assignment problems. Optimization Methods and Software **21**(5), 805–818 (2006)