Empirical Analysis of Operators for Permutation Based Problems

Pierre Desport, Matthieu Basseur, Frédéric Lardeux, Adrien Goëffon, Frédéric Saubion

Empirical Analysis of Operators for Permutation Based Problems

Pierre Desport, Matthieu Basseur, Adrien Goëffon, Frédéric Lardeux, and Frédéric Saubion

LERIA, University of Angers (France)

Abstract. This paper presents an analysis of different possible operators for local search algorithms in order to solve permutation-based problems. These operators can be defined by a distance metric that define the neighborhood of the current configuration, and a selector that chooses the next configuration to be explored within this neighborhood. The performance of local search algorithms strongly depends on their ability to efficiently explore and exploit the search space. We propose here a methodological approach in order to study the properties of distances and selectors in order to buildtheir performances operators that can be used either for intensification of the search or for diversification stages. Based on different observations, this approach allows us to define a simple generic hyperheuristic that adapt the choice of its operators to the problem at hand and that manages their use in order to ensure a good trade-off between intensification and diversification. Moreover this hyperheuristic can be used on different permutation-based problems.

1 Introduction

Many optimization problems can be modeled as permutation problems (e.g., flowshop, traveling salesman or quadratic assignment problems). Dedicated efficient solving methods have been proposed for these problems, but their performance often depend on the considered instances and use most of the time ad-hoc heuristics and/or techniques. Therefore, given a new permutation problem a non-expert user would hardly be able to design a solving algorithm using components that she/he could re-use from previous experiments. A recent trend in optimization consists in promoting more autonomous techniques for the design of search algorithms [1], either by automating the tuning of their parameters [2], by dynamically controlling their behaviour [3] or by automating their design [4]. Focusing on this latest aspect, hyperheuristics [5] is a generic paradigm that can be used to manage a set of efficient heuristics in order to solve a sufficiently large set of problem instances, without *a priori* knowledge. The main principle of hyperheuristics is to adapt the solving process to the given instance at hand. Considering a set of possible heuristics that can be applied on a given problem, one may select the best heuristics with regards to the characteristics and properties of this problem and/or change heuristics during the solving process, according to the current state of the search. Of course, this high level management of the solving heuristics requires to define sufficiently general heuristics and to gather pertinent information on them. In this paper, our focus on permutation problems allows us to consider a sufficiently large family of problems which can be handled by a set of common search operators (from now on we use the generic term operator for our basic search heuristics). These operators, defined in a local search fashion [6], aim at selecting the next configuration of the search space that will be examined by an incremental search process. Such an operator can easily be defined by (1) a notion of neighborhood of the current configuration, thanks to a distance measure, and (2) a selection function within this neighborhood. One aim of this work is thus to carefully study the behaviour of these search operators with regards to different instances of well known permutation problems. Note that different studies on distances for permutation problems have been conducted [7, 8]. Here we focus on how operators, which use such distances, can be compared and efficiently chosen with regards to a given instance of a problem. Note that the dynamic control of operators in local search has been studied in [9]. Our approach can be more related to landscape analysis [10]. An analysis of the correlations between the two basic above-mentioned components of operators would help us to better understand their characteristics and to define a simple hyperheuristic to manage them.

This paper is organized as follow. Basic notions on permutations are recalled in Section 2. Main concepts concerning local search for combinatorial optimization problems are presented in Section 3. The next sections are devoted to the analysis of search landscapes induced by the static structures of the problems, as well as the operational behaviour of local search operators. A simple hyperheuristics is then defined in Section 6 in order to illustrate how previous observations can be used to improve solving algorithms.

2 Basic Notions for Permutations

2.1 Permutations

A permutation of n elements is an arrangement of these n objects sorted in a specific order where they appear only once. The group of permutations [11] can be defined as the group of bijections from X to X where X is a non-empty finite set. Let $[n]$ be a set of objects $[n] = \{1..n\}$. A permutation π is a bijective assignment on $[n]$ such that $\pi(i)$ is the element at position i in the permutation π and $pos_\pi(i)$ is the position of the element i in the permutation π. $\Pi([n])$ is the set of permutations on $[n]$, whose cardinality is thus $n!$. Given $[n] = \{1, 2, 3, 4\}$, $\pi = (1, 2, 3, 4)$ or $\pi = (2, 1, 4, 3)$ are two possible permutations.

Identity. Let $\pi \in \Pi([n])$, the identity permutation I is defined as the permutation that assigns each element of π to itself, i.e. $\forall i \in \{1..n\}, I(i) = i$.

Product of permutations. Let $\pi_1, \pi_2 \in \Pi([n])$, $\pi_1(i) * \pi_2(i) = \pi_2(\pi_1(i)) \forall i \in [n]$. Note that $\pi_1 * \pi_2 \neq \pi_2 * \pi_1$. The neutral element of the product is I (i.e., $\forall \pi, \pi * I = I * \pi = \pi$).

Inverse. The inverse permutation can be defined using the identity permutation π^{-1} can be constructed from π using the property: $\pi^{-1}(i) = pos_\pi(i)$. For instance, if $\pi = (2, 3, 4, 1)$, then $\pi^{-1} = (4, 1, 2, 3)$.

Let us recall now some properties of permutations.

Adjacency Given a permutation $\pi \in \Pi([n])$, two elements i and j are adjacent in π if $|pos_\pi(i) - pos_\pi(j)| = 1$.

Longest Increasing Sequence (LIS). Given a permutation π, the longest increasing subsequence LIS(π) corresponds to the longest subsequence of elements of π that are sorted in ascending order. For instance, for $\pi = (1, 3, 2, 4)$ the longest increasing subsequences are $(1, 2, 4)$ and $(1, 3, 4)$.

Longest Common Subsequence (LCS) The longest common subsequence of two permutations $\pi_1, \pi_2 \in \Pi([n])$ is $LCS(\pi_1, \pi_2) = \{i \in \{1..n\}|p_1(i) = p_2(i)\}$. For instance, with $\pi_1 = (2, 4, 3, 1, 5, 6)$ and $\pi_2 = (1, 2, 3, 5, 4, 6)$, $LCS(\pi_1, \pi_2) = \{2, 3, 5, 6\}$.

2.2 Distance on Permutations

Table 1 presents different distance indicators which will be considered in the rest of the paper. The diameter of a distance measure represents the maximal distance between all permutations. Note that, as an example, the last column corresponds to the distance between the two following permutations (# is the cardinality function):

- $\pi_1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$ and,
- $\pi_2 = (15, 3, 1, 11, 2, 7, 9, 10, 4, 14, 6, 12, 5, 13, 8)$.

Table 1. Distances for permutations.

Distances	**Formula (π_1 and π_2 are permutations)**	**Diameter**	**Ex.**
Hamming	$\#\{i \mid i \in \{1..n\}, \pi_1(i) \neq \pi_2(i)\}$	n	14
Adjacency	$n - 1 - \#\{1 \leq i \mid adj_{\pi_2}(\pi_1(i), \pi_1(i+1))\}$	$n-1$	13
Position	$\sum_{i=1}^{n} \lvert pos_{\pi_1}(i) - pos_{\pi_2}(i)\rvert$	$2\lceil n/2\rceil\lfloor n/2\rfloor$	62
Lee	$\sum_{i=1}^{n} min(\lvert\pi_1(i) - \pi_2(i)\rvert, n - \lvert\pi_1(i) - \pi_2(i)\rvert)$	$n(n/2)$	48
Swap	$\#\{(i,j) \mid 1 \leq i < j \leq n, pos_{\pi_2}(\pi_1(j)) < pos_{\pi_2}(\pi_1(i))\}$	$n(n-1)/2$	44
Interchange	$n - c(\pi_1^{-1} * \pi_2)$	$n-1$	12
Ulam	$n - length(LIS(\pi_1^{-1} * \pi_2))$	$n-1$	9
Insertion	$n - length(LCS(\pi_1, \pi_2))$	$n-1$	8

Note that the interchange distance requires a function, computing the number of permutation cycles which composes a permutation [8].

3 Optimization Problems and Local Search

Let us define the components of a local search algorithm in the context of solving optimization problems [6].

3.1 General Definitions

Optimization Problem An optimization problem is a pair $(\mathcal{S}, f)$ where $\mathcal{S}$ is a search space whose elements represent solutions (or configurations) of the problem and $f : \mathcal{S} \to \mathbb{R}$ is an objective function. An optimal solution (for maximization problems) is an element $s^* \in \mathcal{S}$ such that $\forall s \in \mathcal{S}, f(s^*) \geqslant f(s)$.

Local Search Given an optimization problem, a local search (LS) process consists in starting from an initial configuration and in applying repeatedly basic move operators in order to reach an optimal solution. The trace obtained by such a search process is usually called a search path. An operator is thus a function that returns the next configuration for building the search path. In its simplest form, a move performs the selection of the next configuration to be explored within the neighborhood of the current configuration. A generic and basic outline of an LS metaheuristic is the application of an operator in a simple loop as illustrated in Algorithm 1 where the `SpecificAction()` method represents a step specific to the type of metaheuristic used such as a perturbation (e.g., Iterated Local Search [6]) or enforcing prohibitions (e.g., Tabu Search).

```
$s \leftarrow$ initial configuration;
$s^* \leftarrow s$;
while end condition not met do
    $s \leftarrow op(s)$;
    if $eval(s) < eval(s^*)$ then $s^* \leftarrow s$;
    SpecificAction();
end
return $s^*$
```

Algorithm 1: Algorithmic outline of an LS metaheuristic for minimization

Neighborhood Let $\mathcal{S}$ be the search space of candidate solutions. A neighborhood relation is an irreflexive binary relation $\mathcal{N} \subseteq \mathcal{S}^2$ over the search space. In most cases, the relation is also symmetric.

Search Paths Given a neighborhood relation $\mathcal{N}$, the set of search paths is defined as $\mathcal{P}_{\mathcal{N}} = \{s_1 \cdots s_n \in \mathcal{S}^* \mid \forall i > 1, (s_{i-1}, s_i) \in \mathcal{N}\}$, where $\mathcal{S}^*$ represents the set of words constructed over $\mathcal{S}$. Therefore any pair (s, s') of elements of $\mathcal{S}$, such that[1] $(s, s') \in \mathcal{N}^+$, defines an equivalence class over the set $\mathcal{P}_{\mathcal{N}}$ which

[1] $\mathcal{N}^+$ is the transitive closure of $\mathcal{N}$.

corresponds to all paths that link s to s'. This subset is denoted by $\mathcal{P}_{\mathcal{N}}(s, s')$. In most cases, the neighborhood should be complete, i.e. $\forall s, s' \in S, \mathcal{P}_{\mathcal{N}}(s, s') \neq \emptyset$.

Distances induced by neighborhood The neighborhood relation defines the structure of the search space. The distance between s and s' can therefore be defined as $d_{\mathcal{N}}(s, s') = min_{p \in \mathcal{P}_{\mathcal{N}}(s,s')} |p|$, where $|p|$ is the classic word length and $d_{\mathcal{N}}(s, s) = 0$. If d define a distance, then $\mathcal{N}$ is necessarily symmetric. Note that a neighborhood induces a distance on the search space but, conversely, a distance on the search space can easily be used to define a neighborhood.

Local Search Operators An operator is defined by two main components: neighborhood and selection process. A selector is a function that performs a selection over a neighborhood, eventually guided by the ordering $<$, and is defined as $\sigma : \mathcal{S} \times 2^{\mathcal{S}^2} \rightarrow \mathcal{S}$ (here the selection returns only one neighbor), such that $(s, \sigma(s, \mathcal{N})) \in \mathcal{N}^{=}$ (the reflexive closure of $\mathcal{S}$ in order to include identity). Note that the selectors may include randomization for computing their results (e.g., random choice of a neighbor). Let us consider three classic different selectors:

- *First improve* (select the first improving neighbor from a randomly ordered set of neighbors),
- *Best improve* (select the best neighbor),
- *Random choice* (randomly chosen neighbor).

An operator is fully defined by a pair $(\mathcal{N}, \sigma)$.

Search Landscape. The search landscape is usually defined by the search space and the objective function that should be maximized (without loss of generality). The ordering relation $<$ over $\mathcal{S}$ corresponds to the order induced by the fitness function of the problem.

Operational Landscape. The operational structure of local search is defined by the possible moves in the search landscape according to a neighborhood relation. Again, we consider the paths induced by an operator $o = (\mathcal{N}, \sigma)$:

$$\mathcal{P}_o = \bigcup_{n>1} \{s_1 \cdots s_n \in \mathcal{S}^* | \forall i > 1, s_i = \sigma(s_{i-1}, \mathcal{N})\}$$

Here, we should note that we only have the inclusion $\mathcal{P}_o \subseteq \mathcal{P}_{\mathcal{N}}$, since some neighborhood paths cannot be necessarily constructed by the operators as soon as it includes a selection process among the neighbors. Moreover, if there exists a path in $\mathcal{P}_o$ linking s to s', there does not necessarily exist a path from s' to s. Therefore, due to this non symmetric aspect of operators, it is not obvious to use a simple distance over the paths created by the operators. Now we may handle multiple move operators local search by composing neighborhood relations and selectors.

3.2 Permutations Based Problems

According to the previous definitions, a permutation based problem is a problem such that $\mathcal{S} = \Pi([n])$. Many combinatorial optimization problems can be formulated as permutation-based problems, most of them being NP-hard. In this section we recall three well-known permutation problems that will be used in our experiments.

Quadratic Assignment Problem. Quadratic assignment problem (QAP) [12] models a facilities location problem. The objective of QAP is to assign n facilities to n locations in order to minimize the assignment cost. It may be formalized as follows:

- let f_{ij} the flow between facilities i and j,
- let d_{ij} the distance between locations i and j,
- minimize: $\sum\limits_{i,j=1}^{n} f_{ij} d_{\pi(i)\pi(j)}$.

Flowshop Problem. The flowshop problem [13] is a scheduling problem where the goal is to find the best planning to achieve n jobs on m different machines, minimizing the makespan (total completion time), considering the following constraints :

- All jobs must be processed by all machines,
- A machine can deal with only one job at any time t,
- A job can be processed only by one machine at any time t.

Here we consider that the processing order of the jobs on the machine is always the same. The goal is to find a permutation π representing a processing order of the jobs that minimizes the makespan function $C_{max} = max\{s_{iM} + p_{iM} | i \in [1..n]$ and $M \in [1..m]\}$ where:

- p_{ij} is the time for the machine i to process the job i,
- s_{ij} is the starting time of the job i on the machine j.

Traveling Salesman Problem. The traveling salesman problem (TSP) [14] consists in finding the shortest path in order to visit n cities without visiting any city twice. This constraint is easily enforced by using permutations to represent configurations of the problem. For a permutation π, each element is a city. Given a matrix D such that d_{ij} corresponds to the distance between city i and city j, the objective function to minimize is $\sum\limits_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$. Here, we restrict our study to symmetric TSP (i.e., $d_{ij} = d_{ji}\ \forall i, j \in \{1, 2, .., n\}$).

4 Search Landscape Analysis

In this section, we propose a first study of the search landscape corresponding to the previously selected permutation problems. Our purpose is to highlight how suitable but yet generic local search operators can be defined. We will first study the search landscape from a static point of view by using distance indicators on permutations that have been presented in Section 2.2. This will allow us to exhibit correlations between distances as well as correlations between problems and distances. Before starting our experiments, let us observe what are the typical topological characteristics of our problems, computing the maximal theoretical and the average distances between two permutations.

4.1 Search Space Diameters

The following results will help us to better interpret our further studies. Table 2 provides maximal theoretical (max) and average (avg) distances for three instances. Note that here the original problems have no influence since we consider only the search landscape, which only depends on the size of the permutation. We output here instances from size 20 to 100 in order to highlight the relative differences between distances. Results show similar properties when considering larger instances. Average results are obtained by computing the distance between 10^5 pairs of randomly generated permutations.

Table 2. Diameter and average distances

Instance		Ham.	Adj.	Position Based	Lee	Swap	Inter-change	Insertion	Ulam
Inst. 1	max	20	19	200	200	190	19	19	19
(size 20)	avg	19	17	133	100	95	16	13	14
Inst. 2	max	26	25	338	338	325	25	25	25
(size 26)	avg	25	23	225	169	162	22	18	19
Inst 3.	max	100	99	5000	5000	4950	99	99	99
(size 100)	avg	99	97	3331	2500	2475	95	92	93

4.2 Search Space: Correlation Between Distances

The purpose of this first experiment is to compare the distance indicators. Table 3 shows correlations of distances obtained for 10^5 randomly generated pairs of permutations. Let us recall that the problem under consideration have no influence on the results. We considered two close sizes (20 and 26) and a larger one (100) in order to obtain different effects. Nevertheless, correlations do not depend on the size, as observed here. This study can be related to the work presented in [8] but with additional metrics. Moreover, the experimental process is slightly different even if similar conclusions are reported.

The correlation coefficient cf corresponds to the intensity of the connection between two sets of values and has a value ranging in $[-1, 1]$. Two sets are said to be strongly correlated if $|cf| > 0.5$.

$$cf(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) * (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} * \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ are the mean values of x and y.

Table 3. Correlation distances - distances

Instance	Distance	Ham.	Adj.	Position Based	Lee	Swap	Inter-Change	Insertion	Ulam
Size 20	Hamming	**1.000**	0.004	0.360	0.393	0.226	**0.672**	0.214	0.168
	Adjacency	0.004	**1.000**	-0.003	0.001	-0.005	0.026	0.087	-0.005
	PositionBased	0.360	-0.003	**1.000**	0.142	**0.939**	0.243	**0.605**	0.061
	Lee	0.393	0.001	0.142	**1.000**	0.090	0.261	0.088	0.308
	Swap	0.226	-0.005	**0.939**	0.090	**1.000**	0.148	**0.621**	0.040
	Interchange	**0.672**	0.026	0.243	0.261	0.148	**1.000**	0.124	0.098
	Insertion	0.214	0.087	**0.605**	0.088	**0.621**	0.124	**1.000**	0.038
	Ulam	0.168	-0.005	0.061	0.308	0.040	0.098	0.038	**1.000**
Size 26	Hamming	**1.000**	0.026	0.302	0.344	0.188	**0.640**	0.189	0.132
	Adjacency	0.026	**1.000**	0.005	0.005	0.000	0.021	0.089	0.008
	PositionBased	0.302	0.005	**1.000**	0.093	**0.941**	0.200	**0.589**	0.050
	Lee	0.344	0.005	0.093	**1.000**	0.050	0.213	0.061	0.286
	Swap	0.188	0.000	**0.941**	0.050	**1.000**	0.121	**0.594**	0.027
	Interchange	**0.640**	0.021	0.200	0.213	0.121	**1.000**	0.111	0.077
	Insertion	0.189	0.089	**0.589**	0.061	**0.594**	0.111	**1.000**	0.040
	Ulam	0.132	0.008	0.050	0.286	0.027	0.077	0.040	**1.000**
Size 100	Hamming	**1.000**	0.003	0.220	0.242	0.137	**0.575**	0.137	0.081
	Adjacency	0.003	**1.000**	0.001	0.003	0.000	0.009	0.059	0.001
	PositionBased	0.220	0.001	**1.000**	0.056	**0.944**	0.127	**0.557**	0.018
	Lee	0.242	0.003	0.056	**1.000**	0.037	0.137	0.035	0.228
	Swap	0.137	0.000	**0.944**	0.037	**1.000**	0.077	**0.532**	0.011
	Interchange	**0.575**	0.009	0.127	0.137	0.077	**1.000**	0.071	0.044
	Insertion	0.137	0.059	**0.557**	0.035	**0.532**	0.071	**1.000**	0.015
	Ulam	0.081	0.001	0.018	0.228	0.011	0.044	0.015	**1.000**

Note that the correlation between distances is measured only on the distances obtained between permutations. The objective function of the problem is not involved in the process (only the size of the studied instances influences the results rather than the type of problem). We can observe in Table 3 that distances can be grouped by sets of correlated distances:

$$\{Hamming, Interchange\}, \{Adjacency\}, \{Lee\},$$
$$\{Swap, PositionBased, Insertion\}, \{Ulam\}.$$

This first observation may help us to select distances for either building search operator or better controlling the search process with a specific heuristic. For instance, if a search algorithm using Hamming distance requires diversification, it seems intuitively appropriated to use another distance that is weakly correlated (for instance Lee distance).

4.3 Search Landscape: Correlation between Problems and Distances

The correlation between the objective function of a problem and the neighborhood relation used to define operators is obviously an important feature to ensure good performance for a LS algorithm. Ideally, if distances between configurations are proportional to their objective values difference, then it is easier to reach good solutions, since moves can be clearly guided by improvement strategies.

In table 4 we examine the correlation between the distance indicators and the problems introduced previously, considering the search landscapes induced by their objectives functions. Instances whose known optimal solution are used to study this correlation. The distance between the optimal permutation and a randomly generated permutation is computed as well as the difference between their objective function values. This process is repeated 10^5 times in order to obtain a correlation value between distances and problems. Results are presented in Table 4.

In Table 4 no strong correlation can be observed. Considering QAP, due to its quadratic objective function, it is very difficult to define a metric that can be correlated to the fitness. For the flowshop problem, distances which induce less perturbations in the objective function values of the configurations, show better results as observed for Swap or Insertion indicators. Similar observation can be done for TSP using Adjacency. Nevertheless, the correlation depends on the size of the considered problems since random points can be far from the optimal solution for problems with large diameters.

5 Analysis of the Operational Landscape

We now turn to the operational point of view by considering search operators that can be used in a local search algorithm. Using the previous results and in order to avoid too combinatorial experiments, we consider only the following classic distances: Swap, Interchange and Insertion. The corresponding neighborhood are indeed often used to build operators for permutation problems. We also

Table 4. Correlation problems - distances

Problem	Instance	Size	Ham.	Adj.	Position Based	Lee	Swap	Inter-Change	Insertion	Ulam
QAP	nug20	20	0.105	0.091	0.031	0.006	-0.005	0.086	0.031	0.001
	lipa30_a	30	0.020	0.000	0.006	0.011	0.001	0.018	-0.007	0.006
	sko81	81	0.032	0.033	0.034	0.009	0.002	0.015	0.044	0.001
	bur26a	26	0.092	0.000	0.104	-0.009	0.012	0.059	0.089	0.032
	esc16a	16	-0.001	0.084	-0.001	0.004	-0.006	0.010	0.015	0.002
TSP	a280	280	0.001	0.166	0.007	0.002	0.001	0.001	0.021	0.006
	berlin52	52	0.000	0.392	0.006	-0.005	0.003	0.003	0.061	-0.002
	eil51	51	0.002	0.392	-0.010	0.002	0.000	-0.001	0.069	-0.005
	kroD100	100	0.003	0.268	0.003	0.001	0.001	0.000	0.039	0.002
	tsp225	225	0.002	0.192	-0.005	-0.001	0.000	-0.002	0.023	-0.001
FlowShop	20_5_01	20	0.089	0.049	0.440	0.020	0.474	0.062	0.278	0.017
	20_10_01	20	0.134	0.029	0.465	0.027	0.469	0.086	0.352	0.007
	20_20_01	20	0.110	0.018	0.331	0.076	0.315	0.075	0.253	0.031
	50_5_01	50	0.040	0.016	0.257	0.001	0.275	0.027	0.163	-0.004
	50_10_01	50	0.055	0.037	0.334	0.013	0.342	0.034	0.203	0.010

consider the three selectors: First, Best and Random. According to the definition of an operator provided in Section 3.1 we have thus nine possible operators.

The purpose of the following experiment is to assess the ability of an operator to reach an optimal solution using the shortest possible path (i.e., using the fewest number of permutations). Here, we aim at studying the short term convergence properties of operators for different problems, in order to identify good candidates for intensification. For different instances of each problem, a permutation is randomly generated at distance n, starting from the optimal known permutation, with the different neighborhoods associated to the operators. We observe then if the operator is able to come back to the optimal solution. Tests have been carried out at various distances from the optimal solution. Here, we are mainly interested by results obtained at a distance 5 from the optimal solution. For smaller distances it seems clear that all operators including the "best" selection mechanism are likely to return to the optimal solution. Oppositely, choosing too long distance for small instances leads to search the optimal solution from a totally random permutation. Table 5 shows results for a distance of 5. Values represent the probabilities of a path built by the operator to reach the optimal permutation.

The results show the efficiency of the operators with regards to intensification for the three problems. An operator that frequently reaches the optimal solution from a distance of 5 is indeed a pertinent operator for the intensification of the search. This experiment assesses that it is sufficient to reach a distance 5 from the optimal solution in order to easily reach it with this operator. Nevertheless, let us notice that maximal distances (i.e., diameters) related to operators are not

Table 5. Ability to find the optimal solution starting at a distance of 5

Problem	Instances	Size	Swap		Interchange		Insertion	
			First Improve	Best Improve	First Improve	Best Improve	First Improve	Best Improve
FlowShop	20_5_01	20	0.7	0.84	0.05	0.13	0.14	0.3
	20_5_02	20	0.54	0.81	0.06	0.15	0.17	0.36
	20_10_01	20	0.22	0.48	0	0.2	0.03	0.27
	20_10_02	20	0.15	0.5	0.01	0.18	0.02	0.28
	50_5_01	50	0.81	0.99	0.44	0.63	0.75	0.87
QAP	nug12	12	0.25	0.45	0.13	0.27	0.03	0.09
	bur26a	26	0.57	0.77	0.11	0.78	0.01	0.03
	els19	19	0.41	0.71	0.1	0.7	0	0.07
	lipa40	40	0.58	0.8	0.84	1	0	0.01
	sko100a	100	0.86	0.99	0.01	1	0	0.07
	chr12a	12	0.12	0.31	0.02	0.25	0	0.02
	scr12	12	0.25	0.35	0.17	0.33	0	0.02
	lipa40a	40	0.56	0.77	0.9	0.99	0	0.02
	wil100	100	0.83	1	0.01	0.99	0	0.05
	tai80b	80	0.35	0.83	0	0.98	0	0.05
TSP	100	rd100	0.58	0.95	0.00	0.97	0.54	0.99
	berlin52	52	0.7	0.96	0.04	0.97	0.47	0.97
	eil51	51	0.49	0.92	0.03	0.95	0.29	0.97
	kroD100	100	0.62	0.96	0.00	0.97	0.40	0.98
	lin105	105	0.7	1	0	0.97	0.49	0.99
	tsp225	225	0.8	1	0	1	0.59	1
	st70	70	0.65	0.96	0.01	0.95	0.32	0.97

of the same order of magnitude. For instance, the *insertion* and *interchange* have a smaller diameter than *swap*. We can remark that associations between operators and problems do not correspond the empirical intuitions. The low distance is certainly an explanation of this behavior.

6 Design of a Simple Hyperheuristic

In this section, the previous analysis are used in order to define a simple hyperheuristic approach for solving the three families of problems. The concept of hyperheuristic [5] has been initially introduced as "a heuristics to choose heuristics". Hyperheuristics manage indeed a set of heuristics and select or combine them in order to efficiently solve problems. Instead of manually designing a solving algorithm, an hyperheuristic is used in order to automate the process of selection, combination or generation of heuristics, aiming at solving different problems with a single generic solver. There is currently a very active community on hyperheursitics and a competition [15] has been launched to compare different approaches.

In the following, we show that the analysis of the search and operational landscape of the problems provided previously may help to design simple generic hyperheuristic for permutation-based problems. Our solving approach is rather simple and alternates two stages: an intensification phase and a diversification phase, which allows the search process to escape of local optima [6]. Using previous experiments, it is possible to characterize operators that promote intensification or diversification.

6.1 Algorithm

Our hyperheuristic algorithm takes as input a set of operators and an instance of a problem. Since the *swap* neighborhood has a larger diameter than the *insertion* or *interchange* neighborhoods, it is not considered in the experiments. Indeed, it is more difficult to use operators with different diameters if one wants to ensure fair comparisons. We consider here only two uncorrelated distances *interchange* and *insertion*. This is a restricted choice and other possible distances could have been considered. The set of possible operators is thus: *interchange* / first improve, *interchange* / best improve, *interchange* / random, *insertion* / first improve, *insertion* / best improve, *insertion* / random.

Using previous experiments, we consider the following methodology:

1. Select an operator for the the intensification stage: the algorithm starts with a study of paths on the instance in order to determine which operator should be considered for intensification. The algorithm selects thus the operator with the highest success rate as the intensification operator. The distances study of Section 4.2 is then used to select the diversification operator.
2. Select an operator for intensification: since the distances *interchange* and *insertion* are weakly correlated, we assume that if an operator using the *insertion* neighborhood is selected for intensification, then it may be interesting to select an operator that uses *interchange* neighborhood for diversification (and vice versa). The random selection mechanism will be considered as selector in order to ensure an efficient diversification.

The hyperheuristic is detailed in Algorithm 2.

6.2 Results

Algorithm 2 has been evaluated on different instances of each problem. The results have been compared to the best known values and to an algorithm that selects uniformly an operator at each iteration among the possible ones. A basic Hill Climbing using a unique operator (combinations of *swap*, *interchange* or *insertion* and first, best or random selection) has been used, but obtains poor results in being stuck in local optima. Same experimental conditions are used to test the different algorithms: a maximum number of iterations is set for each instance, 20 runs are executed for each instance and the same initial permutations are used for the hyperheuristic and the random algorithms. Results are presented in Table 6, in which best values are indicated bold.

```
input: Instance I, Set of operators
output: Value (according to an objective function $f$) of the best permutation
        found

Require:
  $(OpIntensification, OpDiversification) \leftarrow Select - operators(I)$ {Intensification
  operator and diversification obtained by experimentation}
  $p \leftarrow Random()$ {Randomly generated permutation}
  $best \leftarrow f(p)$ {Best fitness}
  while not stop condition do
    $p \leftarrow HillClimbing(OpIntensification, p)$ {HillClimbing process}
    if $f(p) < best$ then
      $best \leftarrow f(p)$
    end if
    $p \leftarrow diversification(p)$ {Diversification process}
  end while
  return $best$
```

Algorithm 2: Hyperheuristic algorithm

We can remark that whatever the instance, the hyperheuristic obtains significantly better results than those obtained by the algorithm with uniform selection, both in terms of best result and average. This difference shows the importance of carefully choosing operators for intensification and diversification. One can also note that as well as being generic, hyperheuristic obtained reasonable results with regards best-known ones – even for large instances, especially considering QAP and flowshop problems. Note that this algorithm is generic in comparison to problem-dedicated algorithms for these well-known problems. Our study can be extended to more operators and problems. Our purpose here was rather to highlight that studying the search space as well as the search landscape may be useful to devise generic hyperheuristics.

7 Conclusion

In this paper, an analysis of operators for permutation-based problems is proposed. Properties of distance measures, neighborhood and selection mechanisms were observed for different permutation-based problems, and provide a better understanding of the relative efficiency of operators. Known relationships between operators and problems have been confirmed. Moreover, collected informations can be used to automate the choice of operators for different permutation-based problems. We have proposed a simple hyperheuristic using these operators properties. Further studies will include more combinatorial experiments with more operators and problem instances.

References

1. Y. Hamadi, E. Monfroy, F. Saubion, What is autonomous search?, Tech. Rep. MSR-TR-2008-80, Microsoft Research (2008).

Table 6. Results

Instance	Size	Hyperheuristic			Best	Uniform		
		best	avg.	s.d.	known	best	avg.	s.d.
				TSP				
Berlin52	52	**7542**	7912	175.6	**7542**	8282	8486.4	131.5
eil51	51	432	438.6	5.7	**426**	445	457.8	7.8
st70	70	690	711.4	13.9	**675**	772	821.8	25.5
kroD100	100	23627	25380.9	1332.73	**21294**	26333	32151.5	2292.3
lin105	105	15761	17785	1196.3	**14379**	17910	22387.5	1927.1
rd100	100	8748	9111.1	275.7	**7910**	10509	11483.3	635.6
tsp225	225	4824	5157.6	269.03	**3919**	8113	10038.68	2061.6
				FlowShop				
20_5_01	20	**1278**	**1278**	0	**1278**	**1278**	1283.8	7.0
20_5_02	20	**1359**	1359.2	0.4	**1359**	1360	1360.4	1.2
20_10_01	20	1583	1584.9	2.9	**1582**	1600	1606.1	6.6
20_10_02	20	1660	1666.4	3.1	**1659**	1675	1689.9	9.2
50_5_01	50	**2724**	**2724**	0	**2724**	**2724**	**2724**	0
				QAP				
Bur26a	26	**5426670**	5427870	1784	**5426670**	54322537	5435020	1460.8
tai50a	50	5067098	5074390	6361.6	**4938796**	5241678	5280400	27268.1
lipa40a	40	31645	31857.3	72.6	**31538**	32034	32052.7	11.3
sko100a	100	152560	153183	408.4	**152002**	155372	156912	1021.3
wil100a	100	274034	274553	365.7	**273038**	275750	278877	1335.3

2. H. H. Hoos, Autonomous Search, Springer Verlag, 2012, Ch. Automated Algorithm Configuration and Parameter Tuning, pp. 37–71.
3. J. Maturana, A. Fialho, F. Saubion, M. Schoenauer, F. Lardeux, M. Sebag, Autonomous Search, Springer Verlag, 2012, Ch. Adaptive Operator Selection and Management in Evolutionary Algorithms, pp. 161–190.
4. E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Handbook of Meta-heuristics, Kluwer, 2003, Ch. Hyper-heuristics: An Emerging Direction in Modern Search Technology, pp. 457–474.
5. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. Woodward, A Classification of Hyper-heuristic Approaches, Vol. 146 of International Series in Operations Research &amp; Management Science, Springer US, 2010, pp. 449–468.
6. H. Hoos, T. Stützle, Stochastic Local Search: Foundations & Applications, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
7. M. Deza, T. Huang, Metrics on permutations, a survey, Journal of Combinatorics, Information and System Sciences 23 (1998) 173–185.
8. T. Schiavinotto, T. Stützle, A review of metrics on permutations for search landscape analysis, Computers And Operations Research 34 (10) (2007) 3143 – 3153.
9. N. Veerapen, J. Maturana, F. Saubion, An exploration-exploitation compromise-based adaptive operator selection for local search, in: Genetic and Evolutionary Computation Conference, GECCO, ACM, 2012, pp. 1277–1284.

10. M.-E. Marmion, C. Dhaenens, L. Jourdan, A Fitness Landscape Analysis for the Permutation Flowshop Scheduling Problem, in: International Conference on Metaheuristics and Nature Inspired Computing (META 2010), Djerba, Tunisie, 2010. URL http://hal.inria.fr/inria-00523213
11. C. Praeger, Finite primitive permutation groups: A survey, in: L. Kovács (Ed.), Groups—Canberra 1989, Vol. 1456 of Lecture Notes in Mathematics, 1990, pp. 63–84.
12. T. C. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econometrica (25) (1957) 53–76.
13. R. A. Dudek, S. S. Panwalkar, M. L. Smith, The lessons of flowshop scheduling research, Operations Research 40 (1) (1992) 7–13.
14. G. Reinelt, Tsplib—a traveling salesman problem library, ORSA journal on computing 3 (4) (1991) 376–384.
15. E. Burke, M. Hyde, G. Ochoa, Cross-domain heuristic search challenge (2011). URL http://www.asap.cs.nott.ac.uk/external/chesc2011/