On the Design of a Partition Crossover for the Quadratic Assignment Problem

Omar Abdelkafi, Bilel Derbel, Arnaud Liefooghe, Darrell Whitley

On the Design of a Partition Crossover for the Quadratic Assignment Problem

Omar Abdelkafi[1], Bilel Derbel[1], Arnaud Liefooghe[2], and Darrell Whitley[3]

[1] Univ. Lille, CNRS, Centrale, Inria, UMR 9189 - CRIStAL, F-59000 Lille, France
omar.abdelkafi@univ-lille.fr, bilel.derbel@univ-lille.fr
[2] JFLI – CNRS IRL 3527, University of Tokyo, 113-0033 Tokyo, Japan
arnaud.liefooghe@univ-lille.fr
[3] Colorado State University, USA
whitley@cs.colostate.edu

Abstract. We conduct a study on the design of a partition crossover for the QAP. On the basis of a bipartite graph representation, we propose to recombine the unshared components from parents, while enabling their fast evaluation using a preprocessing step for objective function decomposition. Besides a formal description and complexity analysis of the proposed crossover, we conduct an empirical analysis on its relative behavior using a number of large-size QAP instances, and a number of baseline crossovers. The proposed operator is shown to have a relatively high intensification ability, while keeping execution time relatively low.

1 Introduction

One of the key ingredients in the success of evolutionary algorithms is the design of effective and efficient crossover operators. In the context of gray-box optimization, problem-specific properties can help in designing dedicated operators. This is the case of *partition crossovers*, developed for a number of combinatorial problems (e.g., TSP [18], SAT [2], NK-landscapes [17]), and allowing to efficiently explore large search spaces. The term *partition* crossover is here used in a general sense, to render the idea of decomposing the variables according to their values in the parents, and then recombining them in such a way that the best improving offspring can be computed efficiently. In other words, a partition crossover is based on the idea of *optimal* recombination of two parents, which requires to find the best possible offspring among all possible ones, while fulfilling the genotype inheritance principle. As such, two issues must be considered.

Firstly, one should specify the recombination mechanism allowing gene transmission while fully exploring the improvement potential of parents. In particular, the shared genes from parents are kept identical and the other genes are to be properly mixed. Secondly, since the set of possible offspring underlying such a process is typically huge, one must rely on some properties to compute the best offspring in the most efficient manner. For instance, the so-called k-bounded binary problems [17] can be decomposed as a linear combination of sub-functions of at most k variables. This guarantees that the contributions of non interacting variables to the global fitness are additive, and hence, the choice of the optimal

gene sequence when performing recombination can be performed greedily in an efficient manner. Another example is TSP [18], where the cost of the global tour is additive with respect to the length of different sub-paths sharing the same cities, but in different order, in the parent solutions.

In this paper, we are interested in the Quadratic Assignment Problem (QAP) [4, 7, 11, 13], for which no partition crossover has been developed so far, despite the broad range of dedicated algorithms. A main difficulty comes from the definition of its objective function, where the contribution of every single variable is sensitive to all other variables. Hence, it becomes a challenging issue to design a decomposition process which enables both parent recombination and fast offspring evaluation. This is precisely the aim of our work, and our contribution is to be viewed as a first step towards the design of an efficient and effective partition crossover operator for the QAP. More specifically, we rely on an intuitive bipartite graph representation for decomposition. We then show how recombination and offspring evaluation can be performed on that basis. Moreover, we conduct an empirical study rendering the performance of the designed crossover comparatively to existing ones, either by itself, or when combined with a fast local search process. Our analysis shows that the proposed crossover has a high intensification power while keeping execution time relatively low.

The paper is organized as follows. In Section 2, we introduce the QAP and review some existing crossovers. In Section 3, we describe our main contribution towards the design a partition crossover for the QAP. In Section 4, we report our experimental findings. In Section 5, we conclude the paper.

2 Background

2.1 Problem Definition

The Quadratic Assignment Problem (QAP) [4, 13] aims at assigning n facilities I, to n locations J. Let f_{hi} be the flow between facilities h and i, and d_{sj} be the distance between locations s and j. The objective is to minimize the sum of the products between flows and distances, such that each facility is assigned to exactly one location. The solution space can be defined as the set Π of permutations of $\{1, \ldots, n\}$. Given a permutation solution π, the i^{th} element $\pi(i)$ corresponds to assigning facility i to location $\pi(i)$. The QAP is then stated as:

$$\arg\min_{\pi \in \Pi} \sum_{h \in I} \sum_{i \in I} d_{\pi(h)\pi(i)} \, f_{hi} \tag{1}$$

The QAP is NP-hard [13], and is considered as one of the most difficult problems from combinatorial optimization. As such, we have to rely on heuristic approaches such as stochastic local search and evolutionary algorithms [7].

2.2 Representative Crossover Operators

In this paper, we are interested in designing efficient and effective crossover operators. Among the large number of hybrid genetic algorithms for the QAP; see, e.g., [6,9,10,19], crossover appears to be a crucial component. In this respect,

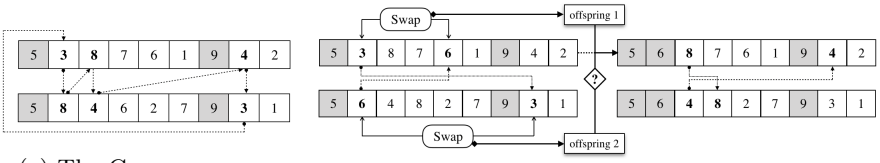
we can distinguish two families of crossovers [11]. In *half-structured* crossovers, the offspring preserve only a part of the parent genes, whereas the remaining part is typically generated at random. In *fully-structured* crossovers, the offspring genes are obtained by preserving the ones from parents. In our work, we consider four usual and representative baseline crossovers, two from the first family, and two from the second one. We first describe the half-structured crossovers, namely OPX and UX, and then the fully structured ones, namely CX and SPX. Notice that these crossovers can also find applications in other problems such as TSP [5].

The OPX crossover. This is a standard one-point crossover, e.g., [6]. Given two parent permutations, a random point is selected to define two parts for each parent. A new offspring is obtained by first preserving the locations from the first parent up to the chosen random point. The remaining part is filled by copying the elements from the second part of the second parent, excluding those that were already copied from the first parent. This may lead to the situation where some facilities are not assigned to any locations. The offspring is hence complemented at random, using the remaining locations that were not yet included.

The UX crossover. This is a standard uniform crossover operator, e.g., [16]. Some locations are selected at random following a Bernoulli distribution with parameter 1/2. The locations occupied by the selected facilities in the first parent are copied. The locations occupied by the remaining facilities in the second parent are also copied unless they were already included from the first parent. The offspring is complemented by randomly assigning the missing locations.

The CX crossover. This is the so-called cycle crossover [8,9,12]. For clarity, we consider the example of Fig. 1a. All shared assignments are copied, i.e., facilities 1 and 7 assigned to locations 5 and 9. The crossover starts iterating from the first different facility assignment, which is facility 2 assigned to location 3 in the *first* parent. Looking at the *second* parent, facility 2 is assigned to location 8. This location is occupied by facility 3 in the *first* parent. Similarly, facility 3 is assigned to location 4 in the *second* parent, and so on. A cycle – alternating between the same set of locations in a different order on both parents – is then detected when arriving to facility 8 assigned to location 3 in the second parent. Hence, one parent is selected at random and the so-computed locations are preserved in the offspring. This procedure is repeated until all facilities are assigned.

The SPX crossover. This is the so-called Swap Path Crossover [1, 3]. The crossover is based on iteratively swapping unshared locations. In Fig. 1b, it starts with facility 2 assigned to location 3 (resp. 6) in the first (resp. second) parent. In the first (resp. second) parent, location 6 (resp. 3) is occupied by facility 5 (resp. 8). Hence, a swap is performed between locations 2 and 5 in the first parent, *as well as* between locations 2 and 8 in the second parent. Two offspring are obtained, respectively to the first and second parent. Then, the best of the two replaces its corresponding parent, say the first one as in Fig 1b. Notice that after this iteration, the two new parents have one more location in common. The same is then repeated iteratively until both parents become the same. The output offspring is the best ever created during all iterations.



(a) The Cx crossover. (b) First steps of the Spx crossover.

Fig. 1: Illustration of the Cx and Spx crossovers.

3 A Partition Crossover for the QAP

A partition crossover is based on the idea of decomposing the evaluation function as well as the set of variables during recombination. This decomposition should enable to construct and to evaluate offspring using partial evaluations. The fastest the evaluation step, the fastest the exploration of a large number of offspring, eventually leading to an improving one, hence exploring the so-called dynastic potential of parents at best. It should be clear that none of the previously-described crossovers fulfill this requirement, although Cx and Spx attempt to construct an offspring based on the idea of gene transmission.

In the following, we aim at designing a new partition crossover for the QAP. We start by introducing some notations, then we describe a decomposition process for the QAP, and the underlying recombination (Proposition 1) and function evaluation process (Corollary 1), in a formal, but intuitive, manner.

3.1 Recombination based on a Bipartite Graph Representation

Following standard notations from graph theory, a bipartite graph G is a graph whose nodes can be divided into two disjoint sets $U(G)$ and $V(G)$, such that an edge in $E(G)$ can only connect a node in U to one in V. Given a subset of nodes $Q \subseteq U$ and $R \subseteq V$, we denote by $G[Q, R]$ the subgraph induced by $Q \cup R$ in G.

Let π be a permutation solution for the QAP. Let us define the bipartite graph whose nodes sets are respectively the set of facilities I and locations J, and where every facility i is connected to its unique location j according to permutation π. This corresponds in fact to a very natural representation of a feasible assignment of the facilities to the locations; see Fig. 2.

Definition 1. *Let $G_\pi = (I, J, E)$ be the bipartite graph such that $E(G_\pi) = \{(i, j) \mid \pi(i) = j\}$.*

Let us now consider two permutation solutions π_1 and π_2 which will play the role of parents for our target crossover. Let $\widetilde{I}$ (resp., $\overline{I}$) be the set of facilities that are assigned (resp., not assigned) to the same locations, denoted $\widetilde{J}$ (resp., $\overline{J}$).

Definition 2. *Let $\widetilde{I} = \{i \in I \mid \pi_1(i) = \pi_2(i)\}$ and $\overline{I} = I \setminus \widetilde{I}$. Then, let $\widetilde{J} = \{\pi_1(i) \mid i \in \widetilde{I}\}$, and $\overline{J} = J \setminus \widetilde{J}$.*

Notice that for any facility $i \in \overline{I}$, the corresponding locations $\pi_1(i)$ and $\pi_2(i)$ are different, and both belong to $\overline{J}$. Let us now define the bipartite graph $G_{\pi_1 \pi_2}$ obtained by merging the edges of G_{π_1} and G_{π_2}; see Fig. 2.

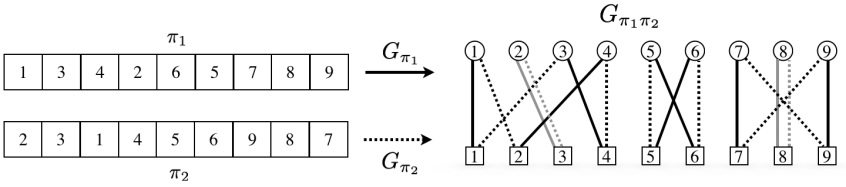


Fig. 2: Illustration of a bipartite graph representation. Solid (resp. dashed) edges are with respect to G_{π_1} (resp. G_{π_2}). We have $\widetilde{I} = \{2, 8\}$, $\widetilde{J} = \{3, 8\}$ and the corresponding edges in gray. There are $k = 3$ connected components C_1, C_2 and C_3 in $G_{\pi_1\pi_2}[\overline{I}, \overline{J}]$, with $U(C_1) = \{1, 3, 4\}$, $U(C_2) = \{5, 6\}$, and $U(C_3) = \{7, 9\}$.

Definition 3. *Let* $G_{\pi_1\pi_2} = (I, J, E(G_{\pi_1}) \cup E(G_{\pi_2}))$

Let us focus on the *connected components* of the so-obtained graph. First, we have that every facility $i \in \widetilde{I}$ is connected, by exactly two edges in $G_{\pi_1\pi_2}$, to a unique location $j = \pi_1(i) = \pi_2(i) \in \widetilde{J}$, and j is not connected to any other facility. Hence, this implies exactly one connected component with these two nodes connected by two parallel edges. Apart from such components (in $\widetilde{I} \cup \widetilde{J}$), the other components of interest connect nodes in $\overline{I}$ to nodes in $\overline{J}$. Then,

Definition 4. *Let* k *be the number of connected components in* $G_{\pi_1\pi_2}[\overline{I}, \overline{J}]$*, and let* $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$ *be the set of these connected components.*

Notice that $k = 0$ iff $\pi_1 = \pi_2$. In the following, we assume that $\pi_1 \neq \pi_2$, and hence $k \geq 1$. By definition, we also have that for every $\ell \in \{1, \ldots, k\}$, C_ℓ is a bipartite graph whose edges form a cycle that are alternatively in $E(G_{\pi_1})$ and in $E(G_{\pi_2})$. An offspring can hence be constructed by: (i) preserving the shared assignments of facilities in $\widetilde{I}$, and (ii) choosing for every set of facilities $U(C_\ell)$, $\ell \in \{1, \ldots, k\}$, the locations they are connected to *either* in G_{π_1} (first parent) *or* in G_{π_2} (second parent). More formally, let $m : \{1, \ldots, k\} \mapsto \{1, 2\}$ be an *arbitrary* mapping function. Such a mapping can be used to decide which of π_1 or π_2 to consider when choosing the locations of $U(C_\ell)$ for every ℓ, i.e., if $m(\ell) = 1$ then use π_1, otherwise use π_2. This is stated in the following proposition summarizing the proposed crossover recombination mechanism.

Proposition 1. *Given two permutations* π_1 *and* π_2 *and an arbitraty mapping function* $m : \{1, \ldots, k\} \mapsto \{1, 2\}$*,* $\overset{\otimes}{\pi}$ *as defined in the following is a feasible permutation offspring.*

- $\forall i \in \widetilde{I}$, $\overset{\otimes}{\pi}(i) = \pi_1(i) = \pi_2(i)$
- $\forall i \in \overline{I}$, $\overset{\otimes}{\pi}(i) = \pi_{m(\ell_i)}(i)$*, where* $\ell_i \in \{1, \ldots, k\}$ *is such that* $i \in U(C_{\ell_i})$

The previous proposition is to recall the Cx crossover, where a cycle corresponds to a connected component in our formalism. However, we consider to explore not solely one random offspring, but the whole set of offspring solutions that can be constructed by Proposition 1 to find the best possible one. Since, the recombination process is fully and uniquely determined by the choice of the

mapping function m, we have to consider all possible mappings. Obviously, there exist 2^k possibilities (including parents) leading to as much possible offspring. Since evaluating one offspring from scratch costs $\Theta(n^2)$, a naive approach to compute the fitness values of these possible offspring has a complexity of $\Theta(2^k \cdot n^2)$, which can be prohibitive. In the next section, we show how to reduce it.

3.2 Decomposition of the Evaluation Function

Following the previous notations, let $m : \{1, \ldots, k\} \mapsto \{1, 2\}$ be an arbitrary mapping function, and let us define for every ℓ and ℓ' in $\{1, \ldots, k\}$:

$$\begin{aligned} Q_{\ell\ell'} &= \sum_{h \in U(C_{\ell'})} \sum_{i \in U(C_\ell)} d_{\pi_{m(\ell')}(h)\pi_{m(\ell)}(i)}\, f_{hi} \; ; \; Q_\ell = \sum_{h \in \widetilde{I}} \sum_{i \in U(C_\ell)} d_{\pi_1(h)\pi_{m(\ell)}(i)}\, f_{hi} \\ R &= \sum_{h \in \widetilde{I}} \sum_{i \in \widetilde{I}} d_{\pi_1(h)\pi_1(i)}\, f_{hi} \qquad\qquad\qquad\quad ; \; Q'_\ell = \sum_{i \in \widetilde{I}} \sum_{h \in U(C_\ell)} d_{\pi_{m(\ell)}(h)\pi_1(i)}\, f_{hi} \end{aligned}$$

where $Q_{\ell\ell'}$ represents the contribution of facilities in C_ℓ w.r.t. the facilities in another connected component $C_{\ell'}$. Similarly, Q_ℓ (resp. Q'_ℓ) represents the contribution of facilities in C_ℓ w.r.t. facilities having the same locations in both parents π_1 and π_2. Finally, R represents the pairwise contribution of the facilities that have the same locations in π_1 and π_2. The following proposition shows that the QAP objective function can be decomposed using these contributions.

Proposition 2. *Let $\overset{\otimes}{\pi}$ be a permutation offspring as defined in Proposition 1. Then,*

$$f(\overset{\otimes}{\pi}) = R + \sum_{\ell=1}^{k} \left(Q_\ell + Q'_\ell + \sum_{\ell'=1}^{k} Q_{\ell\ell'} \right)$$

Proof. By decomposing the facilities w.r.t. $\widetilde{I}$, $\overline{I}$, we get:

$$\begin{aligned} f(\overset{\otimes}{\pi}) = & \sum_{h \in I} \sum_{i \in I} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} \;=\; \sum_{h \in I} \sum_{i \in \overline{I}} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} + \sum_{h \in I} \sum_{i \in \widetilde{I}} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} \\ = & \sum_{h \in \overline{I}} \sum_{\ell=1}^{k} \sum_{i \in U(C_\ell)} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} + \sum_{h \in \widetilde{I}} \sum_{\ell=1}^{k} \sum_{i \in U(C_\ell)} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} \\ & + \sum_{h \in \overline{I}} \sum_{i \in \widetilde{I}} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} + \sum_{h \in \widetilde{I}} \sum_{i \in \widetilde{I}} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} \\ = & \sum_{\ell'=1}^{k} \sum_{h \in U(C_{\ell'})} \sum_{\ell=1}^{k} \sum_{i \in U(C_\ell)} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} + \sum_{h \in \widetilde{I}} \sum_{\ell=1}^{k} \sum_{i \in U(C_\ell)} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} \\ & + \sum_{\ell'=1}^{k} \sum_{h \in U(C_{\ell'})} \sum_{i \in \widetilde{I}} d_{\overset{\otimes}{\pi}(h)\overset{\otimes}{\pi}(i)}\, f_{hi} + \sum_{h \in \widetilde{I}} \sum_{i \in \widetilde{I}} d_{\pi_1(h)\pi_1(i)}\, f_{hi} \\ = & \sum_{\ell'=1}^{k} \sum_{\ell=1}^{k} \sum_{h \in U(C_{\ell'})} \sum_{i \in U(C_\ell)} d_{\pi_{m(\ell')}(h)\pi_{m(l)}(i)}\, f_{hi} \\ & + \sum_{\ell=1}^{k} \sum_{h \in \widetilde{I}} \sum_{i \in U(C_\ell)} d_{\pi_1(h)\pi_{m(\ell)}(i)}\, f_{hi} + \sum_{\ell'=1}^{k} \sum_{i \in \widetilde{I}} \sum_{h \in U(C_{\ell'})} d_{\pi_{m(\ell')}(h)\pi_1(i)}\, f_{hi} + R \end{aligned}$$

$$= \sum_{\ell'=1}^{k}\sum_{\ell=1}^{k} Q_{\ell\ell'} + \sum_{\ell=1}^{k} Q_{\ell} + \sum_{\ell'=1}^{k} Q'_{\ell'} + R \qquad \square$$

As a result, we obtain the following corollary which follows from the fact that the contributions appearing in the decomposition of Proposition 2 can *only* have a *constant* number of values, for all possible choices of the mapping m.

Corollary 1. *The whole set of offspring that can be generated by Proposition 1 can be explored and evaluated in $O(n^2 + k^2 \cdot 2^k)$ time.*

Proof. Let us first notice that computing the components of the bipartite graph can be done in $\Theta(n)$ time. Over all the possible mapping functions m, $(m(\ell), m(\ell'))$ can only take 4 different values, namely, $(1,1)$, $(1,2)$, $(2,1)$ and $(2,2)$. Hence, for every *fixed* values of ℓ and ℓ', $Q_{\ell\ell'}$ can only take 4 possible values. These four values depend solely on π_1 and π_2. Therefore, there can only be $\Theta(k^2)$ possible values for $Q_{\ell\ell'}$ over all ℓ and ℓ', and all possible choices of m. All of these $\Theta(k^2)$ values can be precomputed by a simple preprocessing step. Since by definition C_ℓ and $C_{\ell'}$ do not share any nodes for every $\ell \neq \ell'$, this preprocessing step takes obviously $O(n^2)$ time. Similarly, there are only $\Theta(k)$ possible values for Q_ℓ and Q'_ℓ over all possible values of ℓ, and all possible choices of m. Hence, they can also be precomputed in $\Theta(n^2)$ time. Finally, R does not depend on m, and can also be precomputed in $O(n^2)$ time. To summarize, all possible values taken by $Q_{\ell\ell'}$, Q_ℓ, $Q_{\ell'}$, and R can be precomputed in $\Theta(n^2)$ time and stored in $\Theta(k^2)$ memory space before even any specific choice of the mapping function m is made.

Now, let us consider a specific choice for the mapping function m, which fully determines an offspring permutation $\overset{\otimes}{\pi}$ according to Proposition 1. Then, according to Proposition 2, and given the contributions $Q_{\ell\ell'}$, Q_ℓ, $Q_{\ell'}$, and R were already precomputed by the previous discussion, it takes $\Theta(k^2)$ time to compute $f(\overset{\otimes}{\pi})$. The corollary follows since there are 2^k possible mapping functions m. $\square$

4 Experimental Analysis

In the rest of the paper, we provide an empirical analysis of the designed partition crossover, denoted by PX.

4.1 Experimental Setup

We consider the following 8 QAP instances from the literature[4] [14,15]: tai343e0i with $i \in \{0, \ldots, 7\}$. They have been selected due to their challenging size of $n = 343$ facilities and locations, which is rarely addressed in the literature. We consider the following scenarios.

Scenario #1. The goal of this scenario is the study the relative ability of the PX crossover to find an improving offspring. As depicted in the high-level template of Experiment 1, we consider two settings where the initial parents are

[4] http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html

Experiment 1: Pseudo-code of the first experimental scenario

Let π_1 and π_2 be either (i) two random solutions or (ii) two local optima;
Apply crossover on π_1 and π_2 to obtain an offspring $\overset{\otimes}{\pi}$;
Apply a local search with $\overset{\otimes}{\pi}$ as initial solution to obtain $\overset{\otimes}{\pi}'$;

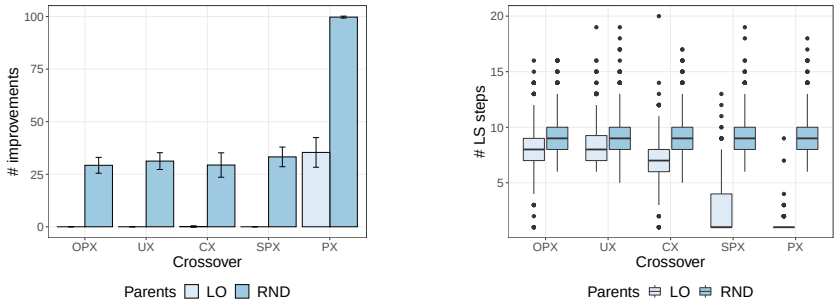
Experiment 2: Pseudo-code of the second experimental scenario

Input: G: maximum number of generations; $p \in [0, 1]$: local search ratio;
Generate an initial random population P;
for $g = 1$ *to* G **do**
    Apply crossover on two randomly selected parents π_1 and π_2 to obtain $\overset{\otimes}{\pi}$;
    if $p < rand(0, 1)$ **then**
        Apply local search with $\overset{\otimes}{\pi}$ as initial solution to obtain $\overset{\otimes}{\pi}'$ and let $\overset{\otimes}{\pi} = \overset{\otimes}{\pi}'$;
    Replace the oldest individual of the population with $\overset{\otimes}{\pi}$;

either (i) random solutions, or (ii) local optima. For the latter case, we run a basic hill climbing local search using the standard swap neighborhood to construct the initial local optima. Starting from a random permutation, the best swap move is performed until no improvement is possible. The local search is executed as much times as needed to find as much different local optima as needed. All competing crossovers are applied 100 times using 100 pairs of different initial parents. Once an offspring has been generated by crossover, we also consider to check if it is a local optimum w.r.t. the swap neighborhood by running the local search again.

Scenario #2. The goal of this scenario is to study the relative performance of the Px crossover when plugged into a simple evolutionary algorithm. As depicted in the high level template of Experiment 2, we consider a hybrid evolutionary algorithm embedding the swap-based local search discussed previously. In each generation, a new offspring is generated by performing crossover followed by a local search with probability p. We consider a simple random parent selection and a non-elitist replacement where the newly generated offspring replaces the oldest individual. The value of p is chosen in the set $\{0, 0.05, 0.2\}$. This allows us to study the relative behavior of crossover using a variable amount of local search, ranging from no local search at all ($p = 0$), to a small ($p = 0.05$) and a high ($p = 0.2$) amount. For each configuration, 10 independent runs are performed with a maximum number of generations $G = 1\,000$ and a population size of n.

CPU running time. Let us finally notice that we manage to analyze the CPU execution time. It is hence important to recall that the implementations of all algorithms were optimized as much as possible. In particular, computing the best move in a swap-based local search can be performed in a very efficient manner for the QAP. In fact, this can be done in an incremental manner on the basis of the current solution [14, 15]. For the sake of fairness when analyzing execution time, this well-established and important consideration is carefully implemented



(a) Number of improving offspring (b) Number of local search moves

Fig. 3: Results from Experiment 1. Results are over all eight instances.

in all experiments. Moreover, when running the Px crossover, we restrict the maximum number of explored offspring to at most 2^{15}, which means that when the number of connected components $k > 15$, not all possible offspring from Proposition 1 are explored. All algorithms are implemented in C++ and run on an Intel Xeon(R) CPU E3-1505M v6 3.00GHz.

Let us finally notice that more advanced settings including other QAP instances and algorithms, as well as finely tuned operators/components using automated algorithm configuration methods, etc, are left for future investigations.

4.2 Experimental Analysis and Results

We start our analysis by reporting our findings from Experiment 1 in Fig. 3.

Crossover improvement ratio. In Fig. 3a, we can see that the Px crossover has a significantly higher improvement rate. Using random parents, it can produce an improving offspring in almost 100% of the cases, whereas all other considered crossovers have a ratio of around 28%. Using local optimal parents, the Px is still able to find improving offspring in around 30% of the cases. This is to contrast with the other crossovers that fail in almost all cases. This first set of observations shows that the Px crossover has a relatively high intensification ability since it is even able to improve over local optima.

Local optimality. In Fig. 3b, we further show the number of moves performed by the local search initialized with the generated offspring (in line 3), i.e., 0 moves means that the offspring is a local optima w.r.t. the swap neighborhood. For all crossovers except Px, the local search improves the constructed offspring independently of using random or local optima parents. This means that these crossovers are more diversification-oriented, since even when using local optimal parents, they are likely to produce an inferior offspring that can be improved by local search. Hence, we can think about these crossovers as acting in a perturbative manner. The situation is completely different for the Px crossover. On one side, with random parents, where the offspring improvement ratio was found to be almost 100%, the local search can still find improvements. On the other side, with local optimal parents, where the offspring improvement ratio is about 30%,

Table 2: Ranks of crossovers (a lower rank is better). The first (resp. second) part is with respect to solution quality (resp. execution time). For each value of parameter p (local search ratio), a rank c indicates that the corresponding crossover was found to be significantly outperformed by c other ones w.r.t. a Wilcoxon statistical test at a significance level of 0.05. Rows are w.r.t. instances.

	$p = 0$					$p = 0.05$					$p = 0.2$				
$\#Ins$	OPX	UX	CX	SPX	PX	OPX	UX	CX	SPX	PX	OPX	UX	CX	SPX	PX
Mean average deviation to the best (in subscript)															
1	2_{810}	2_{809}	2_{808}	1_{796}	0_{710}	$0_{0.38}$	$0_{0.39}$	$0_{0.37}$	$0_{0.37}$	$0_{0.37}$	$0_{0.26}$	$0_{0.24}$	$0_{0.30}$	$1_{0.32}$	$1_{0.34}$
2	2_{105}	2_{105}	2_{105}	1_{103}	0_{94}	$0_{0.04}$	$0_{0.04}$	$0_{0.03}$	$0_{0.04}$	$1_{0.05}$	$0_{0.02}$	$0_{0.03}$	$0_{0.02}$	$0_{0.03}$	$0_{0.03}$
3	2_{115}	2_{115}	2_{115}	1_{113}	0_{102}	$0_{0.04}$	$0_{0.03}$	$0_{0.03}$	$0_{0.04}$	$1_{0.05}$	$0_{0.02}$	$0_{0.02}$	$0_{0.02}$	$3_{0.03}$	$3_{0.03}$
4	2_{96}	2_{96}	2_{96}	1_{95}	0_{84}	$0_{0.04}$	$0_{0.04}$	$0_{0.04}$	$0_{0.04}$	$1_{0.05}$	$0_{0.03}$	$0_{0.02}$	$0_{0.02}$	$1_{0.03}$	$2_{0.03}$
5	2_{107}	2_{107}	2_{108}	1_{106}	0_{96}	$0_{0.04}$	$0_{0.03}$	$0_{0.03}$	$0_{0.04}$	$1_{0.04}$	$0_{0.02}$	$0_{0.02}$	$0_{0.02}$	$1_{0.03}$	$2_{0.03}$
6	2_{123}	2_{123}	2_{124}	1_{121}	0_{110}	$0_{0.06}$	$0_{0.05}$	$0_{0.05}$	$0_{0.06}$	$1_{0.06}$	$0_{0.04}$	$0_{0.04}$	$0_{0.03}$	$1_{0.05}$	$1_{0.05}$
7	2_{115}	2_{115}	2_{115}	1_{113}	0_{101}	$0_{0.04}$	$0_{0.04}$	$0_{0.04}$	$0_{0.05}$	$0_{0.04}$	$0_{0.03}$	$0_{0.03}$	$0_{0.04}$	$0_{0.04}$	$0_{0.03}$
8	2_{110}	2_{110}	2_{110}	1_{108}	0_{96}	$0_{0.04}$	$0_{0.03}$	$0_{0.03}$	$0_{0.03}$	$1_{0.04}$	$0_{0.02}$	$0_{0.02}$	$0_{0.01}$	$3_{0.03}$	$3_{0.03}$
Mean CPU execution time (in subscript)															
1	$0_{0.61}$	$1_{0.62}$	$2_{0.75}$	4_{286}	$3_{3.39}$	0_{194}	0_{198}	0_{196}	4_{478}	0_{185}	1_{705}	1_{714}	1_{676}	4_{870}	0_{502}
2	$0_{0.61}$	$1_{0.62}$	$2_{0.72}$	4_{287}	$3_{3.44}$	0_{202}	0_{202}	0_{199}	4_{475}	0_{187}	2_{698}	1_{697}	1_{662}	4_{857}	0_{494}
3	$0_{0.60}$	$1_{0.63}$	$2_{0.70}$	4_{286}	$3_{3.38}$	0_{210}	0_{208}	0_{206}	4_{481}	0_{195}	1_{712}	1_{730}	1_{682}	4_{876}	0_{510}
4	$0_{0.60}$	$1_{0.62}$	$2_{0.75}$	4_{287}	$3_{3.41}$	0_{193}	1_{198}	0_{194}	4_{468}	0_{175}	1_{724}	1_{728}	1_{693}	4_{880}	0_{513}
5	$0_{0.60}$	$1_{0.62}$	$2_{0.74}$	4_{287}	$3_{3.39}$	0_{189}	0_{187}	0_{183}	4_{463}	0_{172}	1_{681}	2_{684}	1_{643}	4_{842}	0_{488}
6	$0_{0.61}$	$1_{0.62}$	$2_{0.72}$	4_{286}	$3_{3.39}$	0_{209}	0_{212}	0_{206}	4_{481}	0_{193}	1_{714}	1_{725}	1_{690}	4_{876}	0_{515}
7	$0_{0.61}$	$0_{0.62}$	$2_{0.69}$	4_{286}	$3_{3.37}$	0_{207}	0_{206}	0_{205}	4_{477}	0_{193}	1_{697}	2_{701}	1_{670}	4_{875}	0_{505}
8	$0_{0.61}$	$1_{0.62}$	$2_{0.70}$	4_{287}	$3_{3.40}$	0_{221}	0_{223}	0_{219}	4_{494}	0_{200}	1_{715}	1_{734}	1_{693}	4_{890}	0_{521}

the local search cannot find improvements, which indicates that the produced offspring is also likely to be a local optimum. This means that the PX crossover has some ability to act as a tunneling operator allowing to jump from two local optima to a new improving local optimum (with around 30% success rate).

Solution quality *vs* execution time. In the following, we report our findings from Experiment 2. In Table 2, we rank the different algorithms according to: (i) the quality of the best solution that the evolutionary algorithm is able to find during its execution, and (ii) the total CPU execution time. The situation is clearly different depending on whether a local search is used or not.

When performing only crossover with no local search ($p = 0$), the PX crossover is able to find substantially better solutions, followed by SPX, and then by OPX, UX and CX, which do not show any significant difference statistically. Without surprise, these crossovers run however faster than PX and SPX. Notice here the extremely high cost of SPX. Interestingly, the proposed PX crossover implies a relatively reasonable increase in terms of running time (about 3.5 seconds) compared against SPX (about 287 seconds).

When performing crossover with local search ($p \in \{0.05, 0.2\}$), we can first see that, compared to not using local search at all, all variants can find much better solutions, while having a higher execution time. Interestingly, the PX and

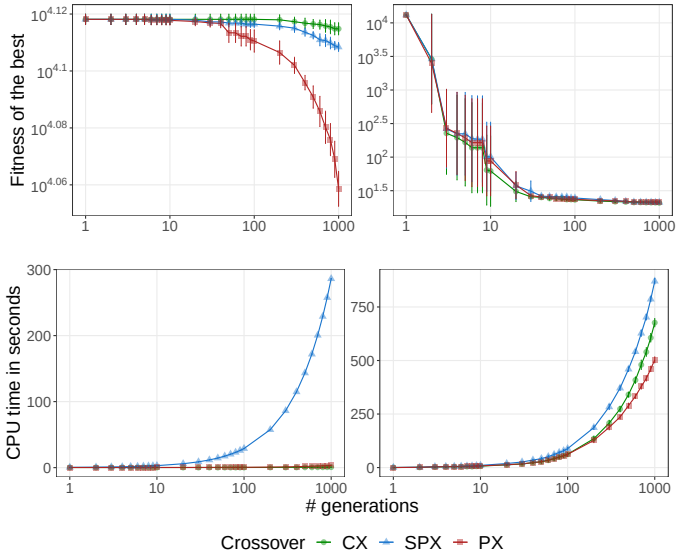


Fig. 4: Experiment 2 on first instance. Top: convergence profile. Bottom: execution time. Left: $p = 0$ (no local search). Right: $p = 0.2$.

Spx crossovers (which were previously found to perform better than Opx, Ux, and Cx) now provide slightly worst solutions. On the other side, when analyzing the execution time, we found that using the Px crossover is significantly faster than all other variants, with at least 25% CPU time gain; e.g., for $p = 0.2$, the average execution time over all configurations is of 506 seconds with Px, against 677, 706, 714, and 871 with Px, Cx, Opx, Ux, and Spx, respectively. As commented before, knowing that our swap-based hill climbing local search was carefully implemented using a state-of-the-art fast incremental evaluation procedure for finding the best move [14,15], such an observation might be surprising at first sight. However, it can be explained from two perspectives. Firstly, due to Corollary 1, performing the Px crossover is reasonably fast. Secondly, the Px crossover was previously found to have a relatively high intensification power. Thus, it is likely to produce a high-quality offspring, eventually being a local optima. Hence, it is more likely that the local search stops more quickly when attempting to improve the produced offspring. In this case, the cost of the local search is also reduced, hence leading to a decrease in the overall CPU time.

Finally, the previous results about solution quality and execution time are found to hold at any generation, independently of the configuration. This is illustrated in Fig. 4 rendering the convergence profile and the CPU execution time as a function of generations for the first QAP instance. This also confirms that the Px crossover is more intensification-oriented, and should be complemented by other diversification mechanisms when effectively integrated into more advanced evolutionary search processes.

QAP connected components. To complement our analysis, we provide further observations on the characteristics of the QAP and the relevance of the Px

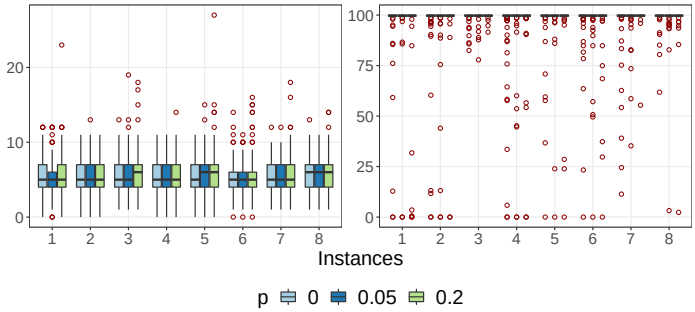


Fig. 5: Left: Number of connected components (k). Right: $\left|\overline{I}\right|/n$ in %.

crossover. More precisely, remember that given k connected components implied by the unshared facility assignments $\overline{I}$, the Px crossover is able to provide the best over 2^k possible offspring. In Fig. 5, we report the average value of k (Left), as well the percentage of unshared facility assignments (Right), i.e., $\left|\overline{I}\right|/n$ in %, over all generations and all considered Px runs. We can see that for all considered QAP instances, solutions contain very few shared facility assignments, i.e., less than 1.5%, and the value of k stays relatively low, i.e., 6 in average with the exception of a few outliers exceeding 10. This suggests that the exploration power of the Px crossover can be improved in different ways, since its time complexity guarantees (Corollary1) should be able to support the fast evaluation of much more offspring solutions. For instance, it would be interesting to investigate the splitting of existing connected components into smaller ones at the aim of processing, and hopefully finding, more improving offspring.

5 Conclusion

In this paper, we presented our first investigations on the design of a partition crossover for the QAP. The proposed recombination and evaluation process is proved to provide a reasonable trade-off between the intensification power and the running time complexity. Our empirical study provides first insights towards the design and integration of more powerful partition crossovers for the QAP. In particular, a future challenging issue is to investigate complementary decomposition techniques allowing to break the QAP bipartite graph into further smaller connected components, while maintaining a reasonable recombination and evaluation cost. In this respect, a promising idea would be to investigate the knowledge about the flow and distance values in order to identify the critical facilities and locations when performing decomposition. The challenge is then to guide the recombination process by identifying the most promising components to consider while keeping evaluation cost as low as possible.

Acknowledgments. The three first authors are supported by the French national research agency (ANR-16-CE23-0013-01) and the Research Grants Council of Hong Kong (RGC Project No. A-CityU101/16).

References

1. Ahuja, R.K., Orlin, J.B., Tiwari, A.: A greedy genetic algorithm for the quadratic assignment problem. Comput. Oper. Res. **27**(10), 917–934 (2000)
2. Chicano, F., Ochoa, G., Whitley, D., Tinós, R.: Enhancing partition crossover with articulation points analysis. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 269–276 (2018)
3. Glover, F.: Genetic algorithms and scatter search: unsuspected potentials. Statistics and Computing **4**(2), 131–140 (1994)
4. Koopmans, T.C., Beckmann, M.: Assignment problems and the location of economic activities. Econometrica **25**(1), 53–76 (1957)
5. Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review **13**(2), 129–170 (1999)
6. Lim, M.H., Yuan, Y., Omatu, S.: Efficient genetic algorithms using simple genes exchange localsearch policy for the quadratic assignment problem. Comput. Optim. Appl. **15**(3), 249–268 (2000)
7. Loiola, E.M., Maia, N.M., Boaventura-Netto, P.O., Hahn, P., Querido, T.: A survey for the quadratic assignment problem. European Journal of Operational Research **176**(2), 657 – 690 (2007)
8. Merz, P., Freisleben, B.: A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In: Proceedings of the Congress on Evolutionary Computation. pp. 2063–2070 (1999)
9. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Transactions on Evolutionary Computation **4**(4), 337–352 (2000)
10. Misevicius, A.: An improved hybrid genetic algorithm: new results for the quadratic assignment problem. Knowledge-Based Systems **17**(2), 65 – 73 (2004)
11. Misevicius, A., Kilda, B.: Comparison of crossover operators for the quadratic assignment problem. Information Technology and Control **34**(2) (2005)
12. Oliver, I.M., Smith, D.J., Holland, J.R.C.: A study of permutation crossover operators on the traveling salesman problem. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application. p. 224–230 (1987)
13. Sahni, S., Gonzalez, T.: P-complete approximation problems. Journal of the ACM **23**(3), 555–565 (1976)
14. Taillard, E.: Robust taboo search for the quadratic assignment problem. Parallel Computing **17**(4), 443 – 455 (1991)
15. Taillard, E.: Comparison of iterative searches for the quadratic assignment problem. Location Science **3**(2), 87 – 105 (1995)
16. Tate, D.M., Smith, A.E.: A genetic approach to the quadratic assignment problem. Computers & Operations Research **22**(1), 73 – 83 (1995)
17. Tinós, R., Whitley, D., Chicano, F.: Partition crossover for pseudo-boolean optimization. In: Proceedings of the ACM Conference on Foundations of Genetic Algorithms XIII. p. 137–149 (2015)
18. Tinós, R., Whitley, D., Ochoa, G.: Generalized asymmetric partition crossover (gapx) for the asymmetric tsp. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. p. 501–508 (2014)
19. Vázquez, M., Whitley, L.D.: A hybrid genetic algorithm for the quadratic assignment problem. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation. p. 135–142. San Francisco, CA, USA (2000)