Graph Matching with Distance-Preserving Crossover

Thomas Bärecke
Marcin Detyniecki
Databases and Machine Learning Department
Laboratoire d'Informatique de Paris 6
University Pierre and Marie Curie, Paris, 75005 France
marcin.detyniecki@lip6.fr

Graph models are fundamental to any kind of application on structured real-world problems. Any comparison between graphs by a graph distance measure requires the solution of the inexact graph matching problem, which constitutes a hard combinatorial optimization problem. An inexact matching problem includes in its formulation robustness to any type of perturbation, such as, for instance, noise, inherently present in real-world environments. In this paper, we introduce the concept of distance-preserving crossover operators for genetic algorithms for this task. For large graphs, our algorithm outperforms any state-of-the-art approximate algorithm—in particular, genetic algorithms with alternative crossover operators, which are to the best of our knowledge currently limited to no more than 50 nodes. We use a two-level local search heuristic to further enhance the results, pushing the limits to up to 300 nodes: a first local search step is directly integrated into the crossover operator; another one is applied independently during offspring generation.

Keywords: graph distance; evolutionary computation; genetic algorithm; inexact graph isomorphism; permutation encoding; crossover operator with local search

1. Introduction

Graphs are a universal tool for modeling structured entities. Their higher generality and flexibility, in comparison to the prevalent vectorial data model, comes with an increase in computational cost. For instance, general distance computations for feature vectors are linear in the number of dimensions; they become quadratic for string sequences [1] and exponential in the number of nodes for graphs [2]. In this paper, we specifically address the problem of comparing graphs in noisy environments. In mathematical terms, we propose a method for efficiently resolving the inexact graph matching problem in order to determine the distance between graphs.

https://doi.org/10.25088/ComplexSystems.31.2.219

Intuitively, comparing two graphs is a highly combinatorial problem because not only the vertices (also called nodes) have to be *matched* but also their relationships (called edges). Moreover, the solution of the graph isomorphism problem guarantees that not only are these elements between graphs equal, but their structure is the same.

Mathematically, an attributed relational graph G is defined over a vertex set $\mathcal{V}$. Each node is labeled with an attribute $a \in \mathcal{A}_{\mathcal{V}}$. Their relationships are labeled with edge attributes $e \in \mathcal{A}_{\mathcal{E}}$. A formal definition for a fully connected graph is given by [3]:

$$G = (\mathcal{V}, \alpha, \beta). \tag{1}$$

The edge set is implicitly given by $\mathcal{E} = \mathcal{V} \times \mathcal{V}$. The functions $\alpha : \mathcal{V} \to \mathcal{A}_{\mathcal{V}}$ and $\beta : \mathcal{V} \times \mathcal{V} \to \mathcal{A}_{\mathcal{E}}$ assign the attributes to nodes and edges; missing edges may be modeled using null labels. The attributes are application-dependent content descriptors.

As shown in Figure 1, the two vertices of the graphs are, respectively, alphabetically (i.e., a, b, c, d) and numerically (i.e., 1, 2, 3, 4, 5) named; the vertices' attributes are the colors (red, blue, white, green, black). In these simplified examples, the (labeled) graph matching problem consists of finding the function m so that the colors (labels) and their relationships correspond: $m(a) = 1$, $m(b) = 2$, $m(c) = 3$ and so on. To make it obvious that the problem is not only color matching related, we may think about the particular case when there are no labels, or to put it in other terms, when there is one single color for both graphs.

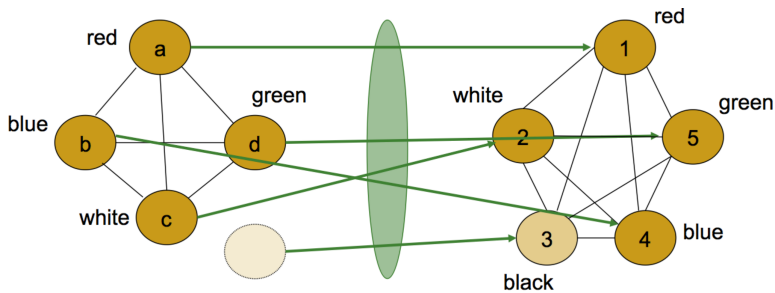

Figure 1. The graph matching problem answers the question, Are these two graphs equal in terms of content and structure?

There are many different forms of graph matching problems in the literature (see [4] for an overview focusing on complexity). The major research area is graph isomorphism, which is a decision problem aiming at finding a bijective correspondence m between the nodes of two unlabeled graphs $G = (\mathcal{V}, \mathcal{E})$ and $G' = (\mathcal{V}, \mathcal{E})$ such that

$(v_1, v_2) \in G \leftrightarrow (m(v_1), m(v_2)) \in G'$. A weaker form is graph homomorphism, where only the edges of G are matched by edges of G' but G' may contain additional nodes and edges, that is, $(v_1, v_2) \in G \rightarrow (m(v_1), m(v_2)) \in G'$. In this case, not all edges of G are matched. Graph monomorphism adds the constraint that m must be injective, meaning that all nodes and edges from G have an image in G'.

Finally, the one we are mostly concerned with is subgraph isomorphism, which searches a bijective mapping between all the nodes of the smaller of the two graphs and a node-induced subgraph of the same size from the other graph (see Figure 1). It differs from monomorphism in the fact that all edges must be preserved in both directions. From the point of view of complexity, it is important to note that subgraph isomorphism is proven NP-complete [4], while for graph isomorphism it is not yet known if it is NP-complete or eventually lies in P [4, 5].

A somehow disconnected problem that is also known as graph matching concerns matching of independent edges inside a single graph, which is usually bipartite. Independence between edges is defined by the absence of common incident nodes. This kind of graph matching is mainly used to decompose a graph into individual pairs, by searching a matching that covers the graph at maximum, for example, in order to establish factory-product associations. This problem is often, but not always, referred to as *bipartite graph matching* in order to distinguish it from the family of graph isomorphism problems.

The remainder of this paper is organized as follows. Section 2 details the inexact graph isomorphism problem, including the derived graph distance computation and a short overview of the variety of algorithms currently used. The following section describes the evolutionary algorithm architecture and introduces all existing and new operators. Section 4 provides a literature review of existing evolutionary approaches to the same problem, including an assessment of their results. We further detail, in Section 5, the evaluation framework, including systematic parameter tuning, performance measures and the results without additional local search. Section 5 comprises a comparison with additional local search strategies. First, since our crossover operator already incorporates some kind of local search, we compare distance-preserving crossover (DPX) to other classic operators with added local search. Second, the performance of DPX is studied when an additional local search strategy is added. We find that there is a further burst in terms of performance. The final section is the conclusion.

https://doi.org/10.25088/ComplexSystems.31.2.219

2. Inexact Graph Isomorphism

In real-world applications, usually the nodes and edges of a graph have attributes that need to be taken into account by the matching procedure. Thus, we face a labeled graph isomorphism problem, where not only the existence of edges must correspond but also their individual labels. However, mostly the attributes come from noisy environments, as, for example, in the case of graphs obtained from segmented images [6–8]. Then, the problem is transformed from a decision problem to the minimization problem of finding the matching with minimal cost. In our example illustrated in Figure 1, the inexact problem would appear when instead of color labels we had values (e.g., light-color frequencies), and we would allow close frequencies to match (in return of a estimated cost, for instance, the frequency distance). The advantage of allowing matching close attributes is that it permits the solution to be robust to noise. In fact, noise will change the values in the two representations (i.e., graphs), making exact matching impossible.

The matching cost is often obtained either by explicitly modeling each possible type of error (in particular when the attributes are nonnumerical labels), or by defining a dissimilarity measure (often a distance) between the attribute values. In this paper, we use the classical *joint distance* (see Section 2.1), which is a convex combination of the nodes' attribute distance and the relationships (edges' attributes) distance.

Formally, the inexact graph isomorphism problem, given two graphs of equal size $G = (\mathcal{V}, \alpha, \beta)$ and $G' = (\mathcal{V}', \alpha', \beta')$, consists in finding the mapping $m : \mathcal{V} \to \mathcal{V}'$ between the nodes of the two graphs that minimizes some cost function, formally called the dissimilarity metric.

The computational challenge arises from the fact that inexact graph matching implies a cost minimization on top of the graph matching problem, which in its turn implies a search of the large combinatorial space of mappings [4, 9]: If $\delta(G, G', m)$ is the matching cost of a mapping m between the two graphs G and G', the result of the *inexact graph matching* between the two graphs is obtained by minimizing over all possible mappings:

$$\delta(G, G') = \min_{m} \delta(G, G', m). \tag{2}$$

2.1 Joint Graph Distance

In order to define the distance between two graphs, we use the classical joint distance, which is nothing else than a weighted sum of the edges' attribute distances and the vertices' attribute distances. The

weighting parameter λ controls the importance of the edges versus the nodes. This straightforward and often-used distance, although not explicitly modeling the absence of nodes and edges, is able to handle them by the introduction of null labels.

In order to rigorously define the *joint graph distance*, we assume that appropriate elementary distance measures for nodes $\delta_{\mathcal{V}}$ and edges $\delta_{\mathcal{E}}$ are defined:

$$\delta_{\mathcal{V}} : \mathcal{A}_{\mathcal{V}} \times \mathcal{A}_{\mathcal{V}} \to \mathbb{R} \tag{3}$$

$$\delta_{\mathcal{E}} : \mathcal{A}_{\mathcal{E}} \times \mathcal{A}_{\mathcal{E}} \to \mathbb{R}. \tag{4}$$

For simplicity of notation, we will write in the following, for the distance of any two vertices v_1 and v_2 from two graphs $G_1 = (V_1, \alpha_1, \beta_1)$ and $G_2 = (V_2, \alpha_2, \beta_2)$, $\delta_{\mathcal{V}}(v_1, v_2)$ instead of $\delta_{\mathcal{V}}(\alpha_1(v_1), \alpha_2(v_2))$ and analogically for the edge distances.

With these notations, the joint graph distance of two graphs implied by the mapping m is defined by:

$$\delta(G, G', m) = \lambda \sum_{v \in \mathcal{V}} \delta_{\mathcal{V}}(v, m(v)) + (1 - \lambda) \sum_{v, w \in \mathcal{V}} \delta_{\mathcal{E}}((v, w), (m(v), m(w))). \tag{5}$$

The parameter $\lambda \in [0, 1]$ influences the relative importance of the nodes' attributes distance and the relationships (edges' attributes) distance, respectively, thus, it is highly application dependent. Since in this study we focus on a general description of our algorithm and evaluate it mostly on artificial data, we assume equal impact of both, that is, $\lambda = 0.5$. Using this parameter value, the absolute distance between two graphs is defined as the minimum over all legal mappings, as stated in equation (2).

3. Genetic Algorithm Design

Our genetic algorithm (GA) follows the classical steps: first, an initial population of s_{pop} solution candidates (chromosomes) is created. Then, a certain number of individuals are selected for producing offspring using genetic variation operators such as crossover or mutation. Some individuals from the child population eventually undergo a local search procedure. Finally, the offspring is integrated in or replaces the parent generation and the selection restarts. For simplicity's sake, in this paper, as replacement strategy we choose the generational approach, which means that we always create a child population of the same size as the parent population and replace the latter entirely at each generation.

https://doi.org/10.25088/ComplexSystems.31.2.219

3.1 Robust Permutation Encoding

Most genetic algorithms applied to the graph isomorphism problem use a permutations encoding under the form of integer strings. Indeed, any legal solution candidate has an equivalent representation in the space of permutations of $|\mathcal{V}'|$ elements. In fact, consider that the elements of $\mathcal{V}$ and $\mathcal{V}'$, respectively, are arbitrarily numbered from 1 to $|\mathcal{V}'|$. Then the corresponding mapping m will contain all pairs of nodes (v, v') that have the same number. In other words, any permutation of the numeration of V' implies a unique mapping and vice versa.

Figure 2 shows the permutational encoding of the graph match illustrated in Figure 1. We observe that the encoding of m (central rectangle) represents all the permutation pairs, for instance $(b, 4)$, by pointing node 4 of G_2 in position a. Only the identification of the destination nodes is needed since we can assume—because of the permutational nature of the exploration—that the order in the coding of m corresponds to a particular known order of G_1 (here the first position corresponds to a, the second to b, etc.) It is important to notice that this encoding is stable in the permutational space.

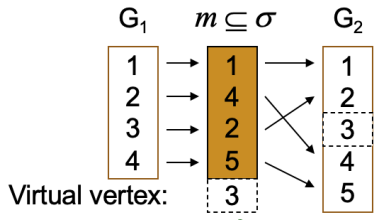

Figure 2. Permutation encoding corresponding to the subgraph match illustrated in Figure 1.

Unfortunately, the inexact subgraph isomorphism problem aims to minimize equation (5) for graphs of different sizes and thus, permutational coding is impossible. To overcome this, we propose to complete the smaller graph with unconnected virtual nodes. In Figure 2 this corresponds to the number 3 appearing at the end of the coding of m. This means that for this subgraph, match node 3 of G_2 was not selected.

Formally, if we assume $|\mathcal{V}| < |\mathcal{V}'|$, without loss of generality, the solution space can be transferred into the space of permutations of size $|\mathcal{V}'|$. Although this leads to a memory overhead of $O(|\mathcal{V}'| - |\mathcal{V}|)$ and to several permutations having identical mappings, it enables us to use all genetic operators available for permutation encodings [10].

3.2 Selection

The evolutionary process of survival of the fittest is modeled by various selection schemes; these determine which individuals from the current population migrate to the intermediate parent population, based on the distribution of fitness values. Various selection strategies are available for generational genetic algorithms (GAs). The most classic strategies include roulette wheel, ranking and tournament selection, among others [11]. In this paper, we choose to use tournament selection as a selection scheme, as it is very simple to implement and sufficiently efficient. In particular, it does not need any scaling or normalization of the fitness values and does not suffer from genetic drift [12].

3.3 Crossover

Crossover is the process of combining (in general) two parent chromosomes to create a common child. As a variation operator, it maintains diversity in the population and permits the sampling of new data points. A generally imposed (soft) constraint on a crossover operator, as well as any other variation operator, is that, given a set of valid solutions as inputs, it exclusively samples valid solutions with respect to the given encoding. We then say that the genotypic search space is closed under the operator. In certain special cases, this might be violated at the cost of introducing a repair mechanism or attributing fitness penalties to invalid solutions, mostly leading to reduced search efficiency. Since crossover operators play a crucial role in the global performance of any GA, they have been exhaustively studied. For many classical problems, like the traveling salesman problem (TSP), the introduction of new crossover operators has led to major enhancements of GA performance. In the following, we present groups of crossover operators particularly adapted to the permutational encoding.

3.3.1 Permutation Crossover Operators

There are three classes of application-independent crossover operators available for permutation encodings differing in the type of information they aim to preserve: position-based operators like CX [13] or POS [14], order-based operators like OX [15] or its uniform variant UOX [14], and operators preserving both gene order and absolute position to some degree, like PMX [16] or its uniform variant UPMX [17].

3.3.2 Position-Based Crossover Operators

Although most of the previous operators have been used in the context of graph matching, it appears that for our coding scheme, the

https://doi.org/10.25088/ComplexSystems.31.2.219

absolute position of the genes is crucial, while ordering information does not matter much. Therefore, position-based operators are the most appropriate. Two novel position-based operators, PBX and UPBX, were introduced in [10]. These operators outperformed PMX as well as UPMX on the graph matching task [18, 19].

3.3.3 Distance-Preserving Crossover

In this paper, we propose three new crossover operators: DPX, SDPX and NDPX, inspired by Freisleben and Merz's distance-preserving crossover (also called DPX, thus) for the TSP [20].

The basic idea of the DPX is to create a child whose distance to each parent equals the distance between the parents. As a result, the algorithm adapts automatically to the tradeoff between exploration and exploitation. In the beginning of a GA run (e.g., for a graph matching), parents are likely to be very different. The operator thus favors exploration over exploitation. While the population gets closer to an optimum, the interparent distances decrease and exploitation becomes more important.

The TSP-DPX works as follows: the content of the first parent is copied to the offspring and all edges that are not in common with the other parent are deleted. It results in short common tours. These broken tours are then reconnected without using the non-shared edges of the parents. A greedy reconnection procedure is employed to achieve this: if the edge (i, j) has been destroyed, the nearest available neighbor k of i among the remaining tour fragments is taken and the edge (i, k) is added to the tour, provided that (i, k) is not contained in the two parents.

For the TSP solution [20], common information refers to tour fragments, that is, (usually short) gene sequences on the genotypic level. For graph isomorphism, we propose to refer to elementary node mappings, that is, individual genes, instead. Thus, in order to obtain a complete chromosome, our operator iterates in random order over the open genes. At each position, it selects the best possible, nontaboo gene using an estimate of the influence of the particular gene on the overall fitness. We search the node pair leading to the minimal distance augmentation when added to the partial mapping already established. To this end, we compute, for each possible gene for a certain position, the sum of the edge distances to all other nodes already mapped and the node distance between the pair to be introduced.

This strategy offers a relatively precise estimation of the real cost, but is computationally expensive. Let n be the size of the (larger) graph and a the number of open positions (i.e., the Hamming distance between the parents, in the chromosome); the complexity of DPX is at most $O(1 / 2na^2)$.

Using a less strict estimate can greatly reduce the computational cost. We propose thus two relaxed versions of the DPX operator, node DPX (NDPX) and signature DPX (SDPX). NDPX takes into account only node distances and is therefore somehow inaccurate. As for SDPX, we compute an aggregation of a node's and its incident edges' attributes. The specific aggregation operator is strongly dependent on the application. The worst-case complexity of both operators equals the complexity of a fitness function evaluation, when the parents are completely different. In general, the complexity depends on the parents' distance and has an upper bound expressed by $O(1 / 2a^2)$.

Figure 3 compares the complexity of DPX, NDPX and SDPX to the complexity of a fitness function evaluation. All shown estimates are upper bounds for one execution of the operator and shown on a logarithmic scale as a function of the parents' distance. For SDPX, the signatures need to be computed in advance, adding the complexity of the aggregation operator once at the start of the algorithm.

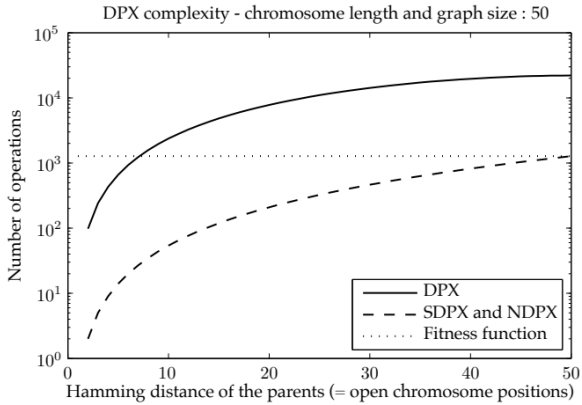

Figure 3. Complexity of the three DPX variants depending on the Hamming distance of the parents and compared to the computational cost of one fitness function evaluation. Complexities are relatively tight upper bounds ignoring the gain of taboo exclusion.

3.4 Mutation

Mutation is usually applied on single genes. In the case of permutation encoding, an appropriate elementary altering operation that preserves the permutation property is the swap of two genes. More disruptive mutation operators are possible, like scramble mutation, which scrambles a certain part of the chromosome's genes, but they are out of the scope of our paper.

https://doi.org/10.25088/ComplexSystems.31.2.219

4. State of the Art: Inexact Graph Isomorphism

During the past decades, numerous algorithms have been applied to the graph matching problem in general [21, 22]. In order to clarify the notation, it seems appropriate to distinguish the terms optimal, exact, inexact and approximate. We refer to exact and inexact when characterizing the nature of the graph matching problem; that is, does a match need to be exact or do we admit noise? The terms optimal and approximate refer to the algorithms used for any of the two problem types; that is, do we guarantee that the given distance (or match) is the best possible or do we admit fairly accurate solutions?

For the inexact graph isomorphism, on the one hand, the optimal algorithms (i.e., guaranteeing that the solution has the best possible minimal cost) are based on the A*-heuristic with some sophisticated lookahead procedures [3, 2, 23–25]. Unfortunately, none of them are computationally efficient, except for some specialized restrictions to certain types of graphs, for instance, planar. Therefore, these methods are limited to very small graph sizes (around 10 nodes).

On the other hand, approximate methods allow a reduced computational cost at the expense of dropping the optimality guarantee. They usually ensure a local maximum of the matching quality. Existing bio-inspired approaches include neural networks [26], ant colony optimization [27] and GAs, which we detail in the following.

4.1 Evolutionary Approaches

As in our proposition, most GAs applied to the graph isomorphism problem use permutations encoded as integer strings. The first successful attempts used PMX and CX and stated that the coding is similar to a TSP [28]. Although experimental evaluation is extremely limited, they were the first to mention that the quality of node assignments depends on value position rather than order. PMX and CX had to be extended for graph monomorphism, that is, subgraph matching.

Singh et al. [29] use both binary and integer strings. Furthermore, they introduce the concept of *color crossover*, which, using a given classification of the graph nodes, restricts the search space in allowing only mappings between nodes of the same class. The main problem is to find such a classification for any given application that the maximum inner class error is strictly inferior to the minimal distance between any two nodes from different classes. If a classification does not satisfy this condition, it could block the algorithm from obtaining the optimal result. In their experiments, Singh et al. [29] use randomly generated graphs containing 10, 30 and 50 nodes with mean (branching-)degrees between three and four for the smaller graphs and eight for size 50 graphs. After a total of 30 runs (10 different graphs with three runs each) for each size, there is no performance difference between the different crossover types; in particular, OX, CX

and PMX were tested. The use of color crossover with two colors significantly improves the results.

Wang et al. [18] tackle inexact graph isomorphism, that is, the noise-robust variant. They use PMX and swap mutation in combination with a local search step. The latter is based on two-opt applied once on each generation's best individual. Their experiments are also based on randomly generated graphs with integer attributes and noise added to these attributes.

All the above-mentioned works focused on pure graph isomorphism, where both graphs are of the same size. Based on the encoding explained in Section 3.1, it is easy to extend the operators to the more general subgraph problem. In fact, we avoid the necessity to extend the crossover operators by making a distinction between phenotype and genotype of the individuals. The genotype of an individual is a complete permutation (and therefore the operators apply), while the phenotype (e.g., used to compute the fitness) restricts itself to actually existing nodes in the smaller graph. Thus, in order to compare state-of-the-art approaches we select the following (most promising) permutation operators: CX, PMX, UPMX, UOX.

Paradoxically, as we observe in previous works [10], position-based operators, in particular PBX and UPBX, outperformed permutation-based PMX as well as UPMX on the graph matching task [18, 19]. Thus, we include these new two operators in the evaluation.

Finally, we also compare the three introduced (in Section 3.3) distance-preserving operators: DPX, SDPX and NDPX.

5. Evaluation Framework

For parameter tuning and general comparison between the operators, we use synthetic graphs. These graphs are fully connected, with edge and node attributes lying in the unit interval, and obtained using the Mersenne twister pseudorandom number generator [30]. We also use this generator to simulate uniform noise and the Ziggurat algorithm [31] for Gaussian noise.

Performance itself may be defined in a number of different ways, taking into account basically two components: the precision of the solution(s) obtained and the computational time that was used to obtain them. Some very common measures are success rate s_r, average runtime to success (ARS), average fitness function evaluations to success (AES) and success performance s_p.

- s_r. The success rate is obtained as the fraction of runs having reached the optimal solution. A high value indicates a good ability of the algorithm to converge to the right solution.

https://doi.org/10.25088/ComplexSystems.31.2.219

- ARS. The average runtime to success corresponds to the mean value of the time needed to obtain the optimal solution. This value allows comparison of all different kinds of algorithms without the need of assessing their specific complexity. However, it is dependent on hardware and implementation, which limits the generality of the results and comparison with other authors. A low value means that when a solution is found, it is found quickly.
- AES. The average number of fitness function evaluations to success corresponds to the mean convergence time as a multiple of the time needed to compute the fitness function. It is similar to the ARS, but results are comparable on different hardware. However, when the compared GAs differ greatly, especially when some operators are far more complex than others, counting the number of fitness function evaluations might be misleading as other parts of the algorithm, like local search or complex operators, might have a larger influence on time complexity. In these cases, a pure runtime comparison might be more appropriate. As with ARS, it purely measures the performance for the runs that find the optimal solution and contains no information about how often it was actually found.
- s_p. The success performance is obtained by dividing ARS by the success rate (ARS / s_r). This normalization, in contrast to ARS, penalizes unsuccessful runs. It provides a single number to measure both the ability to converge to the optimal solution and the speed of convergence in a single number. Notice that some authors also call success performance the equivalent regularization computed on AES.

Since the proposed distance-preserving operators (DPX, SDPX and NDPX) include a fitness estimation, which implies an extra computational cost (as discussed in Section 3.3), using a measure that only counts the number of fitness evaluations will bias the results by favoring them in any comparison. Therefore in our experiments and comparisons, we focus rather on computational time, paying particular attention to having the same computational environment (software and hardware).

5.1 Standard Parameters

Parameter setting is crucial to the performance of evolutionary algorithms. The optimization of the parameter set is itself a combinatorial optimization problem. Furthermore, optimal parameter settings may change with time or not even exist.

In order to identify reasonable values for the three basic parameters: population size, crossover and mutation probability, for each crossover operator, in a reasonable time, we reduce the search space by discretizing all three parameters. We use a nonuniform discretization that takes into account known facts about usually good ranges. Inside promising ranges, the sampling gets finer, while nonpromising

areas are sparsely covered. The following known assumptions were used:

- p_c. The crossover rate should be relatively high, for example, between 0.6 and 1.0 [32, 33]. The sampling gets denser for higher values approaching 0.01 intervals between 0.95 and 0.99.
- p_m. The mutation rate is usually small, for example, around 0.01 to 0.1 [32] or using a more classic assumption $1 / L$ with L being the chromosome length [33]. We sample relatively densely up to 0.15, and afterward sparsely until a maximum of 0.7. Higher values would result in an unwanted random walk behavior.
- s_{pop}. The population size is usually in the order of 20 to 100. We cover the interval from 10 to 1000 and sample more densely for low values.

5.2 Best Parameters

We search the space, starting with the known assumption for the parameter settings. We use 100 individuals, a crossover probability of 0.99 and a mutation probability of 0.1. Then we use a loose gradient descent technique, treating the three parameters sequentially, until we find a stable local optimum. During each step, we adjust the parameter where the highest gain was observed in terms of success performance s_p, taking into consideration both convergence itself and convergence speed [34]. In particular, we use the success rate s_p and AES, eventually completed by ARS for crossover operators with nonconstant computational cost. Each parameter setting is evaluated on 50 graph pairs and 30 runs are done on each of them to obtain statistically significant results.

Finally we explore the area around this optimum using denser sampling, an idea somehow similar to sharpening [35], but different in the fact that we do not aim at augmenting the accuracy of the performance estimates for a set of given parameter settings but at evaluating more parameter settings in the area of a promising one.

The results of this process are detailed in [36] and the optimal parameters are displayed in Table 1. We observe that there is a high variance of the optimal values, depending on the operator. For instance, PBX has an optimal parameter very close to the starting point (population size 100, crossover rate 0.99, mutation rate 0.1), while DPX requires an unexpectedly low crossover rate. Another example of this variability is the extremely high population size needed when using cycle crossover (CX).

In tests, we also observed that the parameter settings are generally stable with respect to graph size (evaluated between 20 and 40) and noise level (from 0 to a standard deviation of 10% of the attribute scale). The only exception being the optimal population size of CX

https://doi.org/10.25088/ComplexSystems.31.2.219

depending on the graph size; that is, bigger graphs need higher population sizes.

Finally, we observed that in general, for growing graph sizes, the sensitivity on the parameters increases; that is, if there is an interval for which the performance is on a plateau, its width decreased.

Crossover Operator	s_{pop} (ind.)	p_c (%)	p_m (%)	Success Rate	Success Performance (sec.)
DPX	50	25	25	1	0.087
SDPX	25	35	30	1	0.165
NDPX	25	40	30	1	0.219
UPBX	90	99	20	0.994	0.298
PBX	120	96	30	0.996	0.388
PMX	120	99	40	0.997	0.430
CX	400	99	60	0.995	0.473
UPMX	90	80	12	0.979	0.669
UOX	10	60	14	0.997	2.372

Table 1. Operators ranked by success performance with their optimal parameter set (population size s_{pop}, crossover probability p_c and mutation probability p_m) for graphs of size 40 with uniform noise (6). Distance-preserving crossovers perform best, followed by strictly position-based crossovers.

5.3 Performance Comparison

Looking at Table 1 from the success performance perspective, we notice three groups of basic crossover operators. Strict position-based crossover (PBX and UPBX, [10]), classical position-based and hybrid operators (CX, PMX, UPMX), and order-based crossover (UOX). The last one can be discarded as it is from its foundation unfit for the graph matching problem under permutation encoding and obtains very poor results, with an extremely high sensitivity to parameter settings.

Success rates are here close to one, translating the fact that almost all the runs reached the optimal solution. This is an expected result, since the graphs' size was chosen in such a way that parameter optimization can be performed (i.e., with a very large number of runs, as explained earlier). The high scores also translate the fact that for mid-sized graphs, any approach—as long as it has optimal parameters—reaches the optimal solution most of the time, the only difference being the time to reach it.

A closer look at our results reveals that UPBX and PBX show similar success performances, while PBX seems more sensitive to its parameters, especially to the mutation probability. PMX and CX need extremely high mutation rates and at the same time high population sizes, indicating a merely random exploration of the current neighborhood guided by the evolutionary selection process. This indicates that the crossover operator itself does not play a major role. We could cross-check with a mutation-only GA. UPMX's parameters seem somehow normal and it shows slightly better performances than PMX and CX. It is, however, extremely dependent on a sound crossover probability of 80% and the tuning of the population size.

CX's poor results contradict the a priori assumption that a purely position-based operator should perform best on our problem. This may be explained by the nondisruptive nature of the operator in combination with a weak mutation. We did not observe a significant performance gain after combining CX with scramble mutation, which leads to more disruptive behavior.

5.3.1 Convergence Comparison

Figure 4 shows the convergence behavior of all operators, using their respective optimal parameters at size 40. The three DPX variants converge extremely fast. Although the accuracy of all operators is close to 1 (see also Table 1) and, in general, the precision decreases with the graph size (for most operators, this decrease starts at size 40), the convergence behavior presented in Figure 4 shows significant differences between the operators.

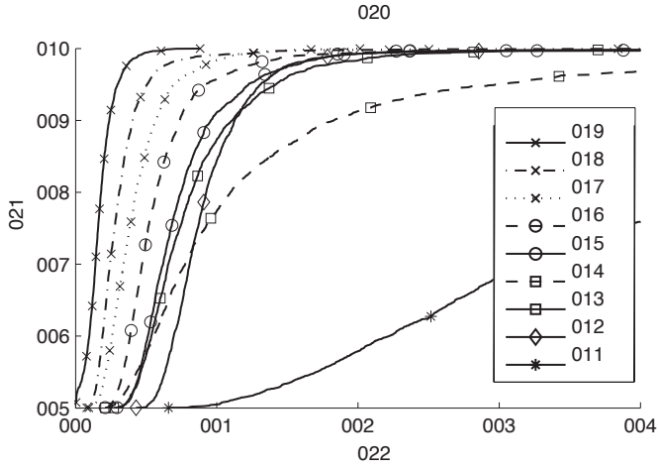

Figure 4. Success rate over time for all crossover operators using optimal parameter settings. DPX clearly outperforms all other operators.

https://doi.org/10.25088/ComplexSystems.31.2.219

DPX converges twice as fast as SDPX. This indicates that the precision of the distance (i.e., fitness) estimation in the greedy part of the operator is far more important than its runtime.

5.3.2 Distance-Preserving Crossover Sensitivity to Parameter Setting

The parameter tuning reveals some interesting facts about the DPX operators. There is a weak correlation between crossover probability and the optimal corresponding population size. The performance of the DPX operators is extremely sensitive to inappropriate crossover probabilities, which should be lower than 50% (see Figure 5). This contradicts the common assumption of a need for relatively high crossover rates.

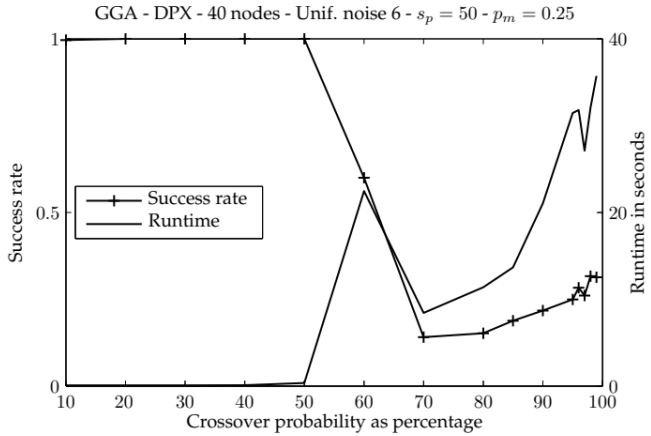

Figure 5. Influence of the crossover probability on the performance of the DPX operator. The success rate decreases and the average runtime increases significantly when the crossover probability exceeds a threshold around 0.5.

All this can be explained by the fact that the solution vectors created by DPX are as far away from each of the parents as the parents are separated, which is a particularly large distance during the first generations. Therefore, if the crossover rate approaches higher values, most parents would be immediately replaced by their—faraway—children. Although exploration benefits from this behavior, exploitation of promising regions is hindered and thus the selection process is blocked. This has a strong impact on the convergence. In fact, we observe that for any distance-preserving operator there is a crossover probability bound that, when exceeded, immediately leads to poor convergence and precision, while any values below that bound only affect runtime without affecting precision at all.

The optimal mutation rates are also higher than expected (from the literature) with values around 25% to 30%. This relatively high

mutation rate may be explained by two effects: On the one hand, the greedy procedure used to fill in the chromosomes introduces a strong bias and potentially reduces diversity. A higher mutation rate contributes to better diversity. On the other hand, swap mutation is not a very disruptive operator, since it just exchanges two out of n genes. A higher mutation rate makes it more disruptive and increases the possible radius of the mutation, leading to better exploitation and hence convergence.

6. Local Search

Adding a local search step to a GA may significantly enhance the results, as the algorithm exploits the exploration capabilities of the GA and at the same time uses the exploitation properties of the local search for faster convergence [37]. A good memetic algorithm needs an appropriate balance between genetic exploration and local search [38]. Too much local search leads to premature convergence, while too much exploration slows the algorithm down. The introduction of a local search step permits scaling the algorithm to larger graphs. Additionally, the greedy procedure of the DPX operator may be considered an adaptive local search whose search radius equals the parent distance. Therefore, we propose to add a local search procedure to all other operators and compare them to DPX.

Local search strategies define a neighborhood around a given chromosome and search it exhaustively. The chromosome is then either replaced by the best chromosome of its neighborhood, which is called Lamarckian evolution, or, according to the Baldwin effect, only its fitness is updated but the genes stay unchanged [39]. The search radius can be defined by the elementary operation $b : s \to b(s)$, which is used to create the neighborhood around the chromosome s.

We apply a two-opt local search procedure; that is, the operation b consists in exchanges of any two genes, with replacement of the chromosomes. The neighborhood contains $C_n^2 = n(n-1)/2$ chromosomes. The complexity of the fitness calculus for these chromosomes is significantly reduced by ignoring all unchanged mappings. The exchange of two genes is equivalent to exchanging the elementary mappings of two graph nodes. Since there are two graph node distances and $2(n-2)$ (there are no cycles and the edge between the two nodes is unaffected since undirected) edge distance changes, the overall complexity of a local search step is

$$O\left(C_n^2 \times \left(2 + 2(n-2) + \frac{n^2+n}{2}\right)\right),$$

https://doi.org/10.25088/ComplexSystems.31.2.219

which is $O(2n)$ more expensive than a single call to the fitness function but covers a space of C_n^2 individuals. Taking into account its complexity, it is obvious that local search should only be applied to a small proportion of individuals. In the following we identify the most promising selection strategy for this purpose.

6.1 Smart Local Search Strategies

In its most basic form, the memetic algorithm creates an individual by the genetic operators and decides whether or not that individual undergoes a local search procedure. It is often useful to concentrate the computational effort on the most promising solutions instead of applying local search to all individuals with constant probability. To this end, we sort the population and apply local search to the first $p_{ls} \times n$ individuals.

Additionally, since the local search procedure is deterministic, the results are the same if applied to the copy of an individual. When a copy of an individual that underwent local search is detected, we have two possibilities. First, we may copy the result of the local search directly without executing it a second time. Second, we can simply ignore the copy and not apply local search on it. In this case, we leave it to the selection process to determine the better individual from these two. We opted for the second solution as it seems a more natural way, and as preliminary tests between the two methods did not show much of a difference.

In the end, we test four variants in order to integrate local search into the generational algorithm, using the letters U for unique, where copies are discarded, and S for sorted. For example, USGGA stands for a sorted population where copies are discarded.

6.2 Comparing Local Search Enhanced Operators and Distance-Preserving Crossover

We determined a pseudo-optimal parameter setting for the tradeoff between genetic exploration and local search by exhaustively testing local search probabilities between 0 and 30% for each crossover operator and algorithm type (Figure 6). Tables 2 and 3 show the success performance of the best parameter setting for every combination at graph sizes of 60 and 100 nodes, respectively. The performances of the generational GA improve for almost all operators, particularly when a sorted implementation is used. The difference between a simple sorted implementation (SGGA) and the one excluding duplicates is generally insignificant. The optimal parameter settings are also similar.

Comparing the two sizes, local search is shown to be more beneficial for bigger graphs, as one would generally expect.

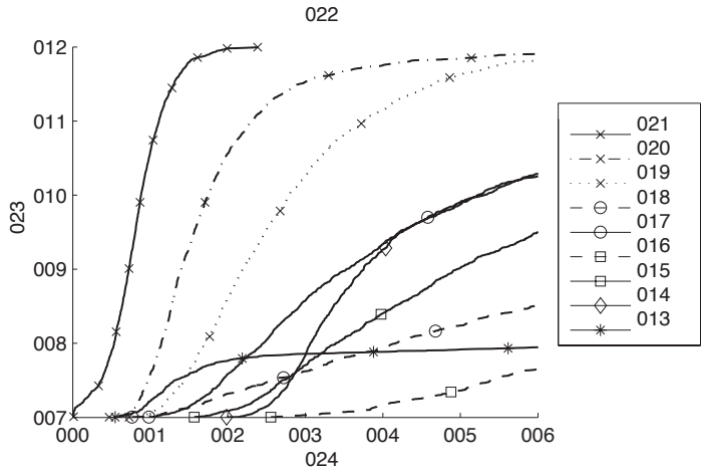

Figure 6. Comparison of crossover operators with local search against distance-preserving operators without local search.

	GA	GGA		UGGA		SGGA		USGGA	
Operator	s_p	s_p	(p_{LS})	s_p	(p_{LS})	s_p	(p_{LS})	s_p	(p_{LS})
DPX	0.41	0.23	(0.01)	0.27	(0.03)	0.12	(0.03)	0.12	(0.03)
SDPX	0.95	0.74	(0.01)	0.86	(0.02)	0.47	(0.07)	0.46	(0.07)
NDPX	1.42	1.12	(0.01)	1.31	(0.02)	1.01	(0.07)	0.99	(0.10)
UPBX	8.20	n.s.f		6.54	(0.01)	6.88	(0.01)	5.83	(0.01)
PBX	2.97	n.s.f		n.s.f		n.s.f		n.s.f	
UPMX	394.26	n.s.f		n.s.f		8.99	(0.08)	8.82	(0.06)
PMX	4.23	n.s.f		n.s.f		3.88	(0.01)	3.75	(0.01)
CX	4.04	n.s.f		n.s.f		3.23	(0.01)	3.02	(0.01)
UOX	122.15	10.21	(0.01)	11.44	(0.18)	7.23	(0.18)	8.68	(0.18)

Table 2. Comparison of success performances of all operators combined with local search for large graphs of size 60. The corresponding best local search probability parameter is given in brackets. Where not reported, no solution was found (n.s.f) in any run. We notice that local search works best when applied on the best individuals of the population. DPX outperforms all other operators independently on the use of local search. DPX can be further improved by adding local search.

https://doi.org/10.25088/ComplexSystems.31.2.219

Operator	GA s_p	GGA s_p	GGA (p_{LS})	UGGA s_p	UGGA (p_{LS})	SGGA s_p	SGGA (p_{LS})	USGGA s_p	USGGA (p_{LS})
DPX	3.87	1.57	(0.01)	2.14	(0.03)	0.83	(0.02)	0.84	(0.03)
SDPX	11.33	6.79	(0.01)	10.11	(0.07)	2.97	(0.07)	3.10	(0.07)
NDPX	39.82	12.62	(0.01)	25.80	(0.05)	8.94	(0.07)	7.72	(0.07)
UPBX	n.s.f	n.s.f		n.s.f		n.s.f		n.s.f	
PBX	76.35	n.s.f		n.s.f		n.s.f		n.s.f	
UPMX	n.s.f	n.s.f		n.s.f		1979.3	0.08	4148.8	(0.09)
PMX	6136.2	n.s.f		n.s.f		222.87	0.02	212.34	(0.02)
CX	n.s.f	n.s.f		n.s.f		146.89	0.01	164.88	(0.01)
UOX	n.s.f	97.41	(0.01)	313.99	(0.16)	93.31	(0.18)	100.39	(0.12)

Table 3. Comparison of success performance of all operators combined with local search for graphs of size 100. The corresponding best local search probability parameter is given in brackets. Where not reported, no solution was found (n.s.f) in any run. Only DPX provides acceptable performance. Most other operators have success rates equal to or near null even with local search.

6.3 Distance-Preserving Crossover with Local Search

As local search enhances the performance of virtually any operator, it is straightforward to examine the possible gain of local search with DPX. Comparing the two graph size sets, local search is shown to be more beneficial for bigger graphs, as would generally be expected. For example, the success performance measure for SDPX is improved by 51% at size 60 (see Table 2), which corresponds to a reduction of the mean time to convergence to half. The improvement attains nearly 74% for larger graphs. The mean convergence time is hence divided by four.

In conclusion, local search pushes forward the size limitations of any operator with the distance-preserving operators being the only ones still offering a near 100% accuracy at graph size 100. For these last, local search accelerates convergence.

6.3.1 Noise Sensitivity

Another important aspect in the case of inexact subgraph isomorphism is the robustness to intrinsic noise. Noise needs to be analyzed in conjunction with graph size, since higher noise levels for small or medium graph sizes do represent a less difficult problem than for large graphs. A noise range of $[-10\% + 10\%]$ of the initial value range of the attributes (for Gaussian noise, we use a σ leading to an identical variance expectations) can safely be considered. (The difference

between the results using uniform and Gaussian noise stems from the fact that the equivalence formula used to determine corresponding σ for each uniform noise interval guarantees identical variance, which is quadratic, while distances are linear, which makes Gaussian instances slightly less complex for higher noise levels.) In fact, we did not intend to approach levels that compromise the optimal solution, that is, where the numerical optimal solution does not correspond to the ground truth due to noise effects. The success rate of our DPX operator (without local search) at graph size 80 ranges from 1 to 0.95 for Gaussian noise and from 1 to 0.88 for uniform noise.

Figures 7 and 8 show the effects of increasing noise levels on convergence and precision of the algorithm for larger graphs. In this case, it is also appropriate to introduce an additional local search step even to DPX. As expected, the success rate decreases with higher noise levels, but stays on acceptable levels. Note that the introduction of local search has no effect if the initial precision was one. In general, as depicted in Figure 8, local search significantly accelerates convergence. A minor side effect of this acceleration is the slight loss in precision observed for cases where the initial precision is below one. Among the different distance-preserving operators, SDPX is more sensitive to higher noise than DPX. This is due to the information loss for the signature with respect to the original graph, which may eventually compromise the optimal solution, as described.

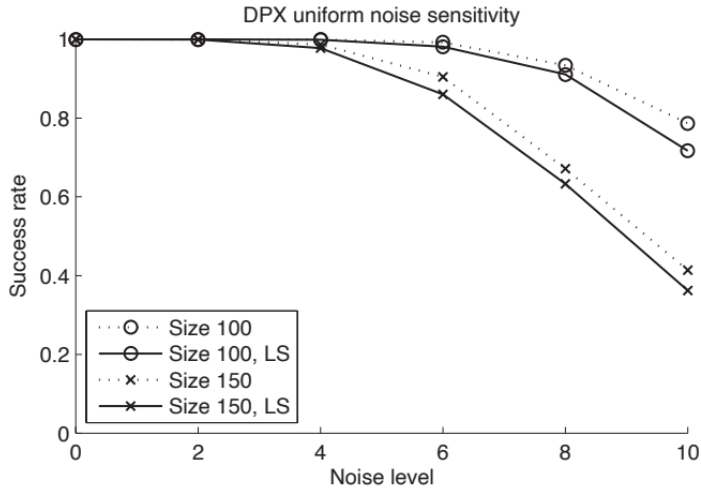

Figure 7. Success rate sensitivity to noise with and without local search (sorted GA 2%).

https://doi.org/10.25088/ComplexSystems.31.2.219

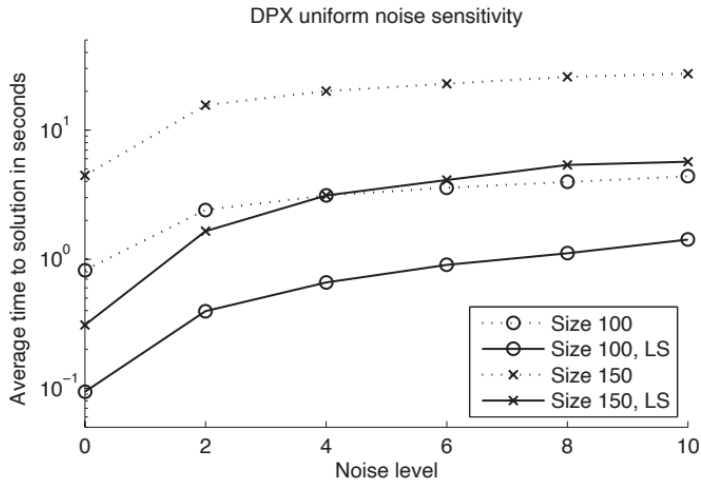

Figure 8. Noise sensitivity with and without 2% local search (sorted GA).

6.3.2 Approaching the Limit

After bringing up the relationship between noise and graph size in the previous section, we would like to explore the limits of our method in terms of graph size. Table 4 shows the mean runtime of successful

Algorithm Type	Size 100	Size 200	Size 300
DPX			
GGA w/o LS	3.7	120.1	5009.2
SGGA w 2% LS	**0.8**	23.1	296.1
SGGA w 3% LS	**0.8**	22.2	350.1
USGGA w 3% LS	**0.8**	27.7	274.8
SDPX			
GGA w/o LS	11.4	4563.3	n.s.f
SGGA w 7% LS	3.2	122.4	2989.0
USGGA w 7% LS	3.3	121.0	2868.2
NDPX			
GGA w/o LS	38.7	n.s.f	n.s.f
SGGA w 7% LS	7.9	687.3	n.s.f
USGGA w 7% LS	8.2	761.9	n.s.f
USGGA w 10% LS	9.3	928.5	n.s.f

Table 4. Mean runtime of successful runs (in seconds) of DPX using different local search implementations. Where not reported, no solution was found (n.s.f) in any run.

runs of the three DPX operators for graphs with 100, 200 and 300 nodes. Without local search, the runtime of DPX at size 100 is more than four times higher than with local search. The advantage is similar (or even greater) for all three operators. Local search is more important with increasing graph sizes.

The success rate obviously decreases with higher graph sizes as with higher noise levels (see Table 4). Without local search, graph sizes of 200 are definitely too large for NDPX and SDPX, and 300 represents a limit for DPX.

As shown in Table 5, the effect of local search changes in cases of extremely low (initial) precision. In that case, local search also leads to a significantly increased success rate and allows reaching still acceptable performance for graphs that are otherwise too large. Therefore, we consider that local search effectively enhances performance when the problem gets more complicated.

Algorithm Type	Size 100	Size 200	Size 300
DPX			
GGA w/o LS	0.99	0.66	0.04
SGGA w 2% LS	0.98	0.62	0.25
SGGA w 3% LS	0.99	0.64	0.25
USGGA w 3% LS	0.98	0.63	0.27
SDPX			
GGA w/o LS	0.96	0.00	n.s.f
SGGA w 7% LS	0.74	0.39	0.07
USGGA w 7% LS	0.73	0.40	0.08
NDPX			
GGA w/o LS	0.49	n.s.f	n.s.f
SGGA w 7% LS	0.61	0.11	n.s.f
USGGA w 7% LS	0.58	0.10	n.s.f
USGGA w 10% LS	0.54	0.10	n.s.f

Table 5. Success rates of DPX using different local search implementations. The best success rates are always obtained by the GA applying local search to the best individuals. Where not reported, no solution was found (n.s.f) in any run.

7. Conclusion

In this paper, we presented a family of distance-preserving crossover (DPX) operators for the inexact graph matching problem. The use of these operators presents a big step forward compared to existing

evolutionary approaches. In particular, the practical size limit for acceptable performance increased significantly. This enables the efficient use of more detailed graph models for real-world applications in the future. Such applications are today primarily in cheminformatics in the identification of chemical compounds, but also in electronic design automation in the verification of the equivalence of various representations of the design of an electronic circuit. In the future, with the increase in the capacity of the algorithms to deal with larger graphs, the number of applications will explode. Any comparison of information described by a graphical model, such as medical, multimedia or social network–based content, will require solving a subgraph isomorphism problem.

For large graphs, our algorithm outperforms all state-of-the-art approximate algorithms, which are to the best of our knowledge currently limited to no more than 50 nodes. We validated the performance of our approach for graphs containing up to 300 nodes with different levels of perturbation.

Our DPX operator outperformed all of the large set of alternative crossover operators studied. In particular, at graph size 100, DPX was the only operator leading to a significant precision of 100% against 0% for most of the others. Moreover, it was still able to find solutions at graph size 300, which is about an order of magnitude larger than the maximum graph size for which the inexact graph matching problem could be solved so far.

In order to reach this sphere, we use a two-level local search heuristic. First, local search is integrated into the crossover operator; second, another local search heuristic is applied, with low probability, independently. All interoperator comparisons were carried out after individually optimizing all parameters, since not only is operator performance substantially influenced by its parameters, but sensitivity to parameter changes is also substantial for most of them.

All tests on larger graphs were performed on artificial data due to the lack of suitable real-world graph models. Our approach opens up the possibility of manipulation and conceiving larger graphs. But the problem is not simple; for instance, in a two-dimensional to three-dimensional face recognition task, we observed that augmenting the level of detail of the graph model led to lower final precision in recognition, although the numerical optimum was found. It would hence be interesting to see to what extent more detailed graph models can be developed for real-world inexact graph matching problems, without becoming too brittle to work with.

References

[1] R. A. Wagner and M. J. Fischer, "The String-to-String Correction Problem," *Journal of the ACM*, **21**(1), 1974 pp. 168–173. doi:10.1145/321796.321811.

[2] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *Journal of the ACM*, **23**(1), 1976 pp. 31–42. doi:10.1145/321921.321925.

[3] S. Berretti, A. Del Bimbo and E. Vicario, "Efficient Matching and Indexing of Graph Models in Content-Based Retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**(10), 2001 pp. 1089–1105. doi:10.1109/34.954600.

[4] J. Köbler, U. Schöning and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Boston: Birkhäuser, 1993.

[5] J. Torán, "On the Hardness of Graph Isomorphism," *SIAM Journal on Computing*, **33**(5), 2004 pp. 1093–1108. doi:10.1137/S009753970241096X.

[6] Z. Harchaoui and F. Bach, "Image Classification with Segmentation Graph Kernels," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, Minneapolis, MN, Los Alamitos, CA: IEEE Computer Society, 2007 pp. 1–8. doi:10.1109/CVPR.2007.383049.

[7] M. C. Boeres, C. C. Ribeiro and I. Bloch, "A Randomized Heuristic for Scene Recognition by Graph Matching," in *Experimental and Efficient Algorithms (WEA 2004)* (C. C. Ribeiro and S. L. Martins, eds.), Berlin, Heidelberg: Springer, 2004 pp. 100–113. doi:10.1007/978-3-540-24838-5_8.

[8] N. Thome, D. Merad and S. Miguet, "Learning Articulated Appearance Models for Tracking Humans: A Spectral Graph Matching Approach," *Signal Processing: Image Communication*, **23**(10), 2008 pp. 769–787. doi:10.1016/j.image.2008.09.003.

[9] W. H. Tsai and K.-S. Fu, "Error-Correcting Isomorphism of Attributed Relational Graphs for Pattern Analysis," *IEEE Transactions on Systems, Man, and Cybernetics*, **9**(12), 1979 pp. 757–768. doi:10.1109/TSMC.1979.4310127.

[10] T. Bärecke and M. Detyniecki, "Memetic Algorithms for Inexact Graph Matching," *2007 IEEE Congress on Evolutionary Computation*, Singapore, Piscataway, NJ: IEEE, 2007 pp. 4238–4245. doi:10.1109/CEC.2007.4425024.

[11] T. Blickle and L. Thiele, "A Comparison of Selection Schemes Used in Evolutionary Algorithms," *Evolutionary Computation*, **4**(4), 1996 pp. 361–394. doi:10.1162/evco.1996.4.4.361.

[12] A. Rogers and A. Prugel-Bennett, "Genetic Drift in Genetic Algorithm Selection Schemes," *IEEE Transactions on Evolutionary Computation*, **3**(4), 1999 pp. 298–303. doi:10.1109/4235.797972.

[13] I. M. Oliver, D. J. Smith and J. R. C. Holland, "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*, Cambridge, MA (J. J. Grefenstette, ed.), Hillsdale, NJ: L. Erlbaum Associates Inc., 1987 pp. 224–230.

[14] G. Syswerda, "Schedule Optimization Using Genetic Algorithms," *Handbook of Genetic Algorithms* (L. Davis, ed.), New York: Van Nostrand Reinhold, 1991 pp. 332–349.

[15] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*, Los Angeles (A. Joshi, ed.) San Francisco, CA: Morgan Kaufmann Publishers Inc., 1985 pp. 162–164.

[16] D. E. Goldberg and R. Lingle, "Alleles, Loci and the Traveling Salesman Problem," in *Proceedings of the 1st International Conference on Genetic Algorithms* (J. J. Grefenstette, ed.), Hillsdale, NJ: L. Erlbaum Associates Inc., 1985 pp. 154–159.

[17] V. A. Cicirello and S. F. Smith, "Modeling GA Performance for Control Parameter Optimization," in *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, Las Vegas (L. D. Whitley and D. E. Goldberg, eds.), San Francisco: Morgan Kaufmann Publishers Inc., 2000 pp. 235–242. dl.acm.org/doi/10.5555/2933718.2933750.

[18] Y.-K. Wang, K.-C. Fan and J.-T. Horng, "Genetic-Based Search for Error-Correcting Graph Isomorphism," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **27**(4), 1997 pp. 588–597. doi:10.1109/3477.604100.

[19] R. Torres-Velázquez and V. Estivill-Castro, "A Memetic Algorithm Guided by *Quicksort* for the Error-Correcting Graph Isomorphism Problem," in *Applications of Evolutionary Computing (EvoWorkshops 2002)*, Kinsale, Ireland (S. Cagnoni, J. Gottleib, E. Hart, M. Middendorf and G. R. Raidl, eds.), 2002 pp. 173–182. doi:10.1007/3-540-46004-7_18.

[20] B. Freisleben and P. Merz, "New Genetic Local Search Operators for the Traveling Salesman Problem," in *Parallel Problem Solving from Nature—PPSN IV*, (H. M. Voigt W. Ebeling , I. Rechenberg and H. P. Schwefel, eds.), Berlin, Heidelberg: Springer, 1996 pp. 890–899. doi:10.1007/3-540-61723-X_1052.

[21] R. C. Read and D. G. Corneil, "The Graph Isomorphism Disease," *Journal of Graph Theory*, **1**(4), 1977 pp. 339–363. doi:10.1002/jgt.3190010410.

[22] D. Conte, P. Foggia, C. Sansone and M. Vento, "Thirty Years of Graph Matching in Pattern Recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, **18**(3), 2004 pp. 265–298. doi:10.1142/S0218001404003228.

[23] H. Bunke and G. Allerman, "A Metric on Graphs for Structural Pattern Recognition," in *Signal Processing II: Theories and Applications: Proceedings of EUSIPCO-83, Second European Signal Processing Conference*, Erlangen, W.-Germany (H. W. Schüssler, ed.), New York: North-Holland, 1983.

[24] L. G. Shapiro and R. M. Haralick, "Structural Descriptions and Inexact Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **3**(5), 1981 pp. 504–519. doi:10.1109/TPAMI.1981.4767144.

[25] W.-H. Tsai and K.-S. Fu, "Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Analysis," *IEEE Transactions on Systems, Man, and Cybernetics*, **9**(12), 1979 pp. 757–768. doi:10.1109/TSMC.1979.4310127.

[26] B. J. Jain and F. Wysotzki, "Solving Inexact Graph Isomorphism Problems Using Neural Networks," *Neurocomputing*, **63**, 2005 pp. 45–67. doi:10.1016/j.neucom.2004.01.189.

[27] O. Sammoud, S. Sorlin, C. Solnon and K. Ghédira, "A Comparative Study of Ant Colony Optimization and Reactive Search for Graph Matching Problems," in *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, Budapest, Hungary (J. Gottlieb and G. R. Raidl, eds.), Berlin, Heidelberg: Springer, 2006 pp. 234–246. doi:10.1007/11730095_20.

[28] M. Krcmar and A. P. Dhawan, "Application of Genetic Algorithms in Graph Matching," in *Proceedings of the 1994 IEEE International Conference on Neural Networks. Part 1 (of 7)*, Orlando, Piscataway, NJ: IEEE, 1994 pp. 3872–3876.

[29] M. Singh, A. Chatterjee and S. Chaudhury, "Matching Structural Shape Descriptions Using Genetic Algorithms," *Pattern Recognition*, **30**(9), 1997 pp. 1451–1462. doi:10.1016/S0031-3203(96)00181-1.

[30] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, **8**(1), 1998 pp. 3–30. doi:10.1145/272991.272995.

[31] G. Marsaglia and W. W. Tsang, "The Ziggurat Method for Generating Random Variables," *Journal of Statistical Software*, **5**(8), 2000 pp. 1–7. doi:10.18637/jss.v005.i08.

[32] V. Nannen and A. E. Eiben, "Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters," in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07)*, Hyderabad, India, California: International Joint Conferences on Artificial Intelligence, 2007 pp. 975–980. www.ijcai.org/Proceedings/07/Papers/157.pdf.

[33] H. Mühlenbein, "How Genetic Algorithms Really Work: Mutation and Hillclimbing," in *Parallel Problem Solving from Nature 2 (PPSN-II)*, Brussels, Belgium (R. Manner and B. Maderick, eds.), New York: Elsevier Science Inc., 1992 pp. 15–25.

[34] P. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," Technical Report, Nanyan Technological University & Kanpur Genetic Algorithms Laboratory, May 2005.

[35] S. K. Smit and A. E. Eiben, "Comparing Parameter Tuning Methods for Evolutionary Algorithms," in *2009 IEEE Congress on Evolutionary Computation*, Trondheim, Norway, Piscataway, NJ: 2009 pp. 399–406. doi:10.1109/CEC.2009.4982974.

[36] T. Bärecke, "Evolutionary Optimisation for Inexact Graph Isomorphism," Ph.D. thesis, Université Pierre et Marie Curie - Paris VI, 2009.

[37] J.-M. Renders and S. P. Flasse, "Hybrid Methods Using Genetic Algorithms for Global Optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **26**(2), 1996 pp. 243–258. doi:10.1109/3477.485836.

[38] H. Ishibuchi, T. Yoshida and T. Murata, "Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling," *IEEE Transactions on Evolutionary Computation*, **7**(2), 2003 pp. 204–223. doi:10.1109/TEVC.2003.810752.

[39] D. L. Whitley, V. S. Gordon and K. E. Mathias, "Lamarckian Evolution, the Baldwin Effect and Function Optimization," in *Parallel Problem Solving from Nature - PPSN III*, Jerusalem, Israel (Y. Davidor, H.-P. Schwefel and R. Männer, eds.), Berlin, Heidelberg: Springer, 1994 pp. 6–15. doi:10.1007/3-540-58484-6_245.