University of Melbourne

MAST90098 Group Project

Group 6:
Harry Xi
Naryman Tarvand
Owen Siu

August, 2023

Contents

1 Introduction & Motivation

The Quadratic Assignment Problem (QAP), is a well-known combinatorial optimisation problem in operations research, management sciences and economics, relating to the optimal planning of facility locations. Broadly, it asks for the assignment of a number of facilities to an equal number of locations, minimising the sum of products involving a flow cost between facilities and a distance cost between locations.

Indeed, it provides a flexible enough framework to be used in a variety of use cases, despite first being conceptualised in the context of economic planning [17]. Further use cases include: hospital layout design [10] and exam scheduling [9].

This report will detail a number of heuristic solution methods, together with extensive experimentation as well as meta-analysis of the instance space.

1.1 Problem Formulation

The modern Quadratic Assignment Problem can be formulated as an integer program of the form [5]:

$$\begin{aligned}
\min \; & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{p=1}^{n} f_{ij} d_{kp} x_{ik} x_{jp} \\
\text{s.t.} \; & \sum_{i=1}^{n} x_{ij} = 1, \;\; \forall j \in \{1,2,3,...,n\} && (1) \\
& \sum_{j=1}^{n} x_{ij} = 1, \;\; \forall i \in \{1,2,3,...,n\} && (2) \\
& x_{ij} \in \{0,1\}, \;\; \forall (i,j) \in \{1,\,2,\,\ldots,\,n\} \times \{1,\,2,\,\ldots,\,n\} && (3)
\end{aligned}$$

Which is equivalent to the condensed summation form, $\sum_{i,\,j=1}^{n} \sum_{k,\,p=1}^{n} f_{ij} d_{kp} x_{ik} x_{jp}$.

Essentially, any x_{ij} represents facility i being allocated to location j.

Moreover, one can also write such a program in matrix form, utilising the matrix inner product for A, B, $n \times n$ matrices with real entries: $\langle A,\, B\rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{ij}$,

$$\min_{X \in P_n} \;\; \left\langle F,\, XDX^T \right\rangle$$

Where P_n describes the set of $n \times n$ permutation matrices, i.e. matrices whose entries satisfy constraints (1), (2) − or $n \times n$ matrices with n independent ones. We also note a number of linearisations of the QAP have been proposed [20] [16] [11] [1], turning the problem into a mixed integer program.

1.2 Literature Review

1.2.1 Koopmans-Beckman Formulation

As previously mentioned, the Koopmans-Beckman formulation, was the first mathematical formulation of the quadratic assignment problem, conceptualised in relation to economic planning decisions [17]. The objective was to maximise economic output, subject to inter-plant transportation costs of material, i.e. outputs of one plant may be a required input for another, thus incurring logistics costs.

The objective was to find an $n \times n$ permutation matrix, $P \in P_n$, maximising,

$$\sum_{k,i} a_{ki} p_{ki} \; - \sum_{k,l} \sum_{i,j} b_{kl} p_{ki} c_{ij} p_{lj}$$

Where each a_{ki} represents "semi-net" revenue of operation industry k at location i, each b_{kl} the commodity flow from industry k to industry l and c_{ij} the distance cost from location i to location j. In the original formulation, the distances c_{ij} must follow the triangle inequality.

One should note that the first summation represents a "linear" assignment problem, but modern formulations generally forgo this aspect to minimise the pure quadratic assignment problem given in the second term.

1.2.2 Computational Complexity & Exact Solution Methods

It is known that the Quadratic Assignment Problem is NP-Hard [25], where a reduction from the Hamiltonian cycle problem can also show that the problem is hard to approximate within a fixed factor of $\epsilon > 1$. Thus, making the Quadratic Assignment Problem hard to solve exactly, and even hard to approximate.

As a result, exact solution methods are generally limited to small instances. In the cases where exact methods were successful, algorithms generally fall into categories: branch and bound [15] [23], and cutting plane methods [4]. Some interesting exact methods have also been employed making use of quantum computing [29], where QAP instances are converted into a Quadratic Unconstrained Binary Optimization (QUBO) problem, which can be solved exactly on quantum computers.

For larger instances, exact solution methods fail to converge in a timely manner − and thus (meta)heuristics are generally preferred for solving more complex problems.

1.2.3 Heuristic Solution Methods

In tackling larger problems, in-exact solution methods are required. Previously, genetic algorithms together with population generation based on greedy heuristics have been experimented with in Ahuja et al. [2]. The authors conclude the genetic algorithm, with tournamenting and migration achieves optimality in most instances, and near-optimality in the remainder of instances. Despite finding optimal, or near-optimal solutions, their procedure has been observed to fail to converge in a timely manner. A number of ways to reduce computation time are presented by Gómez et al. [13], including GPU processing, batch processing as well as parallel processing.

Simulated annealing procedures have also been successful for finding good solutions [30] [8], both making use of the 2-exchange neighbourhood (or 2-OPT swap) in the terminology of TSP permutation encodings. A number of constructive heuristics have also had success, namely, [10], together with its own improvement heuristic. Indeed, one can also make use of local search as Heider [14], where a compromise between best improvement and first improvement is found.

Various aspects of neural computation have also been used in finding heuristic solutions for the QAP such as neural networks [7] − where instances are mapped to Boltzmann networks, then applying simulated annealing, or tabu search as an optimizer.

1.2.4 Related Problems

Basic extensions to the Quadratic Assignment Problem involve increasing the dimension of the problem, to say, the Cubic Assignment Problem [20],

$$\min_{X \in P_n} \sum_{i,j=1}^{n} \sum_{k,l=1}^{n} \sum_{m,p=1}^{n} c_{ijklmp} x_{ij} x_{kl} x_{mp}$$

For matrix $X \in P_n$ with entries x_{ij}. Which extends to the "N-adic" assignment problem. There is also the Bottleneck Quadratic Assignment Problem, which is given as a minimax optimisation problem,

$$\min_{X \in P_n} \max_{1 \leq i,j,k,l \leq n} \{f_{ij} d_{kl} x_{ik} x_{jl}\}$$

Moreover, there are a number of other problems which are special cases of the Quadratic Assignment Problem. Notably, the maximum clique problem and traveling salesman problem are special instances of the QAP.

In particular, the corresponding QAP instance for TSP instances is given when the distance matrix is the graph adjacency matrix and flow matrix the adjacency matrix of C_n, the cycle on n vertices.

2 Solution Methods

We now present a solution encoding and suite of (meta)heuristic algorithms for solving the Quadratic Assignment Problem.

2.1 Solution Encoding

Before discussing our suite of algorithms, we require a consistent solution encoding to represent the Quadratic Assignment Problem. Indeed, constraints (1) and (2) in the integer program given in **Section 1.1** and the subsequent discussion give rise to the natural *permutation matrix* encoding. In such an encoding, we require n independent ones, and each x_{ij} represents an assignment of facility i to location j.

$$\begin{bmatrix}1 & 0 & 0\\0 & 0 & 1\\0 & 1 & 0\end{bmatrix} \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} \begin{bmatrix}0 & 1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}$$

Figure 1: Examples of Feasible Instances for Permutation Matrix Encoding ($n = 3$)

We observe that for given n, we have exactly n non-zero entries in the permutation matrix $-$ meaning the proportion of nonzero entries $1/n \to 0$ as $n \to \infty$. Intuitively, one can rewrite sparse matrices into a more compact representation.

To convert between the matrix representation to compact representation, for permutation matrix P, we multiply P on the right by column vector $(1, 2, 3, \ldots, n)$ and take the tranpose. From the matrix encoding, we think of the entry for index i the location assigned to facility i. However, one may also take this as a facility being located to location i.

This gives us a *permutation* in the algebraic sense (element of symmetric group, S_n).

$$\begin{bmatrix}5 & 1 & 2 & 4 & 3 & 6\end{bmatrix}$$

Figure 2: Example of Feasible Instance for Permutation Encoding Method ($n = 6$)

The matrix encoding requires exactly $n^2 \in O(n^2)$ bits to store a solution, whereas, for the compact permutation encoding, we require $32n \in O(n)$ bits to store a solution with 32-bit integers. Thus, we have memory savings for $n > 32$, and can perform vector operations, rather than matrix operations in computation.

We proceed with the second encoding throughout this report.

2.2 Constructive Heuristic

Constructive Heuristics are a class of heuristics; initialised with an empty solution, and incrementally extended to produce a feasible solution. Indeed, constructive heuristics can be thought of as components in other heuristics – including GRASP and genetic algorithms.

2.2.1 Elshafei Constructive Heuristic

Elshafei [10] proceeds with the observation that the largest changes to the objective function come from pairs of facilities with high flow, assigned to disparate locations. Thus, Elshafei's constructive heuristic seeks to minimise such assignments.

Broadly, the heuristic begins by ordering locations based on their distance to other locations in ascending order and facilities based on their flow interactions with other facilities.

Assuming the solution is incomplete, we alternate between non-greedy phases, improvement, and semi-greedy phases.

While in the non-greedy phases, we pair available facilities and locations based on the ordering above, minimising downstream effects of large distances and large flows. Within the improvement phase, we swap the recently allocated facility with all other allocated facilities, attempting to find a partial solution with a better objective function. The pseudocode is given below,

Algorithm 1 Elshafei Constructive Improvement Phase

```
Input: $\sigma_0'$ – A partial solution
Input: $f_i$ – The recently assigned facility
Input: $F$ – Flow Matrix, $D$ – Distance Matrix
Output: $\sigma_0''$ – An improved solution
 1: $c_{\text{old}} \leftarrow c\,(\sigma_0',\, F,\, D)$
 2: $s_{\text{index}} \leftarrow f_i$
 3: for each assigned facility $j$ in $\sigma_0'$ do
 4:     Swap index $j$ with index $f_i$
 5:     $c_{\text{new}} \leftarrow c\,(\sigma_0',\, F,\, D)$
 6:     if $c_{\text{new}} < c_{\text{old}}$ then
 7:         $s_{\text{index}} \leftarrow j$
 8:         $c_{\text{old}} \leftarrow c_{\text{new}}$
 9:     end if
10:     Swap index $j$ with index $f_i$
11: end for
12: Swap index $s_{\text{index}}$ with index $f_i$
13: return $\sigma_0'$
```

In the semi-greedy phase, we choose the facility with maximum interaction with the recently allocated facility and pick a location that increases the objective function minimally.

The full pseudocode for Elshafei's constructive heuristic is given below:

Algorithm 2 Elshafei Constructive Heuristic

```
Input: $F$ − Flow Matrix, $D$ − Distance Matrix
Output: $\sigma_0$ − A feasible permutation solution
1: $\sigma_0 \leftarrow \varnothing$
2: $R_k \leftarrow \sum_{i=1}^{n} d_{ki}$  for all locations $k$
3: Order locations $k$ in ascending order of $R_k$
4:
5: $E_i^1 \leftarrow$ # of facilities with nonzero flow to or from $i$ for all facilities $i$
6: $E_i^2 \leftarrow -f_{ii} + \sum_{j=1}^{n} f_{ji} + \sum_{j=1}^{n} f_{ij}$  for all facilities $i$
7: Order facilities $i$ in descending order of $E_i^1$, $E_i^2$ separately
8: Order facilities $i$ by combining indices of $E_i^1$, $E_i^2$ in ascending order
9: while $\sigma_0$ is a partial solution do
10:     Assign next available facility, $f_1$ to next available location in rankings
11:     Improve($\sigma_0$, $f_1$, $F$, $D$)
12:     $S \leftarrow \{f_{f_1 i} + f_{i f_1} : i \text{ unallocated}\}$
13:     If $S = \varnothing$ then
14:     $f_2 \leftarrow \text{argmax}(S)$
15:     Greedily assign $f_2$ to an available location        ▷ Minimal change to cost
16:     Improve($\sigma_0$, $f_2$, $F$, $D$)
17: return $\sigma_0$
18: end while
19: return $\sigma_0$
```

Note that in the original paper, an improvement scheme is suggested, which we have replaced with a local search procedure.

2.2.2 A Scalable Constructive Heuristic

We observe that due to the improvement phase of Elshafei's constructive, it runs in quadratic time − and that achieving a good starting solution is comparable to a great starting solution after running some descent procedure, such as local search.

The following "scalable" heuristic gives a good starting solution, and relies on local search to improve the solution. Just as in Elshafei's heuristic, we proceed with the observation that we want to avoid putting facilities with high flow between them in disparate locations.

Additional considerations:

- Given the rankings, we can break ties in the order by considering the remaining sum, $\sum_{i=1}^{n} d_{ik}^2$ for locations and $\sum_{i=1}^{n} f_{ik}^2$ for facilities. Otherwise, use a lexicographic tie-break.
- Distance and flow matrices can be interchanged based on the matrix sum such that there is a higher "flow" for use in the rankings.

The pseudocode is given below:

Algorithm 3 Scalable Constructive Heuristic

Input: F – Flow Matrix, D – Distance Matrix

Output: σ_0: A feasible permutation solution

```
1: $R_k \leftarrow \sum_{i=1}^{n} d_{ki}^2$ for all locations $k$
2: $L_k \leftarrow \sum_{i=1}^{n} f_{ki}^2$ for all facilities $k$
3: Order locations in ascending order of $R_k$
4: Order facilities in descending order of $L_k$
5: for each facility in order of $R_k$ do
6:     Assign the current facility to the next available location
7: end for
```

2.2.3 A Stochastic Constructive Heuristic

The `Greedy Randomised Construction` function consists of two stages. The first stage is the assignment of two initial facilities to locations. To make these initial assignments, the distance values are ordered in increasing order, and the flow values are ordered in decreasing order. Suppose we have a ordered distances,

$$d_{i(1),j(1)} \leq ... \leq d_{i(n),j(n)}$$

and ordered flows

$$f_{k(1),l(1)} \geq ... \geq f_{k(n),l(n)}$$

The product of each element in this order is taken

$$d_{i(1),j(1)} f_{k(1),l(1)}, ..., d_{i(n),j(n)} f_{k(n),l(n)}$$

and the smallest element determines the two factories that are assigned to their corresponding two locations. That is, if the minimum product value is $d_{i(q),j(q)} f_{k(q),l(q)}$ then the initial assignments are factory $i(q)$ assigned to location $k(q)$ and factory $j(q)$ assigned to location $l(q)$.

The second stage is the random assignment of the remaining $n-2$ elements in the solution. During this stage, we enumerate all potential assignments between the unassigned factories and unassigned locations. At each enumeration, the cost of assigning factory F_j to location L_i. With all enumerations and their cost computed, the algorithm randomly chooses a pair out of the best 50% of the enumeration. Then, one assignment is chosen and the second stage is repeated until a feasible solution is reached.

2.3 Local Search Heuristics

2.3.1 Generic Local Search

To attain a better solution from the constructive heuristic, we can apply a search procedure on "neighbouring" solutions, i.e. local search. Indeed, given any feasible solution, we can apply such a search to reach a local minima with respect to the neighbourhood. A generic formulation of local search is given below.

Algorithm 4 Generic Local Search Heuristic

```
Input: $S_0$ - A feasible initial solution
Output: $S$ - A feasible optimal solution
1: improvement ← True
2: $S \leftarrow S_0$
3: while improvement do
4:     improvement ← False
5:     $N \leftarrow$ GenerateNeighbourhood($S$)
6:     $S' \leftarrow$ SelectNeighbour($N$)
7:     if $c(S') < c(S)$ then
8:         $S \leftarrow S'$
9:         improvement ← True
10:    end if
11: end while
12: return $S$
```

2.3.2 Neighbourhood Generation

From a single feasible permutation, σ, we propose the following neighbourhoods:

- Total (2-Exchange) Swap
- Adjacent Swap

The total swap neighbourhood defines all $\binom{n}{2} = n(n-1)/2 \in O(n^2)$ resultant permutations after swapping two facility locations − which can be sampled by sampling 2-combinations.

The adjacent swap neighbourhood defines all $n \in O(n)$ resultant permutations after swapping the locations of a facility i and facility $i+1$, with edge case mapping the nth facility to the first. Importantly, for large n, there are significant compute-time gains in using the adjacent swap neighbourhoods, but also significant performance tradeoffs.

2.3.3 Neighbourhood Selection

We make use of the best improvement, and first improvement schemes for neighbourhood selection − testing both schemes on our neighbourhoods.

2.3.4 Multistarts

Since local search is inherently an algorithm relying on exploitation, we may uniformly sample k permutations of size n as initial solutions and perform local search on each, keeping the best solution. In doing so, we promote exploration of the solution space. Such a procedure can be seen as a naive form of the GRASP algorithm mentioned in Section **2.4.2**.

2.4 Metaheuristics

2.4.1 Genetic Algorithm

Genetic algorithms were some of the first metaheuristics used in solving the QAP. Here, we proceed as usual with the permutation encoding.

First, we require some fitness function – computable for any permutation. Given some member of the population, represented by a permutation, we return the reciprocal of the associated objective function. To deal with division by zero errors, a small positive perturbation, ϵ is added to the objective function before computing the fitness.

Thus, using a proportional fitness assignment and roulette selection scheme, we select two parents to produce two offsprings until our generational population threshold is met.

Our offspring are created via two main genetic operators: a two-point partially mapped crossover, and 2-exchange swap mutation. We note these operators occur stochastically with given probabilities of crossover and probability of mutation. The framework in Azarbonyad and Babazadeh [3] recommends using a crossover probability of 0.8 and mutation rate of 0.2. However, we note this parameter can easily be tweaked to any value [2, 12, 18].

2.4.1.1 Two-Point Partially Mapped Crossover (PMX)

Two-Point Partially Mapped Crossover proceeds in two stages: the exchange of a continuous subsection of the parents to the offspring, then the mapping of genes from parent to offspring.

Since continuous subsections of permutations are completely determined by their endpoints, we may uniformly sample crossover points by sampling two indices in the permutation. Once sampled, we donate to the first offspring, the second parent's gene subsection, and the second offspring the first parent's gene subsection. An example is given below,

$$\begin{aligned}
\textbf{parent1} &: [0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7] \\
\textbf{parent2} &: [2,\ 6,\ 4,\ 0,\ 5,\ 7,\ 1,\ 3] \\
\textbf{offspring1} &: [-1,\ -1,\ -1,\ 0,\ 5,\ 7,\ -1,\ -1] \\
\textbf{offspring2} &: [-1,\ -1,\ -1,\ 3,\ 4,\ 5,\ -1,\ -1]
\end{aligned}$$

Figure 3: An Example of Crossover Stage of PMX

Now, examining the mapping stage: for each offspring, we attempt to add the remaining values outside of the section in order from the corresponding parents (parent one $\leftrightarrow$ offspring one, etc.). When we attempt to insert an element that already exists in the partial solution, we apply the mappings between the crossover section in order. An example is given below,

$$\begin{aligned}\textbf{parent1} &: [0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7]\\ \textbf{parent2} &: [2,\ 6,\ 4,\ 0,\ 5,\ 7,\ 1,\ 3]\\ \textbf{mappings} &: 3 \leftrightarrow 0,\ 4 \leftrightarrow 5,\ 5 \leftrightarrow 7\end{aligned}$$

Figure 4: An Example of Corresponding maps

Where application of the mappings corresponds to the product of transpositions i.e. attempting to insert 7 into offspring 2 will result in a map to 5, then map to 4. That is, we repeatedly apply the mappings when we are unable to insert the value.

$$\begin{aligned}\textbf{parent1} &: [0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7]\\ \textbf{offspring1} &: [3,\ 1,\ 2,\ 0,\ 5,\ 7,\ 6,\ 4]\end{aligned}$$

Figure 5: An Example of Offspring

Index	0	1	2	3	4	5	6	7
Mappings	$0 \to 3$	$1 \to 1$	$2 \to 2$	0	5	7	$6 \to 6$	$7 \to 5 \to 4$

Table 1: Mappings for Offspring 1

2.4.1.2 2-Exchange Mutation

Given an offspring, given as a permutation, the 2-exchange mutation occurs to promote exploration of the solution space. Uniformly, it selects a 2-combination, representing a swap between two indices in the offspring. Once selected, it applies the swap, resulting in a mutated offspring. An example of mutation is given below (shifted permutation):

$$\begin{aligned}\textbf{Prior Offspring} &: [10, 9, 6, 1, 8, 0, 11, 3, 5, 7, 4, 2]\\ \textbf{Mutated Offspring} &: [10, 9, 6, 1, 8, 0, 7, 3, 5, 11, 4, 2]\end{aligned}$$

Figure 6: An Example of Mutation (6, 9)

2.4.2 GRASP

The second metaheuristic implemented is the Greedy Randomised Adaptive Search Procedure (GRASP) method. The method has two key components, a greedy randomised construction procedure followed by a descent search procedure [28].

The pseudocode for GRASP, as in Oliveira et al. [24] is given below,

Algorithm 5 GRASP

```
Output: $S$
1: $c = \infty$
2: while stopping criterion not satisfied do
3:     $S' \leftarrow$ GreedyRandomisedConstruction()
4:     $S'' \leftarrow$ DescentSearch($S'$)
5:     if $f(S'') < c$ then
6:         $S \leftarrow S''$
7:         $c \leftarrow f(S'')$
8:     end if
9: end while
```

The `Greedy Randomised Construction` function is the Greedy Randomised Constructive step in GRASP, addressed in **Section 2.2.3**.

The `Local Search` function is the local search heuristic, addressed in **Section 2.3.1**.

2.4.3 GRASP & Simulated Annealing

We observe that GRASP can be modified to make use of any descent procedure after the initial construction. Indeed, in our experiments, we adapt GRASP to produce improvements in the initial solution by using simulated annealing.

Much like local search, simulated annealing is a descent algorithm. But unlike local search, we can accept worse feasible solutions with some probability that decreases as a function of the iterations.

The motivation in using Simulated Annealing is that we may escape local minima that we would otherwise be unable to escape in Local Search.

3 Experimental Results

3.1 Experiment Instances & Descriptions

Our experiments aim to use all the instances included in the QAP Library [6], with instance sizes ranging from $n = 12$ to $n = 256$. However, for the sake of analysis, we remove a number of instances without given solutions, giving a total of $N = 128$ instances to test. Removed instances are detailed in Appendix A.

We observe that in all groups, by default, the flow and distance matrices are symmetric. The exceptions are `bur` and `lipa` instance groups with some asymmetric flow matrices as well as `bur` and `tai` with asymmetric distance matrices.

An average of mean flow, mean distance and problem size values are given below, grouped by instances:

Instance Group	Mean Flow	Mean Distance	Mean Problem Size
bur	50.70	180.39	26
chr	4.67	32.43	18
els	82.50	1867.29	19
esc	0.63	1.25	27
had	3.32	4.71	16
kra	120.49	61.37	31
lipa	13.64	27.89	55
nug	2.84	2.74	20
rou	45.97	44.01	16
scr	180.17	2.47	16
sko	5.87	2.67	81
ste	1164.48	4.05	36
tai	246.31	1677.28	55
tho	5.56	52.30	73
wil	5.75	4.47	75

Table 2: Mean flow, mean distance and mean problem size by instance group

Table 2 shows the mean flow and mean distance of the diversity of the numbers in the input matrix in each instance group, with the average instance size. If we look at Table 2, the mean instance size is relatively spread across each group. However, the same is not said for the Mean flow and distance. Within the mean flow section the instance groups that have high mean flow in `ste`, `tai`, `scr` and `kra` but overall quite nominal to low flow in the other instance groups. For Mean Distance, there is a high average in `els`, `tai` and `bur`, but a relatively low average in other instance groups.

3.2 Description of Computation & Experiments

The numerical experiments were run in Python 3.12 on a Macbook Pro with M2 Pro, 32GB ram. **Further instructions on running the code is given in the "README" file**. Any code for the instance space analysis can be found in two Jupyter notebooks `feature_importance_for_algo_hardness.ipynb` and `feature_importance_for_best_algo.ipynb`.

We ran a number of experiments, including multistart local search with a varying number of multistarts, and varying generation/selection methods, Elshafei and scalable constructive heuristics with local search, GRASP with local search and simulated annealing as well as the genetic algorithm. A full detailed list of the experiments is available in the appendix.

3.3 Numerical Results

The relative error for Bur and Tai instance groups can be found in Tables 3 and 4, noting that the algorithms were only given a budget of 60 seconds to run − with some adjustments made to constructive heuristics and GRASP, ensuring a feasible solution is output.

The relative error for the other instances can be found in Appendix C.

Table 3: Relative Errors for Bur Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
bur26h	0.0080	0.0064	0.0411	0.0411	0.0002	0.0002	0	0.0613	0.0363
bur26e	0.0040	0.0084	0.0321	0.0321	0	0	0.0001	0.0609	0.0351
bur26d	0.0005	0.0001	0.0390	0.0390	0.0017	0.0017	0.0002	0.0690	0.0304
bur26f	0.0003	0.0133	0.0499	0.0499	0.0002	0.0002	0.0002	0.0642	0.0400
bur26g	0.0027	0.0000	0.0318	0.0318	0.0004	0.0004	0.0000	0.0398	0.0297
bur26c	0.0009	0.0007	0.0388	0.0388	0.0001	0.0001	0.0000	0.0628	0.0274
bur26b	0.0022	0.0055	0.0470	0.0470	0.0019	0.0019	0	0.0479	0.0289
bur26a	0.0034	0.0031	0.0269	0.0269	0.0017	0.0017	0.0010	0.0537	0.0319

Table 4: Relative Errors for Tai Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
tai35b	0.0195	0.0639	0.5272	0.5272	0.0161	0.0161	0.0086	0.4942	0.3616
tai15a	0.0555	0.0554	0.1029	0.1029	0.0221	0.0221	0.0154	0.1911	0.1149
tai35a	0.0331	0.0507	0.1175	0.1175	0.0349	0.0349	0.0235	0.1663	0.1505
tai15b	0.0075	0.0050	0.0157	0.0157	0.0014	0.0014	0.0000	0.0385	0.0122
tai80b	0.0319	0.0738	0.3820	0.3820	0.0295	0.0295	0.0496	0.4512	0.3696
tai150b	0.0370	0.0516	0.2570	0.2570	0.0287	0.0287	0.0338	0.2642	0.2507
tai80a	0.0365	0.0423	0.0945	0.0945	0.0388	0.0388	0.0368	0.1462	0.1307
tai100a	0.0381	0.0316	0.0921	0.0921	0.0293	0.0293	0.0329	0.1298	0.1202
tai25a	0.0487	0.0457	0.1244	0.1244	0.0338	0.0338	0.0309	0.1902	0.1398
tai100b	0.0572	0.0414	0.3958	0.3958	0.0195	0.0195	0.0350	0.4431	0.3535
tai25b	0.1925	0.1162	0.8259	0.8259	0.0127	0.0127	0.0186	1.1299	0.3516
tai64c	0.0019	0.0015	0.2817	0.2817	0.0009	0.0009	0.0000	0.2839	0.0740
tai12a	0.0280	0.0698	0.1339	0.1339	0.0614	0.0614	0.0000	0.3111	0.1240
tai12b	0.2735	0.2006	0.1356	0.1356	0.0000	0.0000	0.0285	0.4314	0.1499
tai50b	0.0645	0.0808	0.4513	0.4513	0.0338	0.0338	0.0140	0.5358	0.4405
tai50a	0.0380	0.0482	0.1110	0.1110	0.0393	0.0393	0.0371	0.1670	0.1429
tai30b	0.2762	0.2104	0.6285	0.6285	0.0227	0.0227	0.0258	0.7660	0.3403
tai30a	0.0600	0.0560	0.1154	0.1154	0.0348	0.0348	0.0314	0.1756	0.1461
tai40a	0.0482	0.0366	0.1142	0.1142	0.0352	0.0352	0.0330	0.1867	0.1489
tai17a	0.0551	0.0869	0.1191	0.1191	0.0274	0.0274	0.0207	0.2281	0.1313
tai256c	0.0067	0.6941	0.1439	0.1439	0.0029	0.0029	0.0038	0.2029	0.1246
tai60b	0.0820	0.0742	0.4996	0.4996	0.0272	0.0272	0.0287	0.5024	0.4105
tai40b	0.1696	0.0307	0.4970	0.4970	0.0296	0.0296	0.0493	0.5909	0.4077
tai60a	0.0380	0.0384	0.1070	0.1070	0.0398	0.0398	0.0355	0.1602	0.1416
tai20a	0.0429	0.0832	0.1231	0.1231	0.0299	0.0299	0.0245	0.2024	0.1190
tai20b	0.0103	0.2990	0.4720	0.4720	0.0107	0.0107	0.0127	0.9123	0.0943

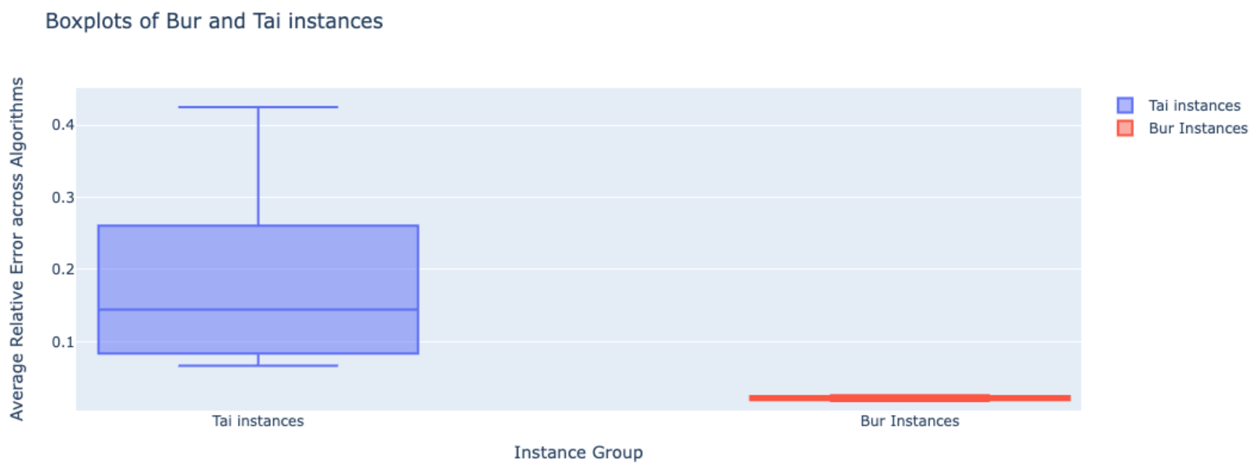

Figure 7: Average Relative Error for Tai and Bur instances

As seen in Figure 7, the average relative error across algorithms for the Bur instances were far smaller than the Tai instances. This may be attributed to the fact that on average, the Bur instances have a problem size of 26, which is far smaller than the Tai average problem size of 55.

It should also be noted that the number of Bur instances is 8, whilst the number of Tai instances is 26.

Instance Groups	Average Relative Error (%)
Bur	0.0155
Chr	10.6872
Els	0
Esc	0
Had	0
Kra	2.2595
Lipa	5.2986
Nug	0.8402
Rou	1.1646
Scr	1.5748
Sko	1.3338
Ste	2.9660
Tai	1.7954
Tho	1.6531
Wil	0.6259

Table 5: Average Relative Error for each Instance Group

Instance Name	Average RE(%)	Instance Name	Average RE(%)	Instance Name	Average RE(%)	Instance Name	Average RE(%)
Bur26a	0.1007	Tai80a	3.1804	Had16	0	Nug24	0.7454
Bur26b	0	Tai80b	2.6018	Had18	0	Nug25	0.8013
Bur26c	0.0029	Chr12a	5.6951	Had20	0	Nug27	1.6813
Bur26d	0.002	Chr12b	0	Kra30a	2.5422	Nug28	1.3163
Bur26e	0	Chr12c	5.3962	Kra30b	1.7392	Nug30	1.1430
Bur26f	0.0178	Chr15a	0	Kra32	2.4972	Rou12	1.1065
Bur26g	0.0004	Chr15b	13.7171	Lipa20a	1.5477	Rou15	0.6900
Bur26h	0	Chr15c	19.2551	Lipa20b	0	Rou20	1.6973
Tai100a	2.9278	Chr18a	20.3100	Lipa30a	1.6087	Scr12	0
Tai100b	1.9541	Chr18b	1.4243	Lipa30b	0	Scr15	3.8483
Tai12a	0	Chr20a	18.2482	Lipa40a	1.2905	Scr20	0.8761
Tai12b	0	Chr20b	16.5361	Lipa40b	15.8181	Sko100a	1.1605
Tai150b	2.2731	Chr20c	13.7463	Lipa50a	1.1193	Sko100b	1.1775
Tai15a	1.5388	Chr22a	3.3788	Lipa50b	0	Sko100c	1.4527
Tai15b	0	Chr22b	5.8767	Lipa60a	0.9588	Sko100d	1.2261
Tai17a	2.0711	Chr25a	26.0274	Lipa60b	19.0028	Sko100e	1.5300
Tai20a	2.4467	Els19	0	Lipa70a	0.8671	Sko100f	1.4292
Tai20b	0	Esc128	0	Lipa70b	20.0718	Sko42	1.5811
Tai256c	0.2801	Esc16a	0	Lipa80a	0.7579	Sko49	1.6506
Tai25a	3.0874	Esc16b	0	Lipa80b	21.0219	Sko56	1.0622
Tai25b	1.2739	Esc16c	0	Lipa90a	0.7126	Sko64	1.4434
Tai30a	2.7375	Esc16d	0	Lipa90b	0	Sko72	1.3704
Tai30b	0.7666	Esc16e	0	Nug12	06920	Sko81	1.0857
Tai35a	2.3509	Esc16f	0	Nug14	0.7890	Sko90	1.1702
Tai35b	0.8471	Esc16g	0	Nug15	0.1739	Ste36a	3.7161
Tai40a	3.3041	Esc16h	0	Nug16a	0.7453	Ste36b	2.0439
Tai40b	2.5415	Esc16i	0	Nug16b	0	Ste36c	3.1380
Tai50a	3.20649	Esc16j	0	Nug17	1.1547	Tho150	1.545
Tai50b	1.99273	Esc32e	0	Nug18	0.7254	Tho30	1.9222
Tai60a	3.28142	Esc32g	0	Nug20	1.4786	Tho40	1.8826
Tai60b	1.95831	Had12	0.24213	Nug21	0.6563	Wil100	0.6373
Tai64c	0	Had14	0	Nug22	0.5006	Wil50	0.6146

Table 6: Average Relative Error (RE) of best solution for each Instance limited to 60 seconds

Heuristic	Percentage of Total Instance(%)
GRASP with Local Search	71.54
$k = 10$ Multistart with Total Swap Neighbour	21.54
Constructive Greedy with Local Search	5.38
Elshafei Constructive with Greedy Local Search	1.54

Table 7: Proportion of all Instances where Heuristic was best

3.4 Discussion

The average relative error for most instances in table 6 were within 4% of optimality, aside from a number of outliers: including the majority of the `chr` instance group, and individual instances `lipa40b`, `lipa60b`, `lipa70b` and `lipa80b`. Indeed, we notice that in these instances of `lipa`, the flow matrix is asymmetric.

On the other hand, we qualitatively assess that the `chr` instance group has sparser flow matrices than other instance groups – indicating lower optimal values. To this end, we intuit that the solution landscape is more "rugged", as a misassignment of a facility, not taking proper advantage of 0 flow values will jump to a drastically higher objective value. However, more analysis is required to make definitive conclusions about the ruggedness of the solution space, but we do see some value to be had in a more rigorous instance space approach to systematically determine features of interest.

We also find our algorithms obtained solutions at, or near-optimal for the `els`, `esc` and `has` instance groups. In Table 6, the `sko` instance group had have a range of large instance sizes from 42 to 100, though their gaps was well within acceptable ranges of 1% - 2 %, implying problem size is not sufficient to rule a problem "hard".

Table 7 partition instances into their "best" algorithm, as determined by relative error/optimality percentage gap. We see that both GRASP and multistart total swap methods were best with a percentage of 71.54% and 21.54%, respectively, where we carefully note their extensive use of local search. Indeed, to a lesser extend, both constructive heuristics also have a significant reliance on local search, which was the slowest to converge, especially for large instances. Further analysis on the convergence and "early-stopping" would allow for strong heuristics under better computation time.

3.5 Convergence of Local Search Methods

Despite satisfactory solutions from local-search procedures given a time budget of 60 seconds, we observed that the time to converge is far too high, especially for larger instances such as `tai256c`. Below, we perform some analyses on the convergence of local search for `tai256c`.

We run Local Search with total swap neighbourhoods for a sufficient number iterations ($N = 30$). Such a process is then repeated five times with different initial solutions, sampled randomly from the solution space. The mean performance at each iteration is computed, and the results can be seen in Figure 8.

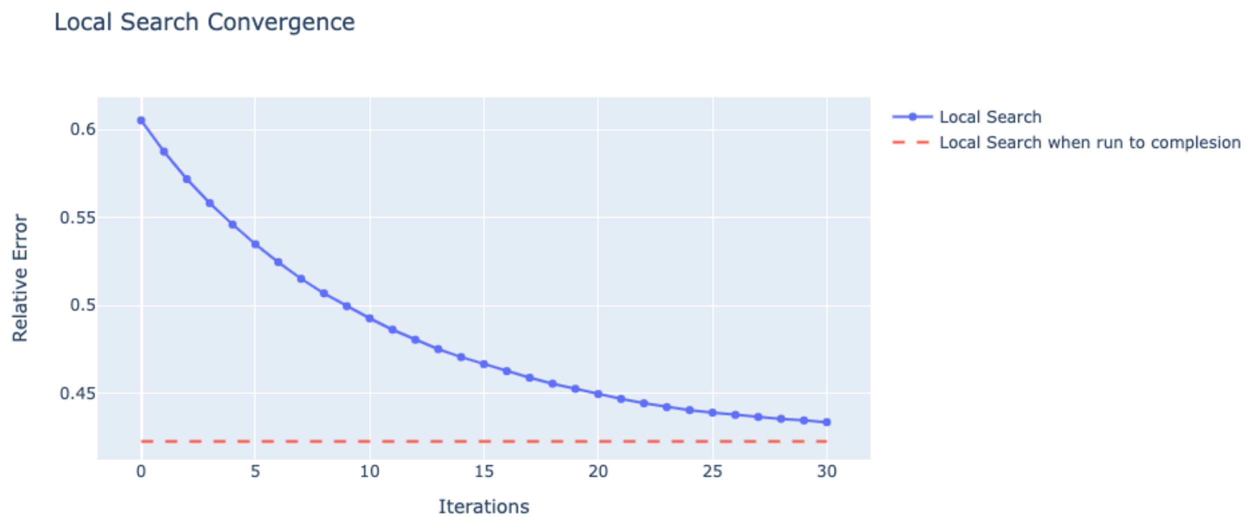

Figure 8: Convergence of Local Search on Tai256c instance

We can see that the Local Search algorithm asymptotes towards the relative error of it's solution when run to completion. For this particular instance, however, Local Search would take on average between 40 to 60 iterations to run to completion, indicating a large performance improvement to be gained from early stopping, with little losses in performance – explaining the good performance with respect to a time budget.

Moreover, empirically, we observed the adjacent swap neighbourhood was too small to achieve meaningful descent progress given a random starting permutation. Rather, there is rarely any descent from the starting permutation, making the use of adjacent swap more akin to a completely random search process, rather than any proper descent method.

3.6 Instance Space Analysis

Following other machine-learning enabled methods of algorithm analysis, [26] [27], we seek to provide a quantitative exploration of our instance space for the QAP.

Indeed, some empirical studies have been applied to the QAP [22], similar to our own where the authors make claims based on their test instances such as finding "effectiveness of the multigreedy approach, thus of the local search operator", conjecturing that, "any approach based on local search is bound to be very effective as an heuristic for QAP".

Thus, our goal of instance space analysis is two-fold, in exploring "hard" instances, as well as understanding what algorithm provides the best performance for an instance, extending prior works in empirical algorithm evaluations.

The analysis performed is restricted to a subset of the algorithms that were produced. The algorithms considered include

- GRASP with local search
- GRASP with Simulated Annealing
- Genetic Algorithm
- Multistart local search (total swap, optimal neighbour)
- Multistart local search (adjacent swap, optimal neighbour)
- Multistart local search (total swap, first improvement neighbour)
- Multistart local search (adjacent swap, first improvement neighbour)
- Local search (constructive greedy initial solution)
- Local search (Elshafei constructive greedy initial solution)

These are run on all instances considered above. Unlike the numerical results above which are restricted such that each algorithm produces an output after 60 seconds, the results in the instance space analysis allow each algorithm to run to completion.

As described above, the instance space analysis explores the features that best predict if an instance is "hard" and which algorithm provides the best relative error for that instance. In service of this, features of instances are engineered as to encode the characteristics of each instance. These features include

- **Problem size** - The number of factories/location
- **Distance/flow sparsity** - The percentage of zeros in the distance/flow matrix
- **Distance/flow asymmetry** - The percentage of non-symmetric items in the distance/flow matrix

- **Distance/flow dominance** - The standard deviation of the distance/flow matrix over its mean
- **Distance/flow max, min, median and mean** - The max, min, median and mean values in the distance/flow matrix.

Analysing instance hardness

First we must find a way to determine if an instance is categorised as a “hard” or “easy”. To do so, K-Means clustering is used to categorised our data into two groups based on the algorithms performance on relative error. The groups seem to be fairly imbalanced where 11 instances are considered in one group and 117 are in another.
PCA is used to reduce the dimension of the relative error of the algorithms to two PC dimension. This visualisation, as seen in Figure 9, serves to confirm that the clustering is appropriately distinguishing between two groups within our data.

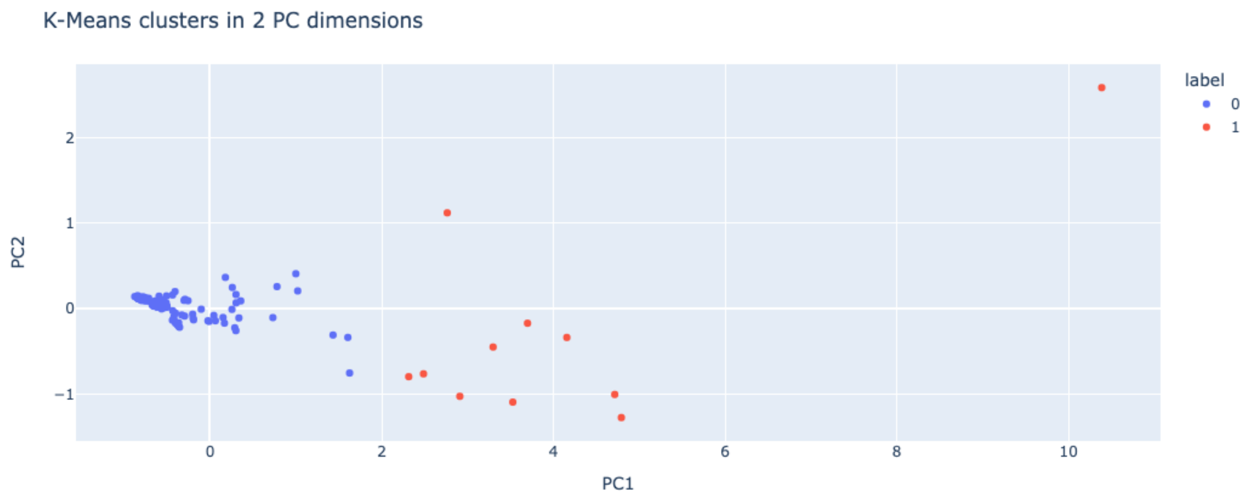

Figure 9: K-Means clustering of algorithm relative error in 2 PC dimensions

In support of this, we can consider the relative error between two labels for a single algorithm. As an example, we look at GRASP (with Simulated Annealing) and the Genetic Algorithm. As seen in Figures 10 and 11, the distribution of relative errors for these two algorithms in group 1 looks to be substantially greater than the distribution of relative errors for these algorithms in group 0.

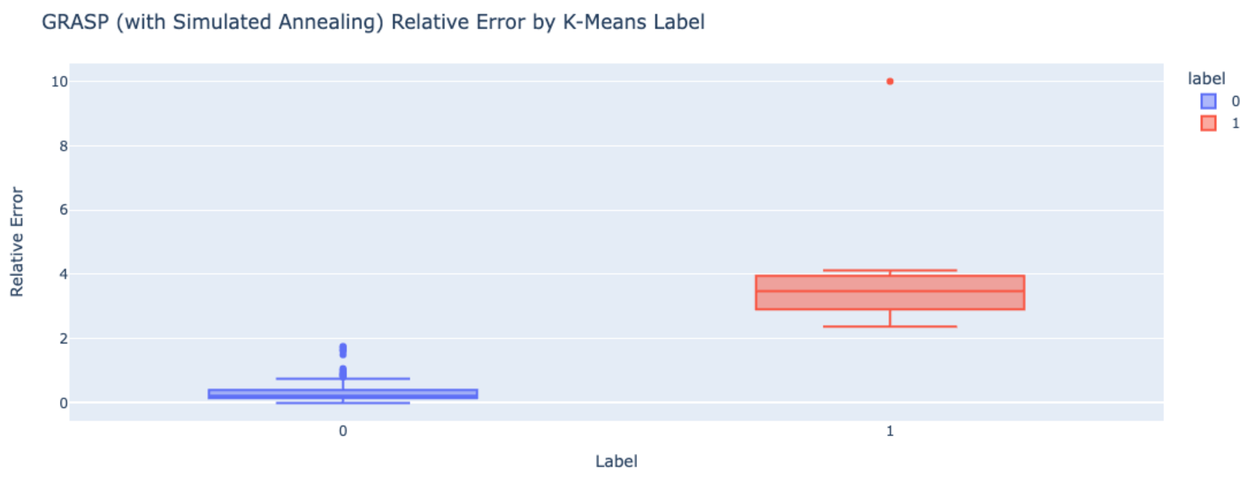

Figure 10: GRASP (with Simulated Annealing) Relative Error by K-Means Label

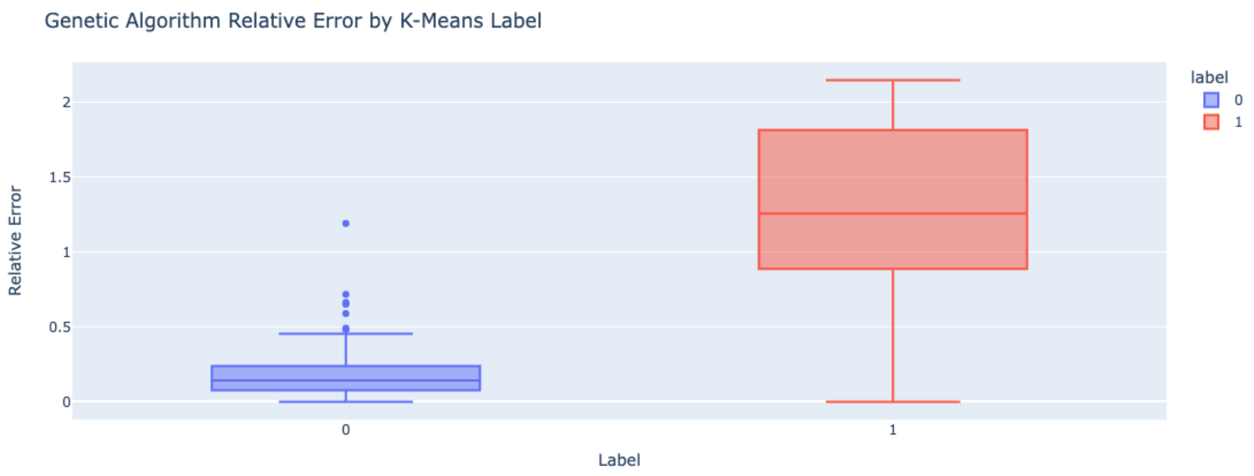

Figure 11: Genetic Algorithm Relative Error by K-Means Label

By combining the labels as determined by K-Means clustering with the features engineered from the instances, we produce the key data used to analyse our data. This data set is then split into a 70/30 train/test split. The training data set is used to train a logistic regression model, where all the engineered features are regressed upon the label of that instance.
The testing performance can be found in Table 8, where we can see that the number of false negatives is 0, the number of false positives is 4 and the prediction accuracy is 89.74%. These values appear to suggest a successfully trained model that provides acceptable model accuracy.
A key consideration here is how the class imbalance influences our regression model and its predictive accuracy. To balance this, we use a weighted Logistic Regression, where the weight of the instances with label 1 is three times greater than the instances with label 0, which places greater loss in the Logistic Regression for incorrectly predicting a 1. Other

approaches such as under-sampling and over-sampling can be used. Based on the dataset we would suggest over-sampling since there is not a great deal of data. This would involve taking samples, with replacement, from instances in the 1 label group.

False Negatives (of 39)	**False Positives (of 39)**	**Prediction Accuracy**
0	4	89.74%

Table 8: Logistic Regression Results

Next we investigate the features used in the regression model. Table 9 contains the p-values obtained in the logistic regression. Based on the p-values present, we can see that the only statistically significant feature is flow dominance.

	coef	**std err**	**z**	**P > |z|**	**[0.025**	**0.975]**
problem_size	0.8014	0.549	1.461	0.144	-0.274	1.876
flow_sparsity	-0.4746	0.501	-0.948	0.343	-1.456	0.507
distance_sparsity	-0.7383	0.593	-1.246	0.213	-1.900	0.423
flow_asymmetry	-0.0620	0.284	-0.219	0.827	-0.618	0.494
distance_asymmetry	0.5764	0.754	0.765	0.444	-0.900	2.053
flow_dominance	2.6617	0.648	4.109	0.000	1.392	3.931
distance_dominance	1.7120	1.250	1.369	0.171	-0.739	4.163
flow_max	-0.4589	0.424	-1.083	0.279	-1.290	0.372
distance_max	-1.1537	1.586	-0.727	0.467	-4.263	1.955
flow_min	0.3721	1.029	0.362	0.718	-1.644	2.388
distance_min	0.4953	3.412	0.145	0.885	-6.191	7.182
flow_mean	-0.2572	0.825	-0.312	0.755	-1.875	1.360
distance_mean	-0.4807	2.584	-0.186	0.852	-5.546	4.584

Table 9: Summary Statistics for Features in the Logistic Regression

Analysing the algorithm with the best performance by instance

Now we want to understand what the best algorithm is based on the features of an instance. To do so we first find the algorithm that has the lowest relative error by instance. The distribution of the relative errors for the 'best' algorithms can be found in Figure 12. We can see that GRASP with local search seems to find the best relative error for instances with generally low relative error. Mutli-start local search and elshafei constructive local search seem to also perform similarly. Interestingly, constructive greedy local search provides the best performance on instances that tend to have greater relative errors across all algorithms.

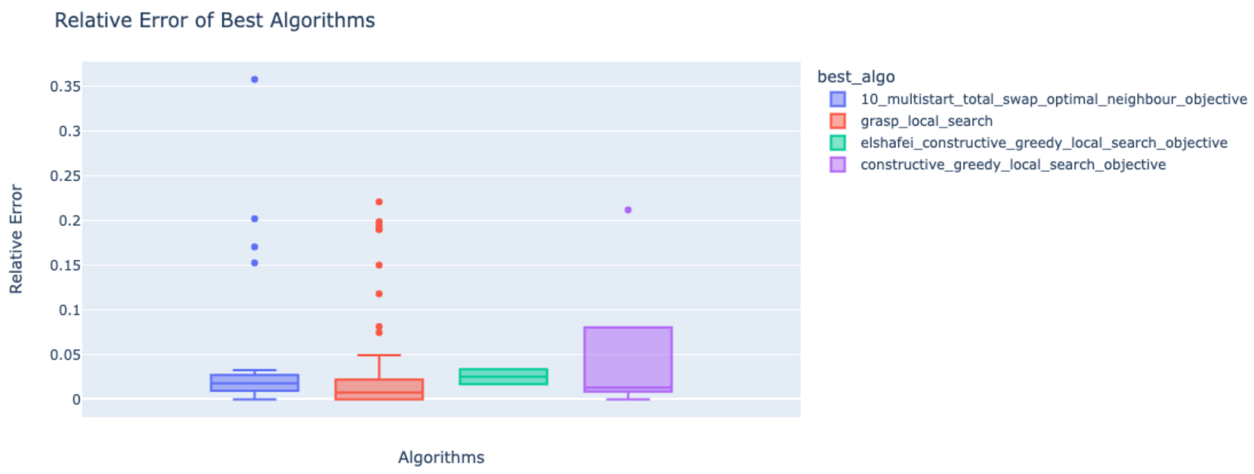

Figure 12: Summary Statistics for features in the Logistic Regression

Once the best algorithm label is assigned and combined with the same feature data as above, we have our main dataset for this analysis. This data, as above, is split into a train and test sets which is trained on a Multivariate Logistic Regression. The GRASP with local search algorithm is the baseline for the regression, meaning the regression is providing predictions if an other algorithm should be used in place of GRASP with local search. The relative statistical summaries can be found in Tables 10, 11 and 12. One should also observe above, the lower variance of both constructive heuristics.

Table 10: Summary Statistics for features in the Logistic Regression ($y = 1$)

	coef	std err	Z	P> $|z|$	[0.025	0.975]
problem_size	1.2693	0.773	1.641	0.101	-0.247	2.785
flow_sparsity	4.1683	2.146	1.942	0.052	-0.039	8.375
distance_sparsity	-1.1320	0.902	-1.255	0.210	-2.900	0.636
flow_asymmetry	-0.0454	0.360	-0.126	0.900	-0.752	0.661
distance_asymmetry	0.2583	0.891	0.290	0.772	-1.487	2.004
flow_dominance	-0.0464	0.638	-0.160	0.871	-3.563	0.316
distance_dominance	-0.0157	1.390	-0.011	0.991	-2.741	2.710
flow_max	-1.9018	1.018	-1.867	0.062	-3.898	0.094
distance_max	0.6431	1.801	0.357	0.721	-2.887	4.173
flow_min	8.5602	3.755	2.280	0.023	1.200	15.920
distance_min	-2.9238	3.977	-0.735	0.462	-10.719	4.871
flow_mean	-5.1331	2.577	-1.992	0.046	-10.184	-0.082
distance_mean	1.6642	3.113	0.535	0.593	-4.437	7.765

Table 11: Summary Statistics for features in the Logistic Regression ($y = 2$)

	coef	std err	Z	P> $\|z\|$	[0.025	0.975]
problem_size	0.9338	0.815	1.146	0.252	-0.663	2.531
flow_sparsity	4.0790	2.096	1.946	0.052	-0.030	8.188
distance_sparsity	-0.8726	0.907	-0.962	0.336	-2.651	0.906
flow_asymmetry	0.1555	0.419	0.371	0.711	-0.665	0.976
distance_asymmetry	0.0510	0.855	0.060	0.952	-1.625	1.727
flow_dominance	-1.3775	0.889	-1.549	0.121	-3.120	0.365
distance_dominance	-0.1198	1.338	-0.090	0.929	-2.742	2.503
flow_max	-1.9246	1.016	-1.894	0.058	-3.916	0.067
distance_max	0.7850	1.721	0.456	0.648	-2.589	4.159
flow_min	8.4038	3.748	2.242	0.025	1.058	15.749
distance_min	-3.3204	3.804	-0.873	0.383	-10.776	4.135
flow_mean	-5.0967	2.572	-1.982	0.048	-10.137	-0.056
distance_mean	1.9823	2.987	0.664	0.507	-3.872	7.836

Table 12: Summary Statistics for features in the Logistic Regression ($y = 3$)

	coef	std err	Z	P> $\|z\|$	[0.025	0.975]
problem_size	0.8149	0.766	1.064	0.287	-0.686	2.316
flow_sparsity	4.7725	2.159	2.210	0.027	0.540	9.005
distance_sparsity	-0.3081	0.882	-0.349	0.727	-2.038	1.422
flow_asymmetry	0.3670	0.375	0.977	0.328	-0.369	1.103
distance_asymmetry	-1.5232	1.106	-1.377	0.168	-3.691	0.645
flow_dominance	-2.3637	1.019	-2.320	0.020	-4.360	-0.367
distance_dominance	-2.8415	1.824	-1.558	0.119	-6.417	0.734
flow_max	-2.5966	1.170	-2.219	0.026	-4.890	-0.303
distance_max	4.0860	2.607	1.567	0.117	-1.023	9.195
flow_min	12.0619	4.673	2.581	0.010	2.903	21.221
distance_min	-10.9638	6.325	-1.733	0.083	-23.360	1.432
flow_mean	-7.8956	3.785	-2.086	0.037	-15.315	-0.476
distance_mean	7.4689	5.021	1.488	0.137	-2.372	17.310

Tables 10 and 11 show us that the flow min and flow mean are the only statistically significant variables that are able to predict if the best algorithm for an instance is constructive greedy local search or elshafei constructive greedy local search objective rather than GRASP with local search. In both cases, we have a positive coefficient for flow min and a negative coefficient for flow mean. We can interpret this as for instances with a relatively smaller flow min value and a relatively larger flow mean value, on average it is more likely that GRASP with local search will out-perform constructive greedy local search or elshafei constructive

greedy local search.
Figure 12 shows us that the flow sparsity, flow dominance, flow max, flow min and flow mean are the statistically significant variables that are able to predict if the best algorithm for an instance is multistart local search with total swap optimal neighbourhoods rather than GRASP with local search. Flow sparsity and flow min have positive coefficient values, which means that for instances with greater flow sparsity or greater flow min values, on average the likelihood of the best algorithm for that instance being the multi-start local search is greater than the likelihood of being GRASP with local search. Additionally, the coefficient for flow dominance, flow max and flow mean are negative. This means that for instances with greater flow dominance, flow max or flow mean min values, on average the likelihood of the best algorithm for that instance being the multi-start local search is lower than the likelihood of being GRASP with local search.

4 Conclusions, Improvements & Limitations

4.1 Improvements & Limitations

An immediate improvement would be the use of a greater breadth of QAP instances outside of QAPLIB, which fundamentally limited our analysis and robustness of the experiments. Indeed, this would find most use in the instance space analysis, where more data of more varied features can paint a better picture of the hardness of some instances and allow for better training. This is most apparent with the symmetry feature: where most groups of data have symmetric flow, and distance matrices − yet we note this is not always assumed. One can circumvent the lack of data and given solutions by generating new instances as in Li and Pardalos [21].

Moreover, various other features can be engineered for greater depth of analysis: including whether the distance matrix abides by the triangle inequality, as was assumed in Koopmans and Beckmann [17] or any of the other metric qualities. To this end, one may ask, is the Δ-QAP (Metric QAP) significantly easier than the general problem?

As for our heuristics suite, various improvements can be made to the limited combinations of metaheuristics, especially the breadth of genetic operators we experimented on. The genetic algorithm is restricted to a two-point PMX crossover and mutation. To further promote the exploitation of strong candidate solutions, we may employ elitism, or greedily build our initial population with the stochastic constructive heuristic. Furthermore, a number of different mutations could also have been tested, including k-swap neighbourhoods, or inversion mutations, as given in the TSP literature [19]. In a similar light, our GRASP-SA implementation can also be adjusted with various other cooling schedules.

Other types of heuristics could be considered as well, including various relaxations and bounds which have been extensively researched, then combined with a beam search process, would reduce the computational complexity from full branch and bound.

4.2 Conclusion

The Quadratic Assignment Problem (QAP), despite its simple integer program model, is extremely difficult to solve exactly, or even approximately in reasonable time. Thus, with the use of (meta)heuristics, we settle for "good" solutions, rather than perfect ones. Indeed, in this project, we designed and implemented a suite of (meta)heuristics from within, and without the literature. A number of constructive heuristics and local search heuristics were tested, as well as various configurations of GRASP and Genetic Algorithms.

Our results conclude that both GRASP and multi-start local search were good heuristics in obtaining good solutions given a time budget of 60 seconds but also required higher computational power to converge. Indeed, we also see we were able to produce good solutions with low variance by combining fast constructive heuristics (Elshafei and scalable) with local search. Indeed, via analysis of convergence, we find that local search, while slow to converge, can be given a budget or some other early stopping rule to improve computation time without sacrificing performance.

Finally, through a streamlined instance space analysis, we were able to statistically determine a key feature in "hard" instances, the flow dominance − which is consistent with the literature. We were also able to find features relating to the effectiveness of various algorithms, including: Elshafei's constructive heuristic, GRASP, 10-multistart local search and the scalable constructive heuristic.

A Instance Details & Experiment Details

A.1 Instance Details

The project aims to use all instances in the QAP library. However, some instances, namely `tai10b.dat`, `tai10a.dat`, `esc64a.dat`, `esc32h.dat`, `esc32a.dat`, `esc32b.dat`, `esc32c.dat`, `esc32d.dat` did not contain their corresponding solutions. For the analysis of the results, the group omitted the problem instances mentioned. With the removal of the missing solutions to the data instances totaling to 128 usable instances.

A.2 Experiment Details

Multi-Start Local Search:

$k = 10$ multi-starts for both total neighbourhoods and adjacent neighbourhoods $-$ combined with both best improvement and first improvement schema.

Genetic Algorithm:

Initial population size $N_{\text{pop}} = 100$, Maximum iterations $MaxIter = 150$, the probability of crossover operation $p_{\text{crossover}} = 0.8$ and the probability of mutation operation $p_{\text{mutation}} = 0.2$.

Simulated Annealing:

The maximum number of iterations $n_{\text{iter}} = 500$, initial temperature $temp = 100$, cooling rate $\alpha = 0.99$ for a geometric cooling schedule.

GRASP:

The maximum number of iterations $n_{\text{iter}} = 25$ with local search and simulated annealing descent procedures.

B Full Results

Table 13: Relative Errors for Bur Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
bur26h	0.0080	0.0064	0.0411	0.0411	0.0002	0.0002	0	0.0613	0.0363
bur26e	0.0040	0.0084	0.0321	0.0321	0	0	0.0001	0.0609	0.0351
bur26d	0.0005	0.0001	0.0390	0.0390	0.0017	0.0017	0.0002	0.0690	0.0304
bur26f	0.0003	0.0133	0.0499	0.0499	0.0002	0.0002	0.0002	0.0642	0.0400
bur26g	0.0027	0.0000	0.0318	0.0318	0.0004	0.0004	0.0000	0.0398	0.0297
bur26c	0.0009	0.0007	0.0388	0.0388	0.0001	0.0001	0.0000	0.0628	0.0274
bur26b	0.0022	0.0055	0.0470	0.0470	0.0019	0.0019	0	0.0479	0.0289
bur26a	0.0034	0.0031	0.0269	0.0269	0.0017	0.0017	0.0010	0.0537	0.0319

Table 14: Relative Errors for Tai Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
tai35b	0.0195	0.0639	0.5272	0.5272	0.0161	0.0161	0.0086	0.4942	0.3616
tai15a	0.0555	0.0554	0.1029	0.1029	0.0221	0.0221	0.0154	0.1911	0.1149
tai35a	0.0331	0.0507	0.1175	0.1175	0.0349	0.0349	0.0235	0.1663	0.1505
tai15b	0.0075	0.0050	0.0157	0.0157	0.0014	0.0014	0.0000	0.0385	0.0122
tai80b	0.0319	0.0738	0.3820	0.3820	0.0295	0.0295	0.0496	0.4512	0.3696
tai150b	0.0370	0.0516	0.2570	0.2570	0.0287	0.0287	0.0338	0.2642	0.2507
tai80a	0.0365	0.0423	0.0945	0.0945	0.0388	0.0388	0.0368	0.1462	0.1307
tai100a	0.0381	0.0316	0.0921	0.0921	0.0293	0.0293	0.0329	0.1298	0.1202
tai25a	0.0487	0.0457	0.1244	0.1244	0.0338	0.0338	0.0309	0.1902	0.1398
tai100b	0.0572	0.0414	0.3958	0.3958	0.0195	0.0195	0.0350	0.4431	0.3535
tai25b	0.1925	0.1162	0.8259	0.8259	0.0127	0.0127	0.0186	1.1299	0.3516
tai64c	0.0019	0.0015	0.2817	0.2817	0.0009	0.0009	0.0000	0.2839	0.0740
tai12a	0.0280	0.0698	0.1339	0.1339	0.0614	0.0614	0.0000	0.3111	0.1240
tai12b	0.2735	0.2006	0.1356	0.1356	0.0000	0.0000	0.0285	0.4314	0.1499
tai50b	0.0645	0.0808	0.4513	0.4513	0.0338	0.0338	0.0140	0.5358	0.4405
tai50a	0.0380	0.0482	0.1110	0.1110	0.0393	0.0393	0.0371	0.1670	0.1429
tai30b	0.2762	0.2104	0.6285	0.6285	0.0227	0.0227	0.0258	0.7660	0.3403
tai30a	0.0600	0.0560	0.1154	0.1154	0.0348	0.0348	0.0314	0.1756	0.1461
tai40a	0.0482	0.0366	0.1142	0.1142	0.0352	0.0352	0.0330	0.1867	0.1489
tai17a	0.0551	0.0869	0.1191	0.1191	0.0274	0.0274	0.0207	0.2281	0.1313
tai256c	0.0067	0.6941	0.1439	0.1439	0.0029	0.0029	0.0038	0.2029	0.1246
tai60b	0.0820	0.0742	0.4996	0.4996	0.0272	0.0272	0.0287	0.5024	0.4105
tai40b	0.1696	0.0307	0.4970	0.4970	0.0296	0.0296	0.0493	0.5909	0.4077
tai60a	0.0380	0.0384	0.1070	0.1070	0.0398	0.0398	0.0355	0.1602	0.1416
tai20a	0.0429	0.0832	0.1231	0.1231	0.0299	0.0299	0.0245	0.2024	0.1190
tai20b	0.0103	0.2990	0.4720	0.4720	0.0107	0.0107	0.0127	0.9123	0.0943

Table 15: Relative Errors for Chr Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
chr22a	0.0338	0.0614	0.3691	0.3691	0.0676	0.0676	0.0627	0.5708	0.4418
chr18a	0.5010	0.5147	1.8580	1.8580	0.4475	0.4475	0.2031	3.4891	1.9569
chr22b	0.0988	0.1372	0.4191	0.4191	0.1011	0.1011	0.0746	0.6790	0.3755
chr18b	0.0952	0.1708	0.5072	0.5072	0.0391	0.0391	0.0143	1.2595	0.5085
chr15a	0.7749	0.4553	1.0610	1.0610	0.2205	0.2205	0	2.8424	0.8517
chr15b	0.2783	0.4974	1.0981	1.0981	0.4556	0.4556	0.1469	4.9835	0.9937
chr15c	0.6128	0.2618	1.4030	1.4030	0.3476	0.3476	0.1926	3.8636	1.4396
chr12a	0.1954	0.4768	0.6413	0.6413	0.1723	0.1723	0.0570	2.2942	0.5492
chr12c	0.1379	0.0753	0.6119	0.6119	0.1579	0.1579	0.0540	1.4830	0.3107
chr12b	0.5791	0.9037	0.7577	0.7577	0	0	0	1.6038	0.1246
chr25a	0.3351	0.5870	1.5664	1.5664	0.5316	0.5316	0.2603	2.7945	1.8983
chr20a	0.4498	0.5392	0.8714	0.8714	0.2619	0.2619	0.2491	2.8887	1.6359
chr20c	0.7879	0.5227	1.9294	1.9294	0.4193	0.4193	0.4122	4.8900	2.2515
chr20b	0.3194	0.4256	1.2524	1.2524	0.1662	0.1662	0.1654	2.5666	1.2167

Table 16: Relative Errors for Esc Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
chr22a	0.0338	0.0614	0.3691	0.3691	0.0676	0.0676	0.0627	0.5708	0.4418
chr18a	0.5010	0.5147	1.8580	1.8580	0.4475	0.4475	0.2031	3.4891	1.9569
chr22b	0.0988	0.1372	0.4191	0.4191	0.1011	0.1011	0.0746	0.6790	0.3755
chr18b	0.0952	0.1708	0.5072	0.5072	0.0391	0.0391	0.0143	1.2595	0.5085
chr15a	0.7749	0.4553	1.0610	1.0610	0.2205	0.2205	0	2.8424	0.8517
chr15b	0.2783	0.4974	1.0981	1.0981	0.4556	0.4556	0.1469	4.9835	0.9937
chr15c	0.6128	0.2618	1.4030	1.4030	0.3476	0.3476	0.1926	3.8636	1.4396
chr12a	0.1954	0.4768	0.6413	0.6413	0.1723	0.1723	0.0570	2.2942	0.5492
chr12c	0.1379	0.0753	0.6119	0.6119	0.1579	0.1579	0.0540	1.4830	0.3107
chr12b	0.5791	0.9037	0.7577	0.7577	0	0	0	1.6038	0.1246
chr25a	0.3351	0.5870	1.5664	1.5664	0.5316	0.5316	0.2603	2.7945	1.8983
chr20a	0.4498	0.5392	0.8714	0.8714	0.2619	0.2619	0.2491	2.8887	1.6359
chr20c	0.7879	0.5227	1.9294	1.9294	0.4193	0.4193	0.4122	4.8900	2.2515
chr20b	0.3194	0.4256	1.2524	1.2524	0.1662	0.1662	0.1654	2.5666	1.2167

Table 17: Relative Errors for Esc Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
esc16h	0	0	0.1386	0.1386	0	0	0	0.2590	0.0161
esc16i	0.2857	0	0.2857	0.2857	0	0	0	1.8571	0
esc16j	0	0.25	0.25	0.25	0	0	0	1.5	0
esc32g	0	0	1	1	0	0	0	2.3333	0
esc32e	0	0	2	2	0	0	0	4	0
esc128	0.0938	0.1875	1.9063	1.9063	0	0	0.125	3.6875	2.5
esc16a	0.0588	0.0294	0.2059	0.2059	0	0	0	0.2353	0.0882
esc16b	0	0	0.0068	0.0068	0	0	0	0.0342	0
esc16c	0	0.0125	0.05	0.05	0	0	0	0.3125	0.075
esc16g	0	0	0.7692	0.7692	0	0	0	1.1538	0.1538
esc16f	0	0	0	0	0	0	0	0	0
esc16d	0	0	0.5	0.5	0	0	0	1.125	0.375
esc16e	0	0	0.3571	0.3571	0	0	0	0.8571	0.1429

Table 18: Relative Errors for Esc Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
esc16h	0	0	0.1386	0.1386	0	0	0	0.2590	0.0161
esc16i	0.2857	0	0.2857	0.2857	0	0	0	1.8571	0
esc16j	0	0.25	0.25	0.25	0	0	0	1.5	0
esc32g	0	0	1	1	0	0	0	2.3333	0
esc32e	0	0	2	2	0	0	0	4	0
esc128	0.0938	0.1875	1.9063	1.9063	0	0	0.125	3.6875	2.5
esc16a	0.0588	0.0294	0.2059	0.2059	0	0	0	0.2353	0.0882
esc16b	0	0	0.0068	0.0068	0	0	0	0.0342	0
esc16c	0	0.0125	0.05	0.05	0	0	0	0.3125	0.075
esc16g	0	0	0.7692	0.7692	0	0	0	1.1538	0.1538
esc16f	0	0	0	0	0	0	0	0	0
esc16d	0	0	0.5	0.5	0	0	0	1.125	0.375
esc16e	0	0	0.3571	0.3571	0	0	0	0.8571	0.1429

Table 19: Relative Errors for Had and Kra Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
had20	0.0101	0.0272	0.0546	0.0546	0.0040	0.0040	0	0.0933	0.0364
had18	0.0022	0.0056	0.0489	0.0489	0.0015	0.0015	0.0007	0.0679	0.0362
had14	0.0242	0.0073	0.0140	0.0140	0	0	0	0.0999	0.0286
had16	0.0220	0	0.0333	0.0333	0	0	0	0.1032	0.0473
had12	0.0073	0.0109	0.0387	0.0387	0	0	0	0.0823	0.0266
kra30a	0.0951	0.0610	0.2386	0.2386	0.0534	0.0534	0.0344	0.4333	0.2709
kra30b	0.0505	0.0878	0.3063	0.3063	0.0205	0.0205	0.0194	0.4408	0.2846
kra32	0.0875	0.0633	0.2721	0.2721	0.0515	0.0515	0.0250	0.4325	0.3099

Table 20: Relative Errors for Lipa Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
lipa40a	0.0137	0.0131	0.0251	0.0251	0.0138	0.0138	0.0129	0.0330	0.0282
lipa60b	0.1994	0.1999	0.2491	0.2491	0.1940	0.1940	0.1927	0.2829	0.2679
lipa40b	0.1854	0.1777	0.2309	0.2309	0.1582	0.1582	0.1713	0.2913	0.2621
lipa60a	0.0101	0.0113	0.0188	0.0188	0.0105	0.0105	0.0096	0.0230	0.0224
lipa20a	0.0234	0.0280	0.0399	0.0399	0.0288	0.0288	0.0155	0.0554	0.0453
lipa20b	0.1520	0.1603	0.2071	0.2071	0.1435	0.1435	0	0.2748	0.2212
lipa50b	0.1844	0.1820	0.2371	0.2371	0.1775	0.1775	0	0.2566	0.2567
lipa70a	0.0088	0.0094	0.0160	0.0160	0.0087	0.0087	0.0089	0.0220	0.0195
lipa50a	0.0127	0.0117	0.0220	0.0220	0.0121	0.0121	0.0113	0.0287	0.0242
lipa70b	0.2117	0.2060	0.2499	0.2499	0.2035	0.2035	0.2024	0.2872	0.2774
lipa30b	0.1578	0.1752	0.2197	0.2197	0.1619	0.1619	0	0.2541	0.2343
lipa30a	0.0200	0.0206	0.0304	0.0304	0.0175	0.0175	0.0174	0.0430	0.0345
lipa90a	0.0075	0.0075	0.0132	0.0132	0.0071	0.0071	0.0075	0.0177	0.0156
lipa90b	0.2210	0	0.2652	0.2652	0.2167	0.2167	0.2162	0.2929	0.2846
lipa80b	0.2201	0.2117	0.2626	0.2626	0.2135	0.2135	0.2104	0.2985	0.2803
lipa80a	0.0088	0.0082	0.0145	0.0145	0.0081	0.0081	0.0076	0.0196	0.0181

Table 21: Relative Errors for Nug Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
nug16b	0.0274	0.0548	0.1145	0.1145	0	0	0	0.2177	0.1565
nug16a	0.0509	0.0919	0.1205	0.1205	0.0075	0.0075	0.0075	0.2410	0.1118
nug12	0.0761	0.0692	0.1003	0.1003	0.0173	0.0173	0.0069	0.2422	0.0969
nug15	0.0574	0.0330	0.1183	0.1183	0.0087	0.0087	0.0017	0.2365	0.1183
nug14	0.0414	0.0750	0.0888	0.0888	0.0138	0.0138	0.0079	0.2702	0.1144
nug28	0.0259	0.0286	0.1707	0.1707	0.0190	0.0190	0.0132	0.2331	0.1971
nug17	0.0658	0.0219	0.1062	0.1062	0.0219	0.0219	0.0185	0.2737	0.1328
nug27	0.0489	0.0539	0.1490	0.1490	0.0168	0.0168	0.0210	0.2809	0.1784
nug25	0.0630	0.0224	0.1624	0.1624	0.0144	0.0144	0.0160	0.2756	0.1709
nug18	0.0404	0.0539	0.1264	0.1264	0.0073	0.0073	0.0218	0.1720	0.1244
nug30	0.0389	0.0327	0.1163	0.1163	0.0222	0.0222	0.0114	0.2786	0.1937
nug24	0.0562	0.0356	0.1233	0.1233	0.0258	0.0258	0.0075	0.2259	0.1789
nug20	0.0397	0.0514	0.1362	0.1362	0.0257	0.0257	0.0156	0.2179	0.1385
nug21	0.0402	0.0451	0.1009	0.1009	0.0066	0.0066	0.0074	0.2789	0.1600
nug22	0.0306	0.0373	0.0373	0.0373	0.0245	0.0245	0.0050	0.3137	0.1574

Table 22: Relative Errors for Sko Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
sko56	0.0193	0.0257	0.1060	0.1060	0.0141	0.0141	0.0106	0.1906	0.1703
sko42	0.0527	0.0172	0.1289	0.1289	0.0199	0.0199	0.0190	0.2098	0.1648
sko81	0.0185	0.0247	0.1092	0.1092	0.0109	0.0109	0.0158	0.1686	0.1495
sko90	0.0203	0.0238	0.1024	0.1024	0.0154	0.0154	0.0154	0.1713	0.1473
sko100b	0.0175	0.0223	0.0956	0.0956	0.0175	0.0175	0.0148	0.1515	0.1422
sko49	0.0286	0.0310	0.1247	0.1247	0.0168	0.0168	0.0174	0.1963	0.1580
sko100c	0.0224	0.0318	0.1004	0.1004	0.0159	0.0159	0.0202	0.1647	0.1456
sko100a	0.0116	0.0206	0.0912	0.0912	0.0121	0.0121	0.0169	0.1644	0.1395
sko100d	0.0149	0.0206	0.0983	0.0983	0.0176	0.0176	0.0203	0.1617	0.1411
sko72	0.0229	0.0341	0.1057	0.1057	0.0199	0.0199	0.0177	0.1948	0.1568
sko100e	0.0225	0.0350	0.1007	0.1007	0.0166	0.0166	0.0158	0.1593	0.1482
sko64	0.0207	0.0352	0.1100	0.1100	0.0144	0.0144	0.0168	0.1843	0.1528
sko100f	0.0144	0.0195	0.0922	0.0922	0.0165	0.0165	0.0156	0.1619	0.1371

Table 23: Relative Errors for Miscenalenous Instance Group

Instance	Elshafei Local Search	Local Search	Multi start (10) Adjacent Swap Optimal Neighbour	Multi start (10) Adjacent Swap First Improvement	Multi start (10) Total Swap Optimal Neighbour	Multi start (10) Total Swap First Improvement	GRASP Local Search	GRASP SA	Genetic Algorithm
tho30	0.0507	0.0538	0.1107	0.1107	0.0248	0.0248	0.0192	0.3290	0.2368
rou20	0.0170	0.0196	0.1167	0.1167	0.0203	0.0203	0.0199	0.2101	0.1410
scr20	0.1813	0.1107	0.2683	0.2683	0.0579	0.0579	0.0234	0.7643	0.2818
tho150	0.0310	0.0440	0.1032	0.1032	0.0155	0.0155	0.0174	0.1961	0.1693
tho40	0.0429	0.0558	0.1825	0.1825	0.0188	0.0188	0.0233	0.3350	0.2629
ste36a	0.1667	0.1633	0.4333	0.4333	0.0739	0.0739	0.0372	0.9404	0.6292
ste36b	0.1426	0.2807	0.8808	0.8808	0.1327	0.1327	0.0204	2.8430	1.3800
scr15	0.1091	0.2546	0.2004	0.2004	0.0386	0.0386	0.0480	0.5346	0.2769
ste36c	0.0980	0.1463	0.2952	0.2952	0.0467	0.0467	0.0314	1.0043	0.6017
scr12	0.0234	0.0409	0.1998	0.1998	0	0	0.0151	0.3990	0.0872
els19	0.2382	0.2760	0.5554	0.5554	0.0090	0.0090	0	1.1332	0.4351
wil50	0.0074	0.0239	0.0517	0.0517	0.0109	0.0109	0.0086	0.1135	0.0912
rou15	0.0703	0.0766	0.1256	0.1256	0.0325	0.0325	0.0248	0.2278	0.1354
wil100	0.0119	0.0110	0.0519	0.0519	0.0089	0.0089	0.0077	0.0900	0.0774
rou12	0.0569	0.0720	0.1101	0.1101	0.0111	0.0111	0.0215	0.1901	0.0754

References

[1] Adams, W. P. and Johnson, T. A. (1993). Improved linear programming-based lower bounds for the quadratic assignment proglem. In *Quadratic Assignment and Related Problems.*

[2] Ahuja, R. K., Orlin, J. B., and Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27(10):917–934.

[3] Azarbonyad, H. and Babazadeh, R. (2014). A genetic algorithm for solving quadratic assignment problem (qap). *arXiv preprint arXiv:1405.5050.*

[4] Bazaraa, M. S. and Sherali, H. D. (1982). On the use of exact and heuristic cutting plane methods for the quadratic assignment problem. *The Journal of the Operational Research Society*, 33(11):991–1003.

[5] Burkard, R. E., Cela, E., Pardalos, P. M., and Pitsoulis, L. S. (1998). *The quadratic assignment problem.* Springer.

[6] Burkard, R. E., Karisch, S. E., and Rendl, F. (1997). Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10:391–403.

[7] Chakrapani, J. and Skorin-Kapov, J. (1992). A connectionist approach to the quadratic assignment problem. *Computers & Operations Research*, 19(3):287–295.

[8] Connolly, D. T. (1990). An improved annealing scheme for the qap. *European Journal of Operational Research*, 46(1):93–100.

[9] Danlami, M., Ahmad, F., Mohamad, Z., Hassan, H., and Zakaria, A. (2014). Examination scheduling system based on quadratic assignment.

[10] Elshafei, A. N. (1977). Hospital layout as a quadratic assignment problem. *Journal of the Operational Research Society*, 28(1):167–179.

[11] Frieze, A. and Yadegar, J. (1983). On the quadratic assignment problem. *Discrete Applied Mathematics*, 5(1):89–98.

[12] Gendreau, M., Potvin, J.-Y., et al. (2010). *Handbook of metaheuristics*, volume 2. Springer.

[13] Gómez, E. C., Chaves, R. P., Hurtado, O. G., et al. (2017). Combinatorial optimization np-hard problem solved by using the quadratic assignment problem (qap) solution through a parallel genetic algorithm on gpu. *Visión electrónica*, 11(2):146–151.

[14] Heider, C. H. (1972). A computationally simplified pair exchange algorithm for the quadratic assignment problem. *Center for Naval Analysis*, 101.

[15] Kaku, B. K. and Thompson, G. L. (1986). An exact algorithm for the general quadratic assignment problem. *European Journal of Operational Research*, 23(3):382–390.

[16] Kaufman, L. and Broeckx, F. (1978). An algorithm for the quadratic assignment problem using bender's decomposition. *European Journal of Operational Research*, 2(3):207–211.

[17] Koopmans, T. C. and Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, 25(1):53–76.

[18] Kratica, J., Tošic, D., Filipović, V., and Ljubić, I. (2001). Solving the simple plant location problem by genetic algorithm. *RAIRO-Operations Research*, 35(1):127–142.

[19] Larranaga, P., Kuijpers, C., Murga, R., Inza, I., and Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170.

[20] Lawler, E. L. (1963). The quadratic assignment problem. *Management Science*, 9(4):586–599.

[21] Li, Y. and Pardalos, P. M. (1992). Generating quadratic assignment test problems with known optimal permutations. *Computational Optimization and Applications*, 1:163–184.

[22] Maniezzo, V., Dorigo, M., and Colorni, A. (1995). Algodesk: An experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem. *European Journal of Operational Research*, 81(1):188–204.

[23] Mautor, T. and Roucairol, C. (1994). A new exact algorithm for the solution of quadratic assignment problems. *Discrete Applied Mathematics*, 55(3):281–293.

[24] Oliveira, C. A., Pardalos, P. M., and Resende, M. G. (2004). Grasp with path-relinking for the quadratic assignment problem. In *International Workshop on Experimental and Efficient Algorithms*, pages 356–368. Springer.

[25] Sahni, S. and Gonzalez, T. (1976). P-complete approximation problems. *J. ACM*, 23(3):555–565.

[26] Smith-Miles, K., Baatar, D., Wreford, B., and Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24.

[27] Smith-Miles, K., Christiansen, J., and Muñoz, M. A. (2021). Revisiting where are the hard knapsack problems? via instance space analysis. *Computers & Operations Research*, 128:105184.

[28] Talbi, E.-G. (2009). *Metaheuristics: from design to implementation.* John Wiley & Sons.

[29] Tosun, U. (2022). A new tool for automated transformation of quadratic assignment problem instances to quadratic unconstrained binary optimisation models. *Expert Systems with Applications*, 201:116953.

[30] Wilhelm, M. R. and Ward, T. L. (1987). Solving quadratic assignment problems by 'simulated annealing'. *IIE transactions*, 19(1):107–119.