Parallel Fast Hybrid Genetic Algorithm for Asymmetric Quadratic Assignment Problem

Faruk Mustafić
Mälardalen University Sweden
Högskoleplan 1
Box 883, 721 23 Västerås, Sweden
faruk1337@gmail.com

Ajdin Mujezinović
Mälardalen University Sweden
Högskoleplan 1
Box 883, 721 23 Västerås, Sweden
ajdin.m@gmail.com

ABSTRACT

In this paper we consider the possible solution method for the quadratic assignment problem (QAP). Goal is to find the best possible assignment of the factories to geographical locations so that we minimize the transportation costs given the distances between locations and material flows between factories. This problem has practical importance because not only this situation but also many other business related optimization tasks can be modeled using it.

Since this problem is in the NP-hard class of problems and exact methods for solving it require exponential time, lots of previous work focused on obtaining good enough possibly suboptimal solutions. Some of the heuristics discussed in literature are ant colony algorithms, simulated annealing etc.

In this paper we consider the implementation of the fast hybrid genetic algorithm adapted from other related works but with addition of parallelization. This algorithm sacrifices some of its solution quality in order to be as efficient as possible. We implement the algorithm specifically for the asymmetric QAP case where costs of transportation from location A to location B are not equal to the same cost from location B to location A. After measuring performance with the standardized data sets from QAPLIB we compare our measurements with other state of the art results in the fields and we conclude that it our implementation is comparable to them, though state of the art results still perform better than our algorithm by several criteria.

Keywords

asymmetric quadratic assignment problem, parallel genetic algorithms, heuristics, discrete optimization

1. INTRODUCTION

In this paper we will deal with the Quadratic Assignment problem. One practical interpretation of this problem is that of assigning some set of production facilities to the locations. Each pair of facilities has the transportation cost assigned to it. Similarly location pairs have distances. The goal is to minimize the sum of products of costs and distances [25]. This can be interpreted as amount of metrics of the money required for transportation.

More formally, quadratic assignment problem (QAP in further text) is a combinatorial discrete optimization problem that can be stated as follows. We are given matrices $\mathbf{W} = (W_{i,j})_{n\times n}$ and $\mathbf{D} = (W_{i,j})_{n\times n}$. We call $p \in \Pi$ where $\Pi = \{x : x \text{ is a permutation of numbers 1 to } n\ \}$ a solution. We need to solve

$$argmax_p f = -\sum_{i=1}^{n}\sum_{j=1}^{n} W(i,j)D(p(i),p(j)) \qquad (1)$$

where $D_p(i,j) \equiv D(p(i),p(j))$ and $p(i)$ represents ith element in the permutation p. Note that we decided to deal with maximization so we had to multiply the objective function by -1.

Since distances and flows of goods are generally symmetric properties majority of the research is focused on the symmetric problem variants. Though distances in the real life scenarios can hardly be asymmetrical, the flows can happen to be of that nature. In this case we encounter the asymmetric QAP which is somewhat harder to solve due to not being able to optimize using symmetry relations.

It can be shown that the problem is NP-hard [30]. Thus most efficient exact solution algorithms to date run in the exponential complexity with regards to the problem size. That is why research over the years focused a lot on the useful heuristics that can give optimal or good suboptimal solutions to the problem in the reasonable time.

Some of the previous work on the problem includes notable heuristics that provided good results. Among these are simulated ant colony algorithms (AC) [33], annealing (SA) [4], various tabu search variants (TS) [32], [24] and genetic (GA) and hybrid genetic algorithms (HGA) [7], [9], [18], [21], [23], [34].

We will in this paper present the parallelized version adapting the fast hybrid genetic algorithm (FHGA) proposed by [25]. We will focus our implementation on the special case of asymmetric QAP that can not use the symmetry to optimize the execution time. It also requires a somewhat different set of formulas for efficient objective function evaluation that we derive in this paper. We hope to prove that we can implement a solution finding algorithm with results comparable to the ones of FHGA or other state of the art results in terms of reaching the 1% of the best known value in reasonable time limit.

IRCSE '16 08 November, Mälardalen University, Sweden Kappa or Gamma lecture hall

The rest of the paper is organized as follows. Section 2 contains background information on the genetic algorithms and its hybrid variants required to understand the implementation we provide in this paper. In Section 3 we outline and prove the formulas valid specifically for the asymmetric QAP case. Then the Section 4 lays down the actual algorithm implementation. After that we benchmark the algorithm on the standardized dataset and discuss the results in Section 5. Finally Section 6 contains conclusions and possible guidelines for future work.

2. BACKGROUND & RELATED WORK

Purpose of this section is to describe all notions that are relevant for understanding aims and achievements of the paper. We start with abstract description and use informal terms to introduce general idea of genetic algorithm (GA). We then list all steps that constitute GA, finalizing with detailed report of each step in distinct subsection. After that we proceed to explore the ideas behind the hybrid algorithms that combine local search heuristics with genetic algorithms. This is a type of algorithm which we will adapt in this paper, specifically the Fast Hybrid Genetic Algorithm proposed by [25].

2.1 Genetic algorithms

Genetic algorithms (GA) are probabilistic search algorithm based on mechanics of natural selection and natural genetics [17]. This type of algorithm holds set of possible solutions (population), iteratively improving that set. Improvement is done by selecting best individuals from population and pairing them for a chance for new better individual. Motivation for this approach comes from Holland (first mentioned in [15] and later discussed and expanded in [16], [26] and [11], [5]) and his observation in [5] that living organisms develop their traits via seemingly planless manner of evolution while (computer) scientist spend valuable time and other resources to develop algorithm. GA are probabilistic in a sense that there is no guarantee that solution is optimal. There are various efforts made towards formal proof of GA effectiveness, which started early in [15] (The Schema Theorem) but they have been later criticized and expanded by [1].

Kumar et al [17] summarized GA application that include sensor based robot path planing [38] , image processing [12], gaming [31], real time systems [20], job shop scheduling [20]. Task of combining GA for solving QAP has been undertaken by [23], [25], [8], [36] to name a few. All of them assumed *symmetric* problem (meaning matrix $\mathbf{D}$ is symmetric), allowing certain optimization to be made [32]. Hadley et al. in [14] worked on the issue, but only to provide method for it's symmetrization. Asymmetric data sets are however listed in widely used QAPLIB [6]. Algorithm 1 shows basic steps that constitute execution of every GA. We describe each specific step as follows.

All GAs start with *population generation*. This is processes of creating initial set of feasible solutions [1] P that is a subset of all possible solutions Π. Solutions can be created at random, or some (greedy) algorithmic procedure. Common parameter to this is population size, which fixes cardinality of set P. For each $p \in P$ *fitness function*, which will be used for comparison of individuals (within population), is evaluated. *Crossover* phase consists of two steps. In first, solution pairs (more generally tuples) that will be involved in producing offspring chromosomes (new solutions) are determined. Population is then extended with production of offspring in second phase. Population is again reduced to original population size with application of *selection* operator. With low probability, *mutation* is introduced to some chromosomes. Mutations randomly changes small number of chromosome features [2] (in our case apply extra permutation) in order to introduce new genes that could improve fitness. If new population meets *termination condition* algorithm halts. Otherwise new iteration is performed in a manner already explained.

Algorithm 1 Main loop of the genetic algorithm

```
pop ← init_pop(PS, n)
while !termination_criteria do
  children ← crossover(pop)
  pop ← population_replacement(pop)
  pop ← mutation(pop)
end while
solution ← find_best_individual(pop)
```

2.1.1 Population generation

Population is initialized so that number of individual solutions are created. Number of individuals (population size) should be chosen with respect to specific problem. Some specific problems have reached optimal population size by means of experimental testing [13]. Initialization in most cases happen in stochastic manner and population is then improved with local search mechanism, which will be discussed later in the paper. Greedy heuristics can also be used (for example, GRASP [29]).

2.1.2 Fitness

Fitness calculation simply refers to evaluation of cost function for each individual in population. In most cases, this is real valued function $\phi : P \to R$ calculated for each individual separately. Also in most optimization problem this value is in fact the objective function. Note that since this case is in general computationally most expensive sometimes the approximate yet easier to calculate replacement fitness function is chosen.

2.1.3 Crossover

Selection operator $\sigma : P \to \{0, 1\}$ within crossover step performs selection of individuals which will be included in generation of new solutions. Implementations of this process varies and are again to be chosen with respect to specific problem. Examples of this being roulette wheel selection [2], tournament selection [22], reward based selection [19]. In [3] Banks et al. provide review paper on selection (and other) operators, and Turčinhodžić and Ribić proposed using multiple selections [35] within same algorithm. Finally, crossover operator $\gamma : P \times P \to \Pi$ generates new solutions from the combinations of ones selected by selection operator σ and appends them to P.

2.1.4 Population replacement

[1] Each solution is called chromosome in genetic programming discourse

[2] Feature is also called gene in genetic programming discussion

In this phase some of individuals are killed, that is removed from P, so that original population size is maintained. Let us note the population size with $PS := |P|$. Fitness is often used as indicator who will survive to next generation, but selecting only fittest individuals would leave no room for diversity. Concept of *elitism* exist and refers to strategy where top $x\%$ of population survives. Elitism could lead to population with low diversity that has little chance of converging to solution. This is due to fact that there is no individuals with many different genes that could improve solution thus being stuck in local optima. One of means for keeping population diversity at acceptable level is selection of individuals with low fitness (usually at random and few of them).

2.1.5 Mutation

With low probability, some of solutions are introduced minor and random recombination (mutations). Not only with goes in favor of increasing diversity, but also brings new genes to population, possibly helping it move out local optima.

2.1.6 Termination

Algorithm terminates once it reaches halting conditions. As summarized by [17], ordinary conditions are:

- solution that satisfies minimum criteria is found
- set number of generations have passed
- allocated budget (computational time/money) is met
- successive iterations fail to improve best know solution
- manual inspection
- combination of the above

2.2 Hybrid genetic algorithms

Genetic algorithms do not perform so well on their own in most of the cases [25]. They have the growth of total fitness approximately proportional to the $log(n)$ with respect to the number of generations n. It is easy to see that the GA can fall into the area of really slow growth and might not be able to go much further than that very quickly. This is why they might not be able to explore enough of the search space and are highly dependent on the design of the genetic operators. And even if they are good enough they do not guaranty us the high quality solutions. This is why most of the modern uses of GA combine it with some other sort of the local search (LS) heuristics in order to further improve solutions produced by the GA. Some of the most used heuristics are simulated annealing and tabu search that proved to be good enough to cope with the problem. In this paper we will combine the GA with the variants of tabu search.

2.2.1 Tabu search

Basic problem of the local search heuristics are the local extrema that the procedure can get stuck in. In order to avoid this many techniques have been produced over the years. One of them is so called tabu search. Here we will briefly get around the concepts of the tabu search and more interested reader is referred to [10].

Local search generally can be summed up in the following basic sequence of steps. Let us assume we are in iteration n of the algorithm. We will call the current solution we operate on x_n. The solution in the next generation x_{n+1} is taken from the set $\Omega(x_n) = \{\omega : \text{we can get to } \omega \text{ by one local move from } x_n\}$. Here move can be any mathematical transformation on defined on x_n that produces the next valid solution. Choice of the $x_{n+1} \in \Omega(x_n)$ is dependent on the heuristics.

Most obvious thing to do would be to take the $x_{n+1} \in \Omega(x_n)$ that maximizes the objective value function (in case of the maximization problem). But this exactly lays the ground for getting stuck in local extrema if certain conditions are met. More specifically if $x_n \in \Omega(x_{n+1})$ and x_n is local optima in the extended neighborhood, that is $f(x_n) \geq f(x_i), \forall x_i \in \Omega(x_n) \cup \Omega(x_{n+1})$. We can see that the algorithm will always return to the x_n from $x_{n+1} \in \Omega(x_n)$ since it always has the greatest value in the neighborhood.

Tabu search avoids this type of behavior by maintaining a list of previous moves. It is forbidden to make a move that would return us to the state we already visited. Simplest implementation of this would be to make the reverse moves of recent moves illegal. Tabu search bears its name to this list which is called tabu list, or the list of forbidden solutions. Not all previous moves are kept in the list indefinitely, but list is limited to a certain length and it is crucial to choose this length parameter properly in order to avoid cycles. This rewrites the definition of the neighborhood to $\Omega(x_n) = \{\omega :$ we can get to ω by one local move $m(x_n)$ from $x_n \wedge m \notin T\}$.

Besides this tabu search also has intensification and diversification mechanisms. Intensification is the process of intensified local search in the promising area of the search space. It mostly includes fixing one of the parameters of the solution while perturbing others. Or the variant of gradient ascent (in case of maximization) can be used. Diversification makes sure that all the areas are searched and tries to explore new areas of the space still unexplored by periodically heavily perturbing the current solution or its constituent parts that have not changed for a longer period of time. Diversification should be less frequent than intensification.

2.2.2 Fast hybrid genetic algorithm

In this paper we will parallelize and adapt the variant of the hybrid genetic algorithm proposed by [25]. He builds his GA on top of the robust tabu search (RTS) proposed by [32]. We present here the brief overview of the algorithm but refer the reader to the original paper describing it in more detail.

This algorithm follows the general GA procedure in a way that it maintains a population of size PS in every iteration i. Let us note this with $P_i = \{p_{1,i}, p_{2,i}, \ldots, p_{PS,i}\}$. Each solution $p_{j,i}$ is a permutation of natural numbers $[1, n]$. Initial population is generated at random.

In each iteration of the algorithm the following steps happen. First selection operator as described in Section 2.1.3 $s : P \to \{0, 1\}$ is applied to select the parents for the new children we will add to population. Let n_{cross} denote the number of children produced in this step each iteration. That means we repeat the selection process n_{cross} times. In this case selection is the random operator that returns pair of parents that will produce child and is probabilistic rank based with higher fitness individuals having greater probability of being chosen. Individuals are selected by randomly generating $v \in [1, PS^{1/\sigma}], \sigma \in [1, 2]$ and then calculating the selected individual rank by $u = \lfloor v^{\sigma} \rfloor$ [25].

instance	n	BKV	$\overline{\delta}$, $C_{1\%}$ / C_{bks}			CPU time (s)
			FHGA	PFHGA		
tai20b	20	122455319	**0**	0.011	0/0	0.1
tai25b	25	344355646	**0**	0.022	3/0	0.6
tai30b	30	637117113	**0**	0.015	5/0	1.2
tai35b	35	283315445	**0**	0.005	6/0	2.5
tai40b	40	637250948	**0**	0.026	3/0	4.8
tai50b	50	458821517	**0**	0.004	8/0	18
tai60b	60	608215054	**0**	0.004	8/0	28
tai80b	80	818415043	**0**	0.013	3/0	136
tai100b	100	1185996137	**0**	0.036	10/0	400
tai150b	150	498896643	0.05 3/10	0.047	10/0	2000

Table 1: Comparison of algorithm performance

All the children are improved in the local search procedure that incorporates robust tabu search. Robust tabu search as described in [32] is the tabu search technique specifically crafted to work well with QAP. It takes advantage of the structure of the fitness function evaluation in order to decrease complexity of computing it. There are further improvements of this procedure suggested by [28]. These mostly deal with symmetric matrix cases. The neighborhood of each solution is defined as $\Omega(p) = \{p' : p' = p \oplus m_{ij}\}$ where m_{ij} is the mutation that switches the places of the ith and jth element in the permutation and $p \oplus m_{ij}$ is application of this mutation to the permutation p [25]. So the neighborhood size is n^2 where n is the problem size. Tabu list maintains the last T permutations applied.

This GA is special in a way that it does not have the mutation operator, but the mutations are applied in the local improvement procedure. All the created children are first locally improved. After that the mutation operators are successively in a progressing strength of mutations applied to children and the resulting solutions are again locally improved. Children are replaced by their best mutations if they happen to improve the objective function. Mutations in this case are simple swaps of places since each solution represents the permutation. Number of swaps determine the mutation strength. The more swaps are done the stronger the mutation. Mutations are in range $[\xi_{min}n, \xi_{max}n]$ where $\xi_{min}, \xi_{max} \in [0, 1]$ are the parameters of the algorithm. This specific combination of mutation and robust tabu search is named enhanced tabu search by [25].

Next on as in regular GA the population replacement takes place. It simply selects the best individuals to survive and keeps the population size constant in each iteration. This process is repeated for N_{gen} number of generations or until the satisfactory solution is reached.

Also in order not to lose the diversity of the population, as the algorithm works on very small populations in order to improve its execution time, the restart mechanism is introduced. At each iteration we measure the scaled entropy of the population given by $\bar{H} = -\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{q(i,j)log_2(q(i,j))}{nlog_2(PS)}$. The value $q(i, j)$ represents the percent of value j at the place i. If this measure is under the certain threshold, which is determined experimentally for each problem size, the population undergoes *cold restart*. This means that all the individuals, except the best one in order not to lose the best solution, are changed by applying mutation of strength n, n being the problem size [25].

3. PARALLEL FHGA FOR SOLVING ASYMMETRIC QAP

Now we can describe the actual algorithm we implemented. It is roughly the implementation of the FHGA with additional parallelization and application of the formulas described in Section 3. Originally we implemented algorithm using c++ but here we will only describe the high level implementation without going into too much detail in order for it too be as much as language independent as possible.

3.1 Formula derivation

In this part we will describe the methods and formulas used in order to provide fast implementation of the evaluation of fitness function adapted for the local search heuristics. There is a symmetric case as described in the [32], but we could not find the asymmetric formulation which we derive here.

First optimization due to [32] and also used in other implementations as well, including [25], is that the evaluation time of the $f(p) = -\sum_{i=1}^{n}\sum_{j=1}^{n} W(i,j)D(p(i),p(j))$ of the permutation $p' = p \oplus m_{ab}$ can be reduced if we already calculated the value $f(p)$. In the formula m_{ab} represents the mutation that switches the ath and bth element of the permutation p and the $p \oplus m_{ab}$ represents that this mutation is applied to the particular permutation p. Reduction of calculation time is achieved since it is unnecessary to calculate all the elements of the sum, but only recalculate the ones which are in either switched rows or columns, a and b and then add this Δ to the initial fitness $f(p)$. So in essence $f(p') = f(p) + \Delta$. Exact formula for $\Delta(p, a, b)$ we got for the asymmetric case is as follows.

$$\Delta(p,a,b) = \sum_{j\notin\{a,b\}\wedge i\in\{a,b\}} (W(i,j)D(p(i),p(j)) - W(i,j)D(p(\widetilde{i},p(j))) + \sum_{i\notin\{a,b\}\wedge j\in\{a,b\}} (W(i,j)D(p(i),p(j)) - W(i,j)D(p(i),p(\widetilde{j}))) + \sum_{i\in\{a,b\}}\sum_{j\in\{a,b\}} W(i,j)D(p(i),p(j)) - W(i,j)D(p(\widetilde{i}),p(\widetilde{j}))$$

$$\widetilde{x} := \begin{cases} b & \text{if } x = a \\ a & \text{if } x = b \end{cases} \quad (2)$$

This equation probably can be made more concise by using the symmetries with respect to elements a and b. Not to be confused with the symmetry of the problem as we are working out the asymmetric case. We get the symmetric case if we apply identity $W(i,j) = W(j,i)$ and $D(i,j) = D(j,i)\ \forall i,j$

Another key observation is also by [32] and is well suited for the local search heuristics. We will label neighborhood with $\Omega(p)$ and define neighborhood size as $NS(p) := |\Omega(p)|$. Now if we calculate the matrix $\mathbf{\Delta}(\mathbf{p}) = (\Delta_{ij})_{NS(p)\times NS(p)}$ where the element at the position (i,j) represents the value of $\Delta(p,i,j)$ as per Formula 2 we can now in time complexity $O(NS(p)\times NS(p))$ calculate the best possible neighbor p'. Also in the next iteration we can reduce the computation time for $\mathbf{\Delta}(\mathbf{p}')$ by observing that for each element of $\mathbf{\Delta}(\mathbf{p}')$ we have to recompute only the elements at the intersections of last move made and the move being calculated. This effectively reduces the computational complexity of calculating $\mathbf{\Delta}$ in each iteration from $O(NS(p)\times NS(p))$ to $O(NS(p))$. If in the last iteration we made move m_{ij} then the formula for this calculation is as following and runs in $O(1)$.

$$\Delta(p',u,v) = \Delta(p,u,v) - \sum_{l\in\{u,v\}}\sum_{m\in\{i,j\}}\delta(l,m) + \sum_{l\in\{i,j\}}\sum_{m\in\{u,v\}}\delta(l,m)$$

$$\delta(l,m) = W(l,m)(D(l,m) + D(\widetilde{l},m) - D(l,\widetilde{m}) + D(\widetilde{l},\widetilde{m}))$$

$$\widetilde{x} := \begin{cases} j & \text{if } x = i \\ i & \text{if } x = j \\ v & \text{if } x = u \\ u & \text{if } x = v \end{cases} \quad (3)$$

Again note that the Formula 3 only applies if the $u \notin \{i,j\} \wedge v \notin \{i,j\}$ where m_{ij} is the chosen mutation that gives the best neighbor when applied to p. In other cases we have to compute $\Delta(p,u,v)$ by Formula 2.

3.2 Algorithm

In the Algorithm 2 we can see the main loop of the hybrid genetic algorithm. As explained before the basic idea is to loop until the certain stopping condition is met while maintaining a population of locally optimal individuals by using local improvement procedure.

Let us now examine more closely the parallelized version of the local improvement procedure as shown in Algorithm 3. We can see that for each child that is being locally improved we have a separate thread. This means that the parallel speed up is up to n, where n is number of available processor cores. Since the number of children is generally not too high as we want fast algorithm hardware requirements are not that high.

Algorithm 2 Main loop of the hybrid genetic algorithm

```
pop ← init_pop(PS, n)
improve_locally(pop)
while !termination_criteria do
  children ← crossover(pop)
  improve_locally(children)
  pop ← population_replacement(pop)
  if get_diversity(pop) < THRESHOLD then
    pop ← restart_population(pop)
  end if
end while
solution ← find_best_individual(pop)
```

Algorithm 3 Parallel local improvement procedure

```
Require: children
for i in range(children.size()) do
  Thread.run(improve_single(children[i]))
end for
```

Finally let us examine the procedure for improving the single individual. It is implementation of the improvement procedure described in [25] as described in Section 2.2.2. We give the pseudocode in Algorithm 4. In order to make algorithm more efficient we apply the formula described in Section 3 in the implementation of the `robust_tabu` function. For the aspiration function in `robust_tabu` we can go in the tabu direction if the solution is better than so far best optimal. Also as an intensification mechanism we use gradient ascent method. All of the moves applied during intensification are also added to the tabu list.

Algorithm 4 Local improvement procedure for a single solution

```
Require: c
c ← robust_tabu(c)
for mutation_levelinrange(m_min, m_max) do
  mutant ← c ⊕ mutation(mutation_level)
  mutant ← robust_tabu(mutant)
  if f(mutant) > f(c) then
    c ← mutant
  end if
end for
```

4. NUMERICAL EVALUATION

Here we will present the evaluation of the described approach in practice. This section is split into two parts. The first one which describes exact measuring methodology used for our benchmarks and second presents the obtained results and discusses them and their implications. Methodology contains report on system used for benchmark and data gathered. Second part presents results gathered by benchmark and their analysis.

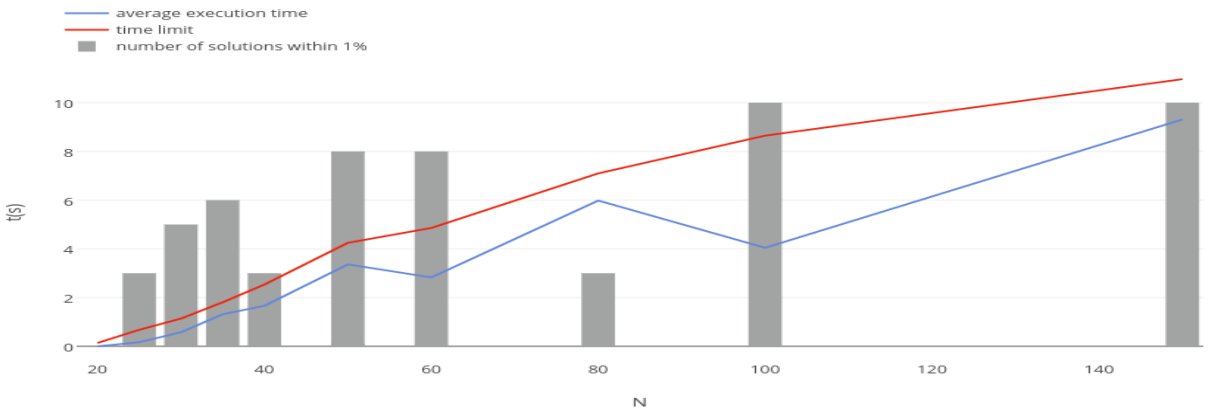

Figure 1: Average execution time

4.1 Methodology

We used the similar measuring method as described in [25]. Idea is that we give each instance of the problem limited run time depending on the problem size. Misevicius determined these times experimentally. Times for each problem size can be seen in the Table 1 in the *CPU time* column corresponding to the problem *taixxb*, where xx tells us the problem size.

Problems are taken from the QAPLIB standard QAP problems of various sizes as described in [6]. The method for generating these problems was proposed in [32] and we primarily used datasets created by these methods. Note that there are two types of problems – a and b. We chose problem type b since it represents the asymmetric case variants.

Proposed solution finding method is stochastic (includes random variables) so for each taixxb dataset we will run algorithm 10 times in order to get statistically significant results. In all runs we record the following: $\overline{\delta}$, $C_{1\%}$, C_{bks}. These denote average deviation from best know value (BKV) ($\overline{\delta} = \overline{z} - z^*$, $\overline{z}$ being the average objective function value over 10 runs and z^* being the best known value), number of solutions that are within 1% of BKV, and C_{bks} number of best solutions found, respectively [25]. Execution time is bounded by upper time limits adopted from [25].

Benchmark was performed on Ubuntu 16.04 LTS machine with Intel Core i7-4510U CPU. `g++` compiler flags in the case of `c++` are `-O3 -pthread`.

4.2 Results and discussion

Results are summarized in the Table 1. We see that proposed algorithm gets within 1% of BKV most of the time, especially for the bigger problem sizes. This indicates that the algorithm scales well with the problem size.

Let us take a bit closer look at this. In Figure 1 we can see the data visualization of the measurements. First note that the graph scale is actually logarithmic meaning that linear function is in fact exponential. The blue line represents the average time that was required to enter the 1% mark of the BKV. Of course we took into consideration only the solutions that actually did make it to this mark. The red line on the other hand is the maximum execution time allowed as for the limits shown in the Table 1. The bars on the graph represent the scaled percentage of the solutions that entered the 1% so that we also can visualize the data regarding the "bad" runs.

Trend we can see is that the blue line increases but with the decreasing slope, further reinforcing that the proposed algorithm scales very well with the problem size. This is also reinforced by the increasing percentages of the solutions making it to the 1% mark. Algorithm seems to get better with the size. One notable exception happens at the problem size of 80, which means that perhaps the problem landscape is a bit harder at this instance or simply the initialization seed gave us bad initial population indicating to the possible sensitivity of the algorithm to the initial conditions. Also the runs that did enter the 1% mark with problem size 80 took unusually longer than the other equivalents which also may be due to the harder problem landscape to tackle. Also algorithm in most of the cases did not use up nearly all the time to enter the 1% zone. Of course, if it entered it at all. Seems that for certain seed algorithm can not fast enough reach the 1% mark.

Finally we should compare our results with the ones obtained by [25]. As we can see in Table 1 FHGA is better than our implementation in every aspect, though our algorithm still provides good results. FHGA gets to some optimal values and enters more of the 1% marks in general. This means that our algorithm needs further speed optimization in order to be more competent with the state of the art results.

5. CONCLUSION

In this paper we managed to implement parallel algorithm based on the FHGA (Fast Hybrid Genetic Algorithm) proposed by the [25] – a variant of the hybrid genetic algorithm designed to run fast for large datasets without losing much of the solution quality. We had to adapt some formulas for the asymmetric case we discussed. Benchmarks were run on standardized datasets from QAPLIB and we obtained good results and have proven that our algorithm implementation scales well with the problem size. Still the state of the art results obtained by [25] indicate that our implementation requires some more optimization.

Besides code optimization, improvements to proposed algorithm could go in direction of improving existing operators. Since mutations could lead out of local optima, operator of mutation could be enhanced or possibly changed in

favor more effective one. Also, greater attention to parallel and distributed approach could be given, possibly using Island model [37] or it's variants as described in [27] for example.

6. REFERENCES

[1] L. Altenberg. The schema theorem and price's theorem. *Foundations of genetic algorithms*, 3:23–49, 1995.

[2] T. Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[3] E. R. Banks, P. Agarwal, M. McBride, and C. Owens. A comparison of selection, recombination, and mutation parameter importance over a set of fifteen optimization tasks. *Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference - GECCO '09*, page 1971, 2009.

[4] A. Bölte and U. W. Thonemann. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research*, 92(2):402–416, 1996.

[5] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial intelligence*, 40(1):235–282, 1989.

[6] R. Burkard, S. Karisch, and F. Rendl. QAPLIB - a Quadratic Assignment Problem Library. *European Journal of Operational Research*, 55(September 1996):115–119, 1991.

[7] Z. Drezner. A New Genetic Algorithm for the Quadratic Assignment Problem. *INFORMS Journal on Computing*, 15(3):320–330, 2003.

[8] Z. Drezner. Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, 33(5):475–480, 2005.

[9] C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. *Quadratic assignment and related problems*, 16:173–187, 1994.

[10] F. Glover and R. Marti. *Tabu Search*, pages 53–69. Springer US, Boston, MA, 2006.

[11] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[12] M. Gong and Y.-H. Yang. Multi-resolution stereo matching using genetic algorithm. In *Stereo and Multi-Baseline Vision, 2001.(SMBV 2001). Proceedings. IEEE Workshop on*, pages 21–29. IEEE, 2001.

[13] S. Gotshall and B. Rylander. Optimal population size and the genetic algorithm. *Proceedings On Genetic And Evolutionary Computation Conference*, pages 1–5, 2000.

[14] S. W. Hadley, F. Rendl, and H. Wolkowicz. Symmetrization of nonsymmetric quadratic assignment problems and the Hoffman-Wielandt inequality. *Linear Algebra and Its Applications*, 167(C):53–64, 1992.

[15] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[16] J. H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–72, 1992.

[17] M. Kumar, M. Husian, N. Upreti, and D. Gupta. Genetic Algorithm: Review and Application. *International Journal of Information Technology and Knowledge Management*, 2(2):451–454, 2010.

[18] M. Lim, Y. Yuan, and S. Omatu. Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. *Computational Optimization and Applications*, 15(3):249–268, 2000.

[19] I. Loshchilov, M. Schoenauer, and M. Sebag. Not all parents are equal for MO-CMA-ES. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6576 LNCS, pages 31–45, 2011.

[20] A. Madureira, C. Ramos, and S. C. Silva. A coordination mechanism for real world scheduling problems using genetic algorithms. In *2002 IEEE World Congress on Computational Intelligence, Hawai (EUA)*, 2002.

[21] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.

[22] B. L. Miller and D. E. Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9(3):193–212, 1995.

[23] A. Misevicius. An improved hybrid genetic algorithm: New results for the quadratic assignment problem. In *Knowledge-Based Systems*, volume 17, pages 65–73, 2004.

[24] A. Misevicius. A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, 30(1):95–111, 2005.

[25] A. Misevicius. A fast hybrid genetic algorithm for the quadratic assignment problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1257–1264, New York, NY, USA, 2006. ACM.

[26] M. Mitchell and H. John. When will a genetic algorithm outperform hill-climbing? *Advances in Neural Information Processing Systems*, 6, 1993.

[27] A. E. Nix and M. D. Vose. Modeling genetic algorithms with markov chains. *Annals of mathematics and artificial intelligence*, 5(1):79–88, 1992.

[28] G. Paul. An efficient implementation of the robust tabu search heuristic for sparse quadratic assignment problems. *European Journal of Operational Research*, 209(3):215–218, 2011.

[29] M. G. C. Resende and C. C. Ribeiro. GRASP: Greedy randomized adaptive search procedures. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Second Edition*, pages 287–312. 2014.

[30] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[31] K. Sandstrom and C. Norstrom. Managing complex temporal requirements in real-time control systems. In *Engineering of Computer-Based Systems, 2002. Proceedings. Ninth Annual IEEE International Conference and Workshop on the*, pages 103–109. IEEE, 2002.

[32] E. Taillard. Robust tabu search for the quadratic assignment problem. *Parallel computing*, 17:443–455, 1991.

[33] E. Taillard. Fant: fast ant system. *Technical Report IDSIA-46-98, Lugano, Switzerland*, 1998.

[34] D. M. Tate and A. E. Smith. A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, 22(1):73–83, 1995.

[35] R. Turcinhodzic and S. Ribic. Using multiple selections to improve the efficiency of the genetic algorithm. In *Information, Communication and Automation Technologies (ICAT), 2015 XXV International Conference on*, pages 1–8, Oct 2015.

[36] M. Vázquez and L. D. Whitley. A hybrid genetic algorithm for the quadratic assignment problem. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pages 135–142. Morgan Kaufmann Publishers Inc., 2000.

[37] D. Whitley, S. Rana, and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–48, 1999.

[38] G. Yasuda and H. Takai. Sensor-based path planning and intelligent steering control of nonholonomic mobile robots. In *Industrial Electronics Society, 2001. IECON'01. The 27th Annual Conference of the IEEE*, volume 1, pages 317–322. IEEE, 2001.