Niche Search: an Application in Vehicle Routing

João Pedro Pedroso

Abstract— **In this paper we describe a hybrid strategy for solving combinatorial optimisation problems, obtained by coupling a local search method to an evolutionary algorithm, and we provide an application to a particular variant of the vehicle routing problem.**

The local search method has been devised specifically for this class of problems. It is based on a composite neighbourhood, which is searched iteratively up to the point where no further improvements can be made.

The evolutionary structure is the niche search, an algorithm based on the evolution of several independent niches. Niches whose individuals' fitness is good remain, and the others tend to be replaced. The separation of the population into niches allows for a good compromise between intensive search (inside each niche) and diversification (through the separation between the niches).

We also describe how we integrate specific problem knowledge into an evolutionary structure, in order to achieve a high performance optimisation algorithm. All the steps that we consider necessary are described in detail: finding an appropriate representation, determining what is a relevant neighbourhood, setting up a local search method and finally integrating the local search into an evolutionary algorithm.

I. Overview

The problem handled in this paper is a vehicle routing variant that has been posed in [1], referred to as the Manhattan Newspaper, or Telegraaf, Problem. It consists of the following: suppose we have a newspaper depot at some location in a city, a set of distributors, and a set of nodes of subscribers where the newspapers should be delivered. The objective is to distribute a newspaper to each of the subscribers, and minimise the time of delivery to the last-served subscriber (or, equivalently, the total distance ran by the distributor who is assigned the longest path).

When two or more solutions have the same objective, the one with the smallest average distribution time is preferred.

Distances between nodes in the city are given by the sum of the vertical distance with the horizontal distance between the nodes (i.e., the city has only vertical and horizontal streets).

These characteristics make this problem different of other vehicle routing problems. In this paper we describe the approach that we have devised for tackling it, which consists on heuristics that combine local search with global search methods. These are intended to, respectively, intensify and diversify the search. Hence, local search routines find a local optimum for a given initial solution, whilst global search supplies the initial solutions where to perform local search. The local search routines have been designed specifically for this problem. They include the definition of an appropriate neighbourhood and a procedure for iteratively exploring it; we describe them in section II. Similar ways for dealing with combinatorial optimisation problems have been described, for example, in [2] and in [3]. Global search is based on the niche search algorithm [4], and is described in section III.

J. P. Pedroso is a researcher at the Riken Institute, Wako-shi, Japan. E-mail: jpp@brain.riken.go.jp.

A. Representation of the solutions

The map of the city is represented by a set of nodes $\mathcal{M} = \{0, 1, \ldots, S\}$, where 0 denotes the depot, and S is the number of subscribers. Each node $n \in \mathcal{M}$ is characterised by its coordinates (x_n, y_n).

The set of distributors is represented by $\mathcal{D} = \{1, \ldots, D\}$, where D is the total number of distributors used.

We represent a solution x of the problem by a set of vectors, $x = \{p_1, \ldots, p_D\}$, where the elements of a given vector p_i are the cities that the distributor i visits, in the order of the visit. The dimension of each of these vectors is $n(i)$, the total number of nodes visited by distributor i (excluding the depot); hence, $p_i = [p_{i_0}, \cdots, p_{i_{n(i)}}]$.

For the purpose of the heuristics discussed in this paper, we have relied exclusively on feasible solutions. A solution is feasible iff all the subscribers (i.e., all the nodes in the map) are visited exactly once by a distributor, and all the distributors start at the depot. More formally, if we consider a problem with S subscribers and D distributors, we define the set of feasible solutions $\mathcal{F}$ as the set of $x = \{p_1, \ldots, p_D\}$ such that:

- $p_{i_0} = 0 \quad \forall i \in \mathcal{D}$
- $p_{i_j} \in \mathcal{M} \setminus \{0\} \quad \forall i \in D,\ j \in \{1, \cdots, n(i)\}$
- $\forall s \neq 0 \in \mathcal{M} \quad \exists!(i,j)\ :\ i \in \mathcal{D},\ j \in \{1, \ldots, n(i)\},\ p_{i_j} = s$

For example, a vector $p_2 = [0, 4, 2, 5]$ in the solution set means that distributor 2 starts at the depot (node 0), and supplies the subscribers at nodes $4, 2$ and 5, in this order. The total time taken by this distributor is hence $t_i = \text{dist}(0,4) + \text{dist}(4,2) + \text{dist}(2,5)$, where the distances $\text{dist}(n,m)$ are the sum of the absolute values of the difference of the coordinates: $\text{dist}(n,m) = |x_n - x_m| + |y_n - y_m|$.

B. Definition of the objective

The objective of this problem is to minimise the time of serving the subscriber which is served the latest. The time of serving the latest of the subscribers in this solution is given by:

$$t(x) = \max_{i \in \mathcal{D}} t_i\ ,\quad t_i = \sum_{j=1}^{n(i)} \text{dist}(p_{i_{j-1}}, p_{i_j})$$

where t_i is the time at which the subscriber $p_{i_{n(i)}}$ is served.

As mentioned above, there is another goal in this problem: to select, from the solutions which lead to the best objective (if there are more than one), the one with the smallest average distribution time. The average time of serving for a solution x is given by

$$a(x) = \sum_{i \in \mathcal{D}} \sum_{j=1}^{n(i)} \frac{\text{dist}(p_{i_{j-1}}, p_{i_j})(n(i) - j + 1)}{S}$$

For the purposes of the heuristic that we implemented, we have relied on a classification of the individuals based on these two goals. If we denote the maximum of the distribution times of a solution x by $t(x)$, and the average distribution time by $a(x)$, solution x_1 is said to be better than x_2 ($x_1 \prec x_2$) iff:

- $t(x_1) < t(x_2)$, or
- $t(x_1) = t(x_2)$ and $a(x_1) < a(x_2)$.

II. Local search

The local search heuristic that we have devised for this problem comprises the search of several types of neighbourhoods, which is performed iteratively until no further improvement is obtained.

The neighbourhoods devised for this problem are divided into two main categories: exchanges of nodes between two distributors, and operations on the path of each of the distributors. In the first category, we consider node pushing from one distributor to another and node exchanges between two distributors. In the second one, we consider 2- and 3-change neighbourhoods, and 2-swap neighbourhood within the path of each of the distributors.

All the neighbourhoods that we consider in this paper are defined on the set $\mathcal{F}$ of feasible solutions.

Node pushing neighbourhood, is defined by pushing nodes from one distributor to another: $N_p(x) = \{y : y \in \mathcal{F}$ and y can be obtained from x as follows: given the paths $p_i, p_j \in x$ of any two distributors i and j of x, remove one node $p_{i_a}, a > 0$ from i and insert it in the path of $j\}$.

Node exchange neighbourhood, allows node exchanges between two distributors i and j: $N_e(x) = \{y : y \in \mathcal{F}$ and y can be obtained from x as follows: given the paths $p_i, p_j \in x$ of any two distributors $i \neq j$ in x, consider the a^{th} node from path i and the b^{th} node from path j, $a, b > 0$; then, swap nodes p_{i_a} with $p_{j_b}\}$.

2-change neighbourhood for this routing problem is an adaptation of the 2-change neighbourhood defined by Lin [5] for the travelling salesman problem. The idea is to operate on the paths of each of the distributors independently, by removing two edges and replacing them with another two (different) edges. It is defined as: $N_2(x) = \{y : y \in \mathcal{F}$ and y can be obtained from x as follows: given a path $p_i \in x$, defining the set of nodes $\mathcal{N}$ visited by a distributor i, remove two edges from this path and replace them with two other edges with both endpoints on $\mathcal{N}\}$.

3-change neighbourhood is an extension of the preceding one ($N_2 \subset N_3$), where 3 arcs are removed and replaced. It also corresponds to an adaptation of the 3-change neighbourhood defined in [5] to this routing problem. $N_3(x) = \{y : y \in \mathcal{F}$ and y can be obtained from x as follows: given a path $p_i \in x$, defining the set of nodes $\mathcal{N}$ visited by a distributor i, remove three edges from this path and replace them with three other edges with both endpoints on $\mathcal{N}\}$.

2-swap neighbourhood operates on the paths of each of the distributors independently, by exchanging the position of 2 nodes of the path: $N_s(x) = \{y : y \in \mathcal{F}$ and y can be obtained from x as follows: given a path $p_i \in x$, defining the path of a distributor i, swap the node at position $a > 0$, p_{i_a} with the node at position $b > 0$, $p_{i_b}\}$.

A. Iterating

Local search is performed by combining the neighbourhoods described above. Given a starting (feasible) solution, each neighbouring region is explored, all the improving solutions being accepted. Search proceeds by iterating through these neighbourhoods, and repeating until no further improvement is achieved.

Improvement in this context means that we can find a solution y in some neighbourhood of the current solution, x, such that $y \prec x$ (and hence y replaces x).

As there are multiple possibilities of combining the search on each of these neighbourhoods, we had to determine a strategy which would, from one side, provide as good as possible local optima, and from the other side be parsimonious in what concerns the computational burden.

We have made some preliminary tests using random-start local search, and more specific tests at the time of their integration in the evolutionary algorithm. The complete local search strategy that appeared to perform best, starting with purely random feasible solutions, is the following:

```
get a feasible solution $x_0$
Procedure Local_Search($x_0$)
$t = 0$
do                                              Start the iterative procedure.
$t = t + 1$
$x_t = x_{t-1}$
$\forall y \in N_{p(x_t)}$ if $(y \prec x_t)$ $x_t := y$      Search all neighbourhoods in a given order.
$\forall y \in N_{e(x_t)}$ if $(y \prec x_t)$ $x_t := y$
$\forall y \in N_{s(x_t)}$ if $(y \prec x_t)$ $x_t := y$
$\forall y \in N_{3(x_t)}$ if $(y \prec x_t)$ $x_t := y$
while $(x_t \prec x_{t-1})$
return $x_t$
end procedure
```

Note that there are two possibilities for updating the best solution when searching in a particular neighbourhood. The first one, called *best-updating*, consists of searching the best solution y^* in the entire neighbourhood. If it is better than our current solution x, then replace x by y^*. The second one, called *better-updating*, consists of replacing x during the local search whenever the current neighbour generated y_i is better than x. In this case, the subsequent "neighbour" y_{i+1} is obtained from the new solution x, and hence does not belong to the initial neighbourhood. Better-updating is used in our implementation because it generally provides superior results, as the number of solutions "tried" in each local search is larger. The notation used above, in the procedure Local_Search, is therefore slightly misleading.

III. Niche search

Evolutionary algorithms function by maintaining a set of solutions, generally called a *population*, and making these solutions evolve through operations that mimic the natural evolution: reproduction and selection of the fittest. Some of these operators where customised for the concrete class of problems that we are dealing with in this paper; we focus on each of them in following sections.

Niche search is an evolutionary algorithm where the total population is grouped into separate niches, which evolve independently for some generations. The claim is that this way several more localised searches are done at the same time, inside each of the niches; we hence expect to keep a good compromise between intensification of the search and diversification of the population. This method has some similarities with the one described in [6], where *competing subpopulations* play a role similar to niches in our algorithm.

Niches are subject to competition between them. The bad niches (i.e., those which have worse populations) tend to extinguish: they are replaced by new ones, which are formed by elements selected from a "good" niche and the extinguishing one.

The representation of a solution in the evolutionary algorithm is the same used for the local search methods, described in section I-A; this differs from most of the genetic algorithms, where normally bitstrings are used (see e.g. [7]).

A. Mutation

The mutation operator that we have devised for this problem consists on selecting a subpath inside the complete path of one of the distributors, removing it, and inserting it into the path of another distributor. This way we expect that after mutation many of the (probably good) subpaths of the original genome will be kept in the mutant.

In niche search there are two parameters controlling mutation: intensity and probability of mutation. The probability of occurrence of mutation determines if it actually occurs or not; the intensity of the mutation determines the importance of the changes induced by this operator, i.e., the size of the subpath (the number of its nodes) that is to be removed from one distributor's path, and inserted in another one's.

Suppose for example that we have an instance with 10 nodes and two distributors. Our solution could be:

[0 1 2 3 4 5] [0 6 7 8 9 10]

If the subpath for mutation is [7 8 9], one solution that could potentially be obtained is

[0 1 2 3 **7 8 9** 4 5] [0 6 10]

B. Crossover

In evolutionary algorithms crossover always means to operate on two solutions of a given problem (the parents) to produce a third one (their descendent).

One of the philosophical ideas motivating the crossover operation is that when two solutions are very similar, the offspring resulting from crossover between them should also resemble them. In particular, two identical solutions should be able to produce a single offspring, identical to them. The aim is to be able to somehow make the search region more concentrated in "good" subregions, as the evolutionary process goes on.

We have devised a specific version of this operator to this routing problem. It consists of the following: take some subpaths of one of the solutions; then remove all the nodes in these subpaths from the other solution; finally, randomly insert all the subpaths in the second solution, producing another feasible solution. The aim is to keep many of the subpaths of the parents unchanged in the offspring.

More concretely, what we do is to select a subpath in one of the parents and insert it on the other parent in such a way that the arc connecting the subpath to that solution is kept. Suppose for example that the subpath to insert in a given solution is [1 2 3]; then, we start searching the arc ending in node 1 in that solution. Admitting that this is the arc [. . . 5 1 . . .], the offspring produced would have this path changed to [. . . 5 1 2 3 . . .].

As another example, consider the 10-node 2-distributor instance again. If we are given the solutions

$$x_1 = [0\ 1\ 2\ 3\ 4\ 5]\ [0\ 6\ 7\ 8\ 9\ 10]$$
$$x_2 = [0\ 1\ 3\ 5\ 7\ 9]\ [0\ 2\ 4\ 6\ 8\ 10]$$

one possible subpath to choose at the crossover could be [2 3 4] from solution x_1. We search for the arc ending in node 2 in the solution x_2, which is the arc [0 2 . . .] from the second distributor. We then remove the nodes 2, 3 and 4 from x_2, obtaining

$$x_2 = [0\ 1\ 5\ 7\ 9]\ [0\ 6\ 8\ 10]$$

Now we are ready to insert the subpath, obtaining:

$$x_2 = [0\ 1\ 5\ 7\ 9]\ [0\ \mathbf{2}\ \mathbf{3}\ \mathbf{4}\ 6\ 8\ 10]$$

As with mutation, niche search has two parameters controlling the crossover: one determines the probability of occurrence, and the other sets the intensity of this operation. The intensity determines the number of crossovers to perform and the size of the subpaths for each of them.

C. Local search

In our implementation we have decided to always bind a (probably non locally-optimal) new solution obtained by a genetic operation (crossover and mutation) into a local optimum. This means that a local search is performed every time a new individual is generated by the genetic part of the algorithm. The procedure for the generation of a new element is, hence:

```
select parents ($p_1$, $p_2$)
Procedure Reproduce($p_1, p_2$)
create son $s$ := crossover($p_1$, $p_2$)      Start with crossover (section III-B).
$s'$ := mutation($s$)                          Mutate the new solution (section III-A).
$x$ := Local_Search($s'$)        We finish the generation of the new element
           performing a local search procedure, starting at solution $s'$ (section II).
return $x$
end procedure
```

D. Selection in each niche: rank-based fitnesses

As explained in section I-B, there are two goals to achieve in this problem: firstly, try to achieve an objective as good as possible; then, if the solution is degenerated, choose the one with the smallest average distribution time. This motivates to have the selection of the individuals that are able to reproduce at each generation based on their ranking, according to the two goals described on that section.

In niche search there is a parameter of each niche, called the *selectivity*, which controls the probability of selection of each individual in relation to their competitors. If this parameter is very low, then the probability of selection of the best individuals is only slightly greater than the probability of selection of the worst; if it is high, then the best individuals have a much greater probability of selection, what means that the "genetic information" of the worse ones is not likely to propagate to the future generations.

The way we handle this issue with the Manhattan problem is the following: we give a fitness for each individual based on its ranking. In a population of n elements, the best is assigned a fitness of 1 (i.e., n/n), the second-best $(n-1)/n$, up to the worse, whose fitness is $1/n$. We then elevate this value to a power, greater or equal to zero, which is the selectivity parameter of the niche[1], to obtain the scaled fitness of each individual.

The selection is then performed through roulette wheel selection, giving to each individual a probability of selection proportional to its scaled fitness (see, for example, [8] for a description of roulette wheel selection).

E. Elitism

Elitism determines whether the best solution found so far by the algorithm is kept in the population or not. As mentioned before, niche search keeps several groups, or niches, evolving with some independence. Each of these groups may be elitist (keeping *its* best element in its population) or not. Elitism generally intensifies the search in the region of the best solution.

Our objectives are two fold: we want the search to be as deep as possible around good regions, but we do not want to neglect other possible regions. The strategy that we devised for accomplishing this is the following: niches whose best individual is different of the best individual of other niches are elitist. When several niches have an identical best individual (and this occurs frequently), only one of them is elitist. With this strategy we hope to have an intensified search on regions with good solutions, and at the same time enforce some degree of diversification.

F. Niche search core algorithm

We summarise now the main steps of the functioning of the niche search algorithm. This is the kernel algorithm, which drives the population operations making use of the solution representation and genetic operators described in the preceding sections. As we said before, niche search is characterised by evolution in two layers: in the higher layer, there is the evolution of niches, subject to competition between them. Each iteration of this process is called a *niche generation*, or simply a generation . In the lower layer, the individuals that compose each niche evolve inside it, competing with other individuals of the niche. Each iteration of this lower layer process is called an *individual's generation*, or a subgeneration.

The code describing the evolution of the set of niches, in what we call a niche generation, is presented below.

```
set t := 0                                    Start with an initial time.
niches(t) = CreateNiches(t)                   Create the desired number of niches for the
                                              run.
InitParameters(niches(t))                     Randomly initialise the parameters that
    characterise each niche: crossover probability and intensity, mutation probability
                                              and intensity, etc.
InitialisePopulation(niches(t))               Randomly initialise the population of each
                                              niche.
Evaluate(niches(t))                           Evaluate the fitness of all the niches in the initial
    population. For evaluating a niche, we used the fitness of its best element (other
                                              strategies are also possible).
iterate                                       Start evolution.
Breed(niches(t))    Create a new generation of individuals in each of the niches,
                    through the lower layer evolution process described below.
Evaluate(niches(t))                           Evaluate the new niches.
weak(t) := SelectWeak(niches(t))              Select the niches that will extinguish.
strong(t) := SelectStrong(niches(t))          Select the niches that will be used
                                              for generating new niches.
newniches(t) := Recombination(weak(t),strong(t))   Create a new niche
    for replacing each of the extinguishing ones. The recombination strategy used is
    to create a population formed of the union of the weak niche with a strong one.
    Then, replace the individuals of the weak niche by a selection of the best
                                              individuals from that population.
InitParameters(newniches(t))                  Assign random parameters to created niches.
Evaluate(newniches(t))                        Evaluate the new niches.
Extinguish(weak(t), niches(t))                Remove the weak niches from the
                                              population
Insert(newniches(t), niches(t))               and include the newly created ones.
niches(t+1) := niches(t)
t := t + 1                                    Increase the time counter.
until Terminated()        Termination criteria: number of generations completed.
display solution                              Solution is the best individual found.
```

Notice that all the parameters that characterise each niche (selectivity, mutation intensity and probability, etc.) are determined exogenously and randomly. The (random) values of the mutation intensity and of the crossover intensity are multiplied by a value, which linearly decreases with the generations passed, being 1 at the beginning and 0 at the end; the selectivity is multiplied by a value, which linearly increases with the generations past, being 0 at the beginning and 1 at the end. The aim of this is to force the population to be more and more homogeneous, as the number of generations increases (and solutions are hopefully closer to the optimum).

We now turn to the evolution of the individuals inside each of the niches. Pseudo-programming code describing how individuals breed at each generation of the niche evolution (i.e., describing what a *subgeneration* is) is presented here. Notice that this process is repeated for each of the niches, at each niche generation.

```
Procedure Breed(niches(t))
for all niche in niches(t) do               (t is the niche generation counter).
g := 0                                      Initialise the "subgeneration" counter.
```

[1]This parameter may change with the phase of evolution; generally, it is low at the beginning of the evolutionary process and high at the end, thus increasing the selectivity stress with time.

```
population(g) := niche        Set the reference population: (only) the elements of
                              the niche that is now breeding.
iterate                       Start evolution.
for all element in offspring(g) do
$p_1$ = Selection(population(g))     Select parents for reproduction
$p_2$ = Selection(population(g))     (in our implementation through roulette
                                     wheel selection).
element := Reproduce($p_1, p_2$)     Create the offspring using the operators
done                                 described in (section III-C).
Evaluate(offspring(g))   Evaluate the objective of all the individuals in the
                         niche's population. Scale to obtain the fitnesses (section III-D).
population(g+1) := offspring(g)      Future population is the offspring.
g := g + 1                           Increase the subgeneration counter.
until Terminated()   Termination criteria: maximum subgenerations achieved,
                     or best individual of current population is not better than that of last
                     subgeneration's (and minimum subgenerations are not achieved yet).
niche := population(g)   Update niche's population. This niche is now ready
done                     to start competition with the others.
end procedure
```

IV. Numerical results

A. The problem instance

One instance of this problem has been defined in [1]. In this instance, we are given the coordinates of 120 subscribers of a newspaper, located in the city of Manhattan, and the coordinates of the depot. The aim is to allocate the nodes to 4 newspaper distributors, for optimising the objective of the problem and the subsequent goal.

B. Random-start local search

The initial solutions for random-start local search were obtained as follows: randomly choose one of the nodes to visit on the map (from those which are not yet assigned), and randomly assign it to a distributor.

We have made some experiments with the several combinations of searching the neighbourhoods defined in section II. We also tried a *random* composite neighbourhood, where the order of searching each of the neighbourhoods is randomly determined at each iteration. The motivation for implementing it was to provide a more robust composite composite local search method, but it turned out that the results obtained were considerably worse than any of the other combinations. This result is quite surprising, as one could imagine that a random choice of the neighbourhood would widen the composite neighbourhood.

The composite neighbourhood which provided better results was $N_p \to N_e \to N_s \to N_3$; we have hence decided to include it in the niche search.

C. Niche search

For the purpose of comparing niche search with local search, we have divided the results into two series: one in which niche search performs the same amount of local searches that were performed in the random-start tests, and another where the computational time is identical. Notice that local search tends to take much less time inside niche search, because the number of iterations required to "stabilise" the solution (i.e., obtain no further improvements through local search) is smaller. The reason for this is that, as we often start the local search from a good solution, it is easier to reach the point where it produces no further improvement.

Previous experiments with niche search have shown that small populations and a small number of niches tend to provide a good compromise between robustness and computational requirements.The number of niches and the population of each niche that we used for obtaining the results described in this section are, hence, relatively small. For an increased reliability, a larger number of these should be adopted (especially a larger number of niches, as this would strongly diminish the probability of getting stuck in a local optimum).

Identical number of local searches For this series of runs, we have tuned the algorithm's parameters in such a way that the number of local searches (i.e., the total number of individuals generated) is the same used for testing local search (section IV-B).

These results show a clear improvement over random-started local search. The average solution found by niche search, 1223.4, is much superior to the one found by pure local search (1271.3), the improvement being about 3.9%.

For performing about 2500 local searches inside niche search, we have used 5 niches, each composed of 3 individuals (hence a total population of 15 individuals), which we made evolve for 25 generations, inside which each niche could produce from 5 subgenerations (if no improvement is made after the 5th subgeneration) to 10 (if all subgenerations lead to improvements).

Identical computational time For this series of runs, we have tuned the algorithm's parameters in such a way that the computational time required is identical to the one that was used in the series of random start local search (this implies that the number of local searches performed in niche search is greater than those performed in random start local search).

TABLE I
Summary of results obtained for 10 independent runs.

Algorithm	Distribution time $t(x^*)$		Mean # local searches	Mean CPU time (s)
Random start local search	best	1247	2500	4365
	mean	1271		
	worst	1297		
Niche search (same # local searches)	best	1205	2484	1374
	mean	1223		
	worst	1246		
Niche search (same CPU time)	best	1191	7730	3998
	mean	1208		
	worst	1225		

Results obtained here provide a further improvement of about 1.2% over the previous section. We arrive to an improvement over random started local search of about 5.2%, for a slightly smaller computational time. These results show a clear interest in using niche search as a mechanism for controlling the start solution of local search.

For the results described in this section, we have used 5 niches, each composed of 3 individuals (hence a total population of 15 individuals). Niches evolved for 75 generations, each having from 5 to 10 subgenerations.

We finish by showing the circuit obtained for the best solution found in these runs, in figure 1.

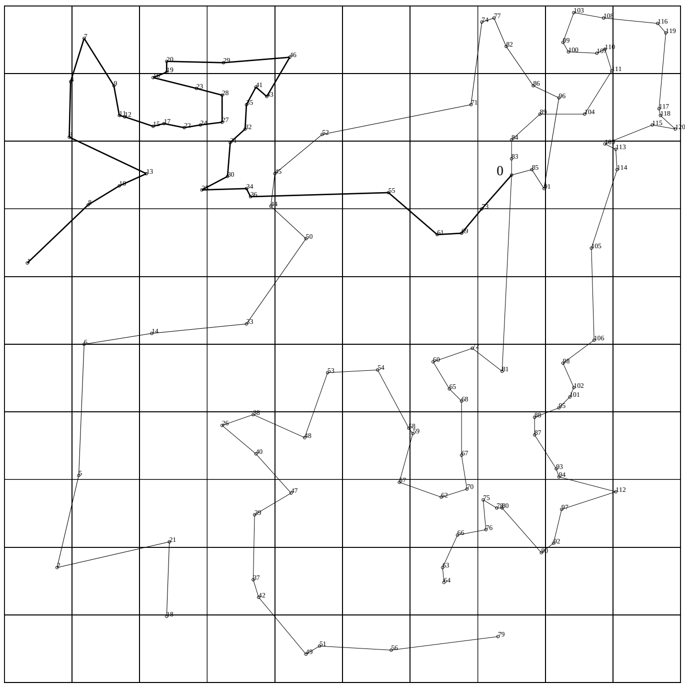

Fig. 1. Best solution found in the runs reported: overview of the paths followed by each of the distributors. Depot is at node 0.

V. Conclusion

In this paper we describe a hybrid strategy for solving combinatorial optimisation problems which is obtained combining an evolutionary algorithm with local search methods. We apply this strategy to tackle the Manhattan problem. Although this strategy does not provide any results in terms of the closeness to the optimum of the problem we deal with (which, to the best of our knowledge, at the present time is unknown for the specific instance that was treated), it does provide interesting results in terms of achieving good feasible solutions (upper bounds). A combination of this strategy with another working on the lower bound would be of great value, and is certainly an attractive direction for future research.

The results obtained by the hybrid strategy show a clear improvement of the combination of evolutionary approaches with local search, which provide a mix of intensification and diversification procedures in the same algorithm. Improvements of the hybrid strategy over random start local search provide a measure of the performance of the niche search, which may be used for comparison with other evolutionary approaches.

The elitist mode implemented proved to be an efficient diversification mechanism: we observed that when the best niches propagated, many times the best individual in several niches would be the same. But, as between all the niches with the same best individual only one could be elitist, the best element of the others would soon change, and very often lead to improvements afterwards.

The roulette wheel selection based on a measure of the ranking was also an important point, as it allowed for considering the two goals of the problem in selection. It was also important in coping with the sometimes dramatic differences that small changes in the structure of the solution imply in terms of the objective, as well as with the fact that often different solutions lead to the same objective value.

There are several things that can be done in order to improve this heuristic, both in the local search strategies and in the niche search. On the side of the local search, we believe that the modification that could probably bring better improvements might be increasing the number of nodes that distributors can exchange between them; i.e., distributors may be able to exchange subpaths between them, instead of only exchanging nodes. On the side of the niche search, there are two modifications that we believe may be worthy. The first is to allow different niches to run different local search methods; for example, each niche might run a different combination of the exploration of the neighbourhoods defined in section II. Another modification, which is somehow related to this one, is to "remunerate" each niche in terms of the *improvement* that it makes on the solution, instead on doing it in terms of the fitness of its individuals.

References

[1] CMG, De Telegraaf, and Technische Universiteit Eindhoven, "Verdien f5.000,- met een kantenwijk in Manhattan", (Contest information sheet), 1996.

[2] Charles Fleurent and Jacques A. Ferland, "Genetic hybrids for the quadratic assignment problem", in *Quadratic Assignment and Related Problems*, vol. 16 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1994.

[3] Bernd Freisleben and Peter Merz, "New genetic local search operators for the traveling salesman problem", In Voigt et al. [12].

[4] João P. Pedroso, "Niche search: an evolutionary algorithm for global optimisation", In Voigt et al. [12].

[5] S. Lin, "Computer solutions to the traveling salesman problem", *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[6] Dirk Shlierkamp-Voosen and Heiz Mühlenbein, "Strategy adaptation by competing subpopulations", in *Parallel Problem Solving from Nature - PPSN III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., 1994, number 866 in Lecture Notes in Computer Science.

[7] Sam R. Thangiah, *Practical Handbook of Genetic Algorithms: New Frontiers*, chapter Vehicle Routing with Time Windows Using Genetic Algorithms, Lewis Publishers, 1995.

[8] David E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.

[9] João P. Pedroso, "Niche search: an application to the manhattan newspaper problem", Discussion Paper 9765, Centre for Operations Research and Econometrics, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1997.

[10] Christos H. Papadimitriou and Kenneth Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall Inc., 1982.

[11] Bruce L. Golden, Gilbert Laporte, and Éric D. Taillard, "An adaptive memory heuristic for a class of vehicle routing problems with minmax objective", *Computers & Operations Research*, pp. 445–452, 1997.

[12] Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, Eds., *Parallel Problem Solving from Nature - PPSN IV*, vol. 1141 of *Lecture Notes in Computer Science*, Berlin, Germany, 1996. Springer.