A Memetic Immunological Algorithm for Resource Allocation Problem

Jole Costanza, Vincenzo Cutello, and Mario Pavone

Department of Mathematics and Computer Science
University of Catania
V.le A. Doria 6 – 95125 Catania, Italy
{costanza,cutello,mpavone}@dmi.unict.it

Abstract. In this research work, we present a combination of a memetic algorithm and an immunological algorithm that we call Memetic Immunological Algorithm – *MIA*. This algorithm has been designed to tackle the *resource allocation problem on a communication network*. The aim of the problem is to supply all resource requested on a communication network with minimal costs and using a fixed number of providers, everyone with a limited resource quantity to be supplied. The scheduling of several resource allocations is a classical combinatorial problem that finds many applications in real-world problems. *MIA* incorporates two deterministic approaches: (1) a local search operator, which is based on the exploration of the neighbourhood; and (2) a deterministic approach for the assignment scheme based on the *Depth First Search* (DFS) algorithm. The results show that the usage of a local search procedure and mainly the DFS algorithm is an effective and efficient approach to better exploring the complex search space of the problem. To evaluate the performances of *MIA* we have used 28 different instances. The obtained results suggest that *MIA* is an effective optimization algorithm in terms of the quality of the solution produced and of the computational effort.

Keywords: Immunological algorithms, memetic algorithms, combinatorial optimization, scheduling resources allocation problem, resource allocation problem, scheduling problems.

1 Introduction

In this work we present an immunological algorithm based on a deterministic approach that involves the *Depth First Search* (DFS) algorithm, and ad-hoc local search strategy to tackle a combinatorial optimization task, the *Resource Allocation* problem [12,2,13], whose main goal is to satisfy the resource allocation requests from several items with minimal efforts. Any request can be satisfied by only one provider, which however has limited resources. Resource allocation problem on a communication network is a classical combinatorial optimization problem, which finds many applications in real-world problems, such as fuel distribution problem, drugs distribution in hospitals, postal services distribution, and distribution networks in general.

In many NP-hard problems, the goal is *to pack* items into a set of *containers*, without exceeding the capacities of each individual container. These kinds of problems are simply refereed as *multi-container packing problem* (MCPP) [15], and a typical example is the classical *bin packing problem* (BPP). In general, for these kinds of problems, two types of containers are considered [11]: (1) with *capacity*, i.e. the sum of the weights of the items cannot exceed a given capacity; (2) with *quota*, i.e. the sum of the weights of the items must be at least as large as the quota. Resource allocation problem asks to satisfy with minimal costs all resource allocation requests received from several items using a given fixed number of providers, each one based on a limited *capacity*, i.e. maximal quantity to be supplied. How to schedule the resource distribution depends on several aspects: how many quantities have been required in the overall; how many providers are available; how much is the capacity of each available provider; the traffic on communication network; and many other conditions that influence either communication network or the quantity of available resource. Moreover, the problem is also subject to several constrains: any provider is able to supply a limited quantity of resource; the number of providers is given and fixed; the sum of resource quantity of a subset of items satisfied by a provider cannot exceed its own capacity. Thus, to face this problem is required optimizing several objectives, which are subject to various constraints: maximize the number of resource satisfied; maximize the number of items served; minimize the path from one item to other in the network; minimize the costs on each path in the communication network. Resource allocation on a communication network can be seen as a problem very similar to the *Vehicle Routing Problem*(VRP), and primarily with its variant called *Capacitated Vehicle Routing Problem* (CVRP) [17]. VRP represents the class of problems in which a fixed number of vehicles must visit a set of customers, or cities, through a set of routes. The goal of VRP is to satisfy any customer request, everyone with a known demand, minimizing the costs on the routes, and respecting some constraints, such as: (i) each customer can be visited once and at most by one vehicle, and (ii) each vehicle starts, and ends to the depot. CVRP is the more studied member of the family, where capacity restrictions are imposed for the vehicles.

2 The Problem

The problem of resource allocation on a communication network can be formulated as follows: let be $G = (V, E)$ a graph, where the node $v \in V$ represents an item, and the set E represents the paths on the network. We assume that $|V| = n$ and $|E| = m$. For any node v is assigned a weight $q(v) \geq 0 \quad (q : V \rightarrow \mathbb{R})$, which indicates the resource required by the item v; also for any edge $e \in E$ is assigned a weight $c(e) > 0 \quad (c : E \rightarrow \mathbb{R})$, which represents the cost on the segment e. Given a set of providers, $R = \{1, \ldots, h\}$, such that $|R| < |V|$. To each provider $r \in R$ is assigned a limited resource capacity, i.e. a maximal resource quantity that can be supplied by r: $b(r) > 0$ with $b : R \rightarrow \mathbb{R}$. We note that the sum of all weights of items is greater than the maximal capacity allowed, $\left(\sum_{v\in V} q(v) > max_{r\in R}\{b(r)\}\right)$, and that $min_{v\in V}\{q(v)\} \leq min_{r\in R}\{b(r)\} \leq max_{v\in V}\{q(v)\} \leq max_{r\in R}\{b(r)\}$.

The goal of this combinatorial optimization task is to assign h providers over n items such that: (1) the number of satisfied items is maximal, i.e. all resource allocated required have been supplied; (2) each provider must supply resources to all those items that are as topologically near as possible in the communication network; finally (3) the sum of the weights of items supplied by provider r must be not greater than the capacity of r itself. This definition is equivalent to partitioning the set V in h subsets, such that their union is V, and their intersection is the empty set. A proper way to face this problem could be to tackle it as a multi-objective problem; however, in this research work we have tackled the problem using a *single-objective function*:

$$f(\boldsymbol{x}) = \frac{C_{tot}(\boldsymbol{x})}{\sum_{r\in R}|V_r|} \times \left[1 + \left(|V| - \sum_{r\in R}|V_r|\right)^{\beta}\right] \tag{1}$$

where V_r is the set of all items that have been supplied by the provider r; β is a constant, and represents a penalty factor that gives priority to solutions able to satisfy all resources required; and $C_{tot}(\boldsymbol{x})$ is the total cost produced by the given solution $\boldsymbol{x}$ on the communication network $G = (V, E)$, and it is given by

$$C_{tot}(\boldsymbol{x}) = \sum_{r=1}^{|R|}\left(\sum_{j=1}^{|V_r|} q(x_{rj}) + \sum_{j=1}^{|E_r|} c(e_{rj})\right)$$

where E_r is the subset of all edges visited by the provider r, and e_{rj} is the edge connecting $(x_{r(j-i)}, x_{rj})$.

To understand what are the parameters that affect the output, we performed the *Morris method* [14] on a graph with 256 vertices, and 32640 edges (*queen16_16.col* – see section 4). The Morris method is a sensitivity analysis useful to understand the effects of a parameter with respect to all others, which vary simultaneously. Figure 1 shows the sensitivity analysis carried out for our objective function (equation 1). Inspecting this figure is possible to see how the vertices {36, 115, 139, 152, 172, 234} seem to be the most important, since they affect more on the objective function than the remaining vertices, whereas nodes 118, 101, and 182 are the less influential ones.

3 The Memetic Immunological Algorithm

MIA is based on clonal selection principle whose own main features are cloning, hypermutation, and aging operators. As in the classical clonal selection algorithms, the antigen (Ag) represents the problem to tackle, that is the communication network $G = (V, E)$, whilst the B cells are a population of candidate

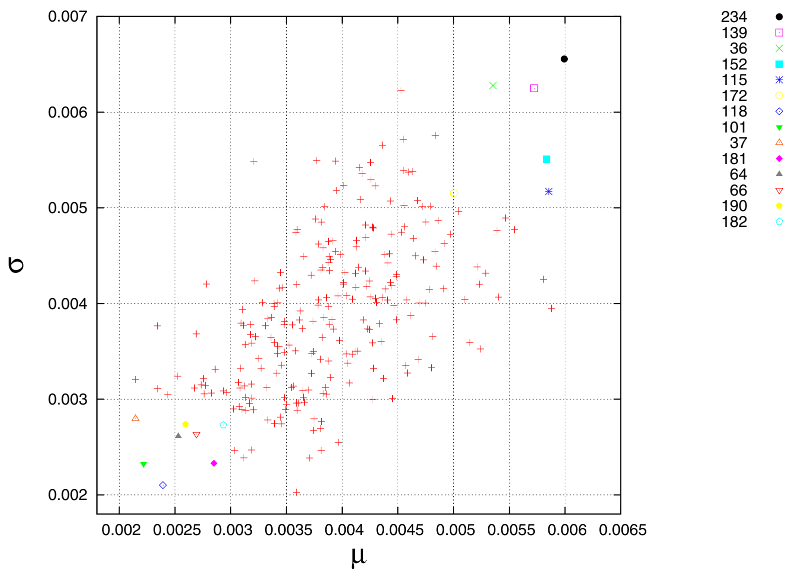

Fig. 1. Normalized μ and σ of Sensitivity analysis using the Morris method. A high value of μ indicates a parameter with an important overall influence on the output. A high value of σ indicates a parameter involved in interaction with other parameters or whose effect is nonlinear.

solutions, defined hence as strings of integers (vertices). B cells are represented as a permutation of vertices from which the algorithm starts the visiting process in order to assign the items to providers. With $P^{(t)}$ we denote a population of d individuals of length $\ell = |V|$. The first population is created randomly by a uniform distribution, which represents a subset of the space of solutions.

MIA incorporates the *static cloning operator*, that clones all B cells *dup* times producing an intermediate population $P^{(clo)}$. Afterwards, each cloned B cell is subject to the hypermutation operator, which mutates any clones M times without an explicit usage of mutation probability. In this work, the number M of mutations is inversely proportional to the fitness function, albeit there exists several approaches (e.g. ones proposed in [6]). Let $e^{-\rho \hat{f}}$ the *mutation rate*, where ρ is an input parameter, and $\hat{f}$ is the fitness function normalized in the range $[0, 1]$, then the number of mutations M on a given candidate solution $\boldsymbol{x}$ is given as $M(\boldsymbol{x}) = \lfloor(\alpha \times \ell)+1\rfloor$, with ℓ the length of $\boldsymbol{x}$. At least one mutation is guaranteed on any B cell, which happens exactly when the candidate solution is very close to the optimal one. Thus, for any B cell the hypermutation operator chooses randomly M times two vertices u and v, and then swaps them. At the end of the hypermutation process, we have a new population that is denoted by $P^{(hyp)}$. To normalized the fitness function in the range $[0, 1]$, as proposed in [9], *MIA* uses the best current fitness value decreased of an *user-defined threshold* Θ; this is necessary because *a priori* is not known any kind of information about global optima. As proposed in [10,6,7], also *MIA* is based on the particular scheme that when an hypermutated B cell improves the value of the fitness (called *constructive mutations*), then it will be considered to have age equal to 0. Using this scheme, we give an equal opportunity to each new B cell to effectively explore the given landscape. To improve the quality of the solution, MIA incorporates an heuristic local search based on the exploration of the neighbourhood, where the neighbours are explored through the swapping of the vertices. This operator is applied to the best B cell of $P^{(hyp)}$, producing a new population $P^{(LS)}$.

Table 1. Pseudo-code of *Memetic Immune Algorithm* – MIA

```
MIA($d, dup, \tau_B, \rho$)
  $P^{(0)} \leftarrow$ Init_Population($d$)
  Evaluate_Fitness($P^{(0)}$)
  $t \leftarrow 1$
  while ($\neg Termination\_Condition()$)do
      $P^{(clo)} \leftarrow$ Cloning ($P^{(t)}, dup$)
      $P^{(hyp)} \leftarrow$ Hypermutation($P^{(clo)}, \rho$)
      Evaluate_Fitness($P^{(hyp)}$);
      $P^{(LS)} \leftarrow$ LocalSearch($P^{(hyp)}[best]$)
      Evaluate_Fitness($P^{(LS)}$);
      Static_Aging($P^{(t)}, P^{(hyp)}, P^{(LS)}, \tau_B$);
      $P^{(t+1)} \leftarrow (\mu + \lambda)$-Selection($P_a^{(t)}, P_a^{(hyp)}, P_a^{(LS)}$);
      $t \leftarrow t + 1$;
end_while
```

After the perturbation operators, all old B cells inside the populations $P^{(t)}$, $P^{(hyp)}$, and $P^{(LS)}$ are eliminated using the *static aging operator*. The parameter τ_B in input indicates the maximum number of generations, that allows to each B cell to remain into own population; when a B cell is $\tau_B + 1$ old it is erased from the own population independently from its fitness value. Each B cell is allowed to remain into the population for a fixed number of generations; an exception is made only for the B cell with the current best fitness value (*elitist static aging operator*). The strength of this operator is to produce a high diversity into the current population, and avoid premature convergences. After the aging operator, follows the $(\mu+\lambda)$*-Selection operator*, which generates the new population $P^{(t+1)}$; it selects the best d survivors B cells from the populations $P^{(t)}$, $P^{(hyp)}$, and $P^{(LS)}$. If only $(d_A < d)$ B cells are survived, then it creates randomly $(d - d_A)$ new B cells (*Birth phase*).

In table 1 we report the pseudo-code of the memetic immunological algorithm, where *Termination_Condition()* is a Boolean function, which returns true if the maximum number of generations, or the maximum number of fitness function evaluations allowed, is reached; false otherwise. Instead, *Evaluate_Fitness()* computes the fitness function value of each B cell using the equation 1.

Local Search. The used approach for the design of the local search was taken from [7,5], and it relies on the definition of neighbourhood, where neighbours are generated through the swapping of the vertices. One B cell $\boldsymbol{y}$ is said a neighbour of a B cell $\boldsymbol{x}$ if it can be obtained from $\boldsymbol{x}$ by swapping two of its elements. Since swapping all pairs of vertices is time consuming, we have used a reduced neighbourhood by a radius R_{LS}, as proposed in [5,8]: in each B cell, all vertices were swapped only with their R_{LS} nearest neighbours, to the left and to the right. Taking into account the large size of the neighbourhood the local search procedure is applied only on the best hypermutated B cell (i.e., the best of $P^{(hyp)}$). If a single swap between two vertices reduces the fitness function value, then the new mutated B cell is added into the new population $P^{(LS)}$; otherwise it is not taken into account, and hence erased. The process continues until the whole neighbourhood with radius R_{LS} will be explored. To avoid the problem to study

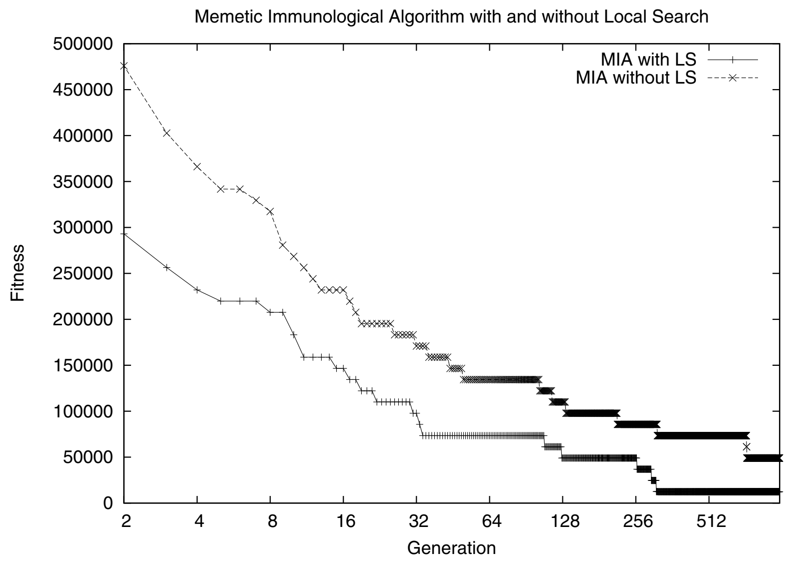

Fig. 2. Convergence process of MIA with and without local search (LS)

which is the best tuning for R_{LS} radius we assign a random value in the range $[1, (|V| - 1)]$, using a uniform distribution. In this way it is guaranteed to swap at least two vertices. Figure 2 shows the convergence process of the best fitness values with and without the local search procedure. The experiment was made on a graph with 82 vertices (planar topology), and fixing the minimal values for the parameters: $d = 100$, $dup = 1$, $\tau_B = 5$, $\rho = 5.5$ and $MaxGen = 1000$. This figure shows how the local search procedure ease the convergence towards better solutions.

Heuristics for the assignment scheme. How to assign a provider to one item or vice versa is a central point in the design of the algorithm. Since any provider has a limited resource capacity, choosing one vertex rather than an other can determine the satisfiability of all received requests, or only some of them. Let $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ a generic B cell; $R = \{r_1, \ldots, r_h\}$ the set of the providers; and $b(r_i)$ the capacity of the ith provider, with $i \in [1, h]$. A provider r_i is randomly chosen to be assigned to the first x_1 vertex of the permutation $\boldsymbol{x}$, and afterwards decreasing the capacity of r_i, i.e. $b_{(curr)}(r_i) = b_{(prev)}(r_i) - q(x_1)$. For all remaining vertices x_j, with $j = (2, \ldots, \ell = \mid V \mid)$, is possible to distinguish the following three cases:

1. if exists a vertex $v \in V$ adjacent to x_j, and a provider $r \in R$ assigned to v (i.e. r supplies v) such that $\sum_{v \in V_r} q(v) + q(x_j) \leq b(r)$, where V_r is the subset of the vertices already assigned to r, then the provider r is assigned to supply x_j. If there exist two or more vertices adjacent to x_j, with assigned different providers suitable to satisfy x_j, then the one with higher available capacity will be assigned to x_j;
2. for all vertices $v \in V$ adjacent to x_j, either there exists no $r \in R$ assigned to v, or if there exists, it is not able to satisfy x_j. Thus, if there are one or more free providers, i.e. not yet assigned, then one of these is randomly chosen, and assigned to x_j; otherwise a deep search is made into the neighbourhood of x_j. If after the search, at least one available provider was found, this is assigned to x_j, otherwise the vertex will be labelled *"not satisfied"*. Of course, case 2 occurs only if the first step failed. In this work, as search model, was used the classical *"depth first search"* algorithm (DFS) [4], but reduced of a radius $R_{(DFS)}$: is fixed a limit $R_{(DFS)} < |V|$ to the depth of the search into the neighbourhood. The aim of this reduced DFS is to satisfy the request of the vertex x_j trough an available provider not too far from it, in such way to generate homogeneous groups. If, by the reduced DFS we found two or more suitable providers then the nearest one is assigned to the vertex x_j. In the experiments described in section 4, $R_{(DFS)}$ was fixed as 15% of $\mid V \mid$. We call this kind of approach *"random + dfs assignment"*.
3. There exists no provider able to satisfy the given vertex (when cases 1 and 2 fail): i.e. for all $r \in R$, $\sum_{v \in V_r} q(v) + q(x_j) > b(r)$, where V_r is the subset of the vertices assigned to r. In this case the vertex will be labelled *"not satisfied"*.

After any assignment, the capacity of each chosen provider r is decreased: $b_{curr}(r) = b_{prev}(r) - q(x_j)$.

It is possible to derive an algorithmic variant of the *"random + dfs assignment"*, which occurs in the case 2: before randomly choosing a free provider to be assigned, this variant checks if there exist an available provider inside the nearest neighbourhood. This is done applying the reduced DFS algorithm, with radius $R_{(DFS)}$ equal to 5% of $|V|$. We call this variant as *"dfs^2 assignment"*. This new scheme guarantees the design of homogeneous groups, that is, all providers will supply only items close to each other.

Figure 3 shows the comparisons of the best fitness values obtained by the two described heuristics. These curves were obtained on a graph with 82 vertices, and using the following parameters: $d = 100$, $dup = 2$, $\tau_B = 15$, $\rho = 5.5$, and $MaxGen = 1000$. The figure shows how dfs^2 allows to MIA a better convergence towards high quality solutions, lower costs and better compactness of the groups. The inset plot shows for both approaches the success rate of the DFS over the

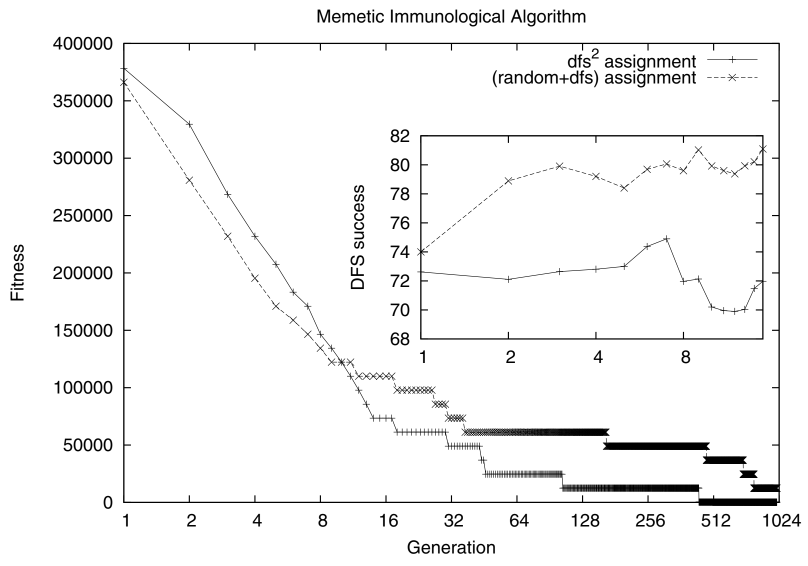

Fig. 3. Convergence process of the best fitness function values for *"random + dfs assignment"* and *"dfs^2 assignment"* heuristics. The inset plot shows the success rate of the DFS.

best run. Obviously, as we expected, the curves of $random + dfs$ are higher than dfs^2, since it obtains poor solutions, and therefore it needs more calls to the DFS with long radius: more calls generate a higher success rate. However, the overall percentages of the DFS successes, averaged with their own calls, computed over 10 independent runs, is higher in dfs^2 (73.08%) than in $random + dfs$ (66.28%). This means that our optimization strategy, which is to reduce the search of an available provider in a nearest neighbourhood, is efficient in order to design very compact groups. In table 2 is showed the comparison between the two approaches, varying the parameters on a graph with 82 vertices (a planar graph). This instance, has been used to evaluate the performances of MIA in terms of fitness value, and the ability to develop homogeneous groups. The table shows the best fitness values, the mean and the standard deviation. Last column, Δ, of the table indicates the differences of the two approaches with respect the best fitness values. In bold face is highlighted the best fitness values for each pairs of parameters. The results have been obtained with $d = 100$, $\rho = 2$, $MaxGen = 100$, and 10 independent runs. The number of the providers h is equal to 6, using the same capacity for all h providers. Given all weights on the vertices, we can consider $\Lambda = \frac{\sum_{i=1}^{n} q(x_i)}{h}$ as lower bound to determine in average how much should be the capacity of each provider to satisfy all request. This lower bound is not necessarily the optimal capacity value. In this work, the averaged capacity is the lower bound Λ increased by 0.2%. Inspecting the results in the table, dfs^2 is suitable to find better solutions and more homogeneous groups.

Table 2. dfs^2 vs. $random + dfs$. For each experiment we report the best fitness value, the mean and the standard deviation. Last column, Δ, indicates the difference of the best fitness function values. For these experiments has been used the graph with $|V| = 82$. In bold face is highlighted the best fitness value for each pairs of parameter *dup* and τ_B.

dup	τ_B	dfs^2	$random + dfs$	Δ
1	5	**236.28** 2675.12 ± 4877.68	12422.63 19742.75 ± 8091.68	−12186.35
	20	**236.04** 1455.39 ± 3658.04	12428.42 20963.34 ± 10967.25	−12192.38
	∞	**228.84** 1453.59 ± 3662.04	12439.09 20967.5 ± 5583.16	−12210.25
5	5	**235.37** 7555.31 ± 5969.99	12443.66 17314.52 ± 5965.55	−12208.29
	20	**232.38** 232.93 ± 1.1	12425.92 17305.04 ± 8091.91	−12193.54
	∞	**244.33** 13647.11 ± 6563.78	12418.84 23396.64 ± 16768.86	−12174.51
10	5	245.00 245.00 ± 0.0	**226.83** 24616.53 ± 21121.18	+18.17
	20	**225.19** 5106.2 ± 14638.93	227.32 227.32 ± 0.0	−2.13
	∞	**230.49** 1452.14 ± 3663.05	12427.99 18526.56 ± 6098.57	−12197.5

We note that high values of mean and standard deviation (tables 2 and 3) are due to β penalty parameter of the objective function (eq. 1); when a solution is not able to satisfy all items required then the objective function returns high values. Thus, high values of mean and standard deviation indicate that the algorithm is not able to satisfy all required in all runs. In all experiments $\beta = 5$.

4 Results

To evaluate the performance of MIA algorithm we have used two different metrics: (1) MIA is able to obtain good approximated solutions using the capacity of the providers as small as possible, and (2) the homogeneity in the assignment of the providers to the vertices, i.e., to avoid that a provider has to supply two vertices placed in opposite sites from a topological point of view. For the experiments, we used a graph with 82 vertices and planar topology. Afterward, to extend our test bed we have tested MIA on several graphs, taken by the *dimacs colouring benchmark* [1]. Once experimentally proved that dfs^2 has better performances than $random + dfs$ with respect to the costs and the homogeneity of the solutions (see table 2), all results presented in this section have been obtained using the dfs^2 heuristic.

Table 3. Best solution, mean of the best solutions, and standard deviation (σ) obtained on the graph with 82 vertices, varying the parameters *dup*, and τ_B. For this class of experiments was fixed $d = 100$, $MaxGen = 100$, and each test was made 10 independently runs. Moreover, we have fixed $\rho = 4$ for all trucks with same capacity, and $\rho = 5.5$ with different capacity values. The shown results were obtained fixing either the same capacity for all used trucks (the lower bound Λ increased by 0.2%), than with different capacity values.

		using same capacity			***using different capacity***		
dup	τ_B	*best*	*mean*	σ	*best*	*mean*	σ
	5	229.45	231.42	1.28	36830.49	53902.77	23894.93
1	20	232.81	233.15	0.68	36845.67	36845.67	0.004
	∞	228.05	2670.81	4884.66 49017.31	57563.87	17294.56	
	5	226.22	5116.51	9750.32	36830.98	38051.81	3662.52
5	20	235.00	5112.00	9754.00	36828.41	36835.69	4.29
	∞	244.64	11213.28	10125.65	**12436.16**	**14877.58**	**7314.36**
	5	229.15	2686.72	4874.28	24639.02	40491.00	17291.04
10	20	**220.92**	**6323.55**	**6102.63**	36825.49	41707.80	5979.59
	∞	229.82	7547.70	9759.12	24635.98	42929.53	23299.84

In table 3 we report the results obtained by MIA using for all providers either the same maximum quantity of resources to be supplied, or different capacities. These experiments have been made varying the parameters $dup = \{1, 5, 10\}$, and $\tau_B = \{5, 20, \infty\}$, and a population size constant ($d = 100$), $\rho = 4$ for the experiments where all resources have the same capacity, and $\rho = 5.5$ for all experiments with different capacity values. Moreover, the maximum number of generations was fixed to 100, and each test has been averaged over 10 independent runs. Inspecting the results obtained using the same capacity, is possible to see that, although the best solution is obtained with high values of *dup* parameter ($dup = 10$), in general the best performances have been obtained using small values of the *dup* parameter. If we use different capacities, instead, $dup = 5$ seems to be the adequate setting.

To simulate a real world application, we have considered the graph as road network, where each weight has been randomly generated in the range $[200, 10000]$ for the vertices, while for the edges in the range $[5, 180]$. Moreover, we have used a small number of providers in order to better simulate a real application. Since in the real world case is likely that not all items require resource allocation, i.e. someone can have weight null, the random generator assigns each weight on vertices with a probability $P = 0.5$.

To understand the real exploration and exploitation capabilities of MIA we have compared MIA with a classical Genetic Algorithm (GA) and a deterministic algorithm based on locally optima choice strategy. For this deterministic algorithm we present three different versions: (1) starting from the vertex V_1 the naive method proceeds sequentially $(V_1, V_2, ..., V_n)$. We labelled this version as *naive*; (2) starting from a random vertex V_k, this method proceeds as follow

Table 4. *MIA* vs. *GA* and three different versions of a deterministic algorithm. These experiments have been made using *dimacs graph colouring instances* as test bed [1], being one of the most popular in literature. For each instance is showed the number of items satisfied (Γ), and relative best cost found. The experiments have been performed for 30 independent runs. We point out that if one of the algorithms is not able to satisfy all requested of the items, the relative costs have been not included in the table (−).

				MIA		*naive*		*DBO*		*DPB*		*GA*	
instance	$\mid V \mid$	$\mid E \mid$	*h*	Γ	*best*	Γ	*best*	Γ	*best*	Γ	*best*	Γ	*best*
DSJC125.1.col	125	1472	4	125	**1018.06**	124	–	123	–	124	–	125	1056.5
DSJC125.5.col	125	7782	4	125	**977.92**	123	–	123	–	124	–	125	1010.92
DSJC125.9.col	125	13922	4	125	**978.532**	124	–	124	–	125	1167	125	1034.34
queen6_6.col	36	580	3	36	**1106.46**	35	–	35	–	35	–	36	1131.4
queen7_7.col	49	952	3	49	**1252.84**	48	–	48	–	49	1304.76	49	1274.53
queen8_8.col	64	1456	3	64	**1276.75**	63	–	63	–	64	1309.96	64	1298.4
queen8_12.col	96	2736	4	96	**1133.69**	94	–	95	–	96	1148.92	96	1146.2
queen9_9.col	81	2112	4	81	**1146.1**	80	–	80	–	80	–	81	1161.33
queen10_10.col	100	2940	4	100	**1157.4**	99	–	99	–	100	1198.01	100	1182.73
queen11_11.col	121	3960	4	121	**1082**	120	–	119	–	121	1127.98	121	1105.54
queen12_12.col	144	5192	5	144	**1252.47**	143	–	143	–	144	1296.76	144	1271.63
queen13_13.col	169	6656	5	169	**1282.47**	168	–	168	–	169	1313.1	169	1297.33
queen14_14.col	196	8372	5	196	**1283.31**	195	–	195	–	196	1307.6	196	1294.6
queen15_15.col	225	10360	6	225	**1103.09**	224	–	224	–	225	1126.5	225	1111.21
queen16_16.col	256	12640	6	256	**1326.89**	255	–	255	–	256	1345.1	256	1335
miles500.col	128	2340	4	128	**1141.23**	127	–	127	–	128	1168.028	128	16833
miles750.col	128	4226	4	128	**1110.67**	127	–	127	–	128	1150	128	1153.65
miles1000.col	128	6432	4	128	**1105.77**	126	–	127	–	128	1143.3	128	1137
miles1500.col	128	10396	4	128	**1096.96**	127	–	127	–	128	1130.4	128	1128.61
myciel5.col	47	236	3	47	**1246.87**	46	–	46	–	47	1280	47	1266.3
myciel6.col	95	755	4	95	**1140.14**	94	–	94	–	95	1147.7	95	1149.1
myciel7.col	191	2360	5	191	**1180.82**	190	–	190	–	191	1194.6	191	1193.7
david.col	87	812	4	87	**1076.67**	86	–	86	–	*n.a.*	–	87	1114.6
games120.col	120	1276	4	120	**1279.02**	120	1301	119	–	120	1292	120	1350
anna.col	138	986	5	138	**984.129**	138	–	138	–	*n.a.*	–	138	44505.3

$(V_k, V_{k+1}, ..., V_n, V_1, ..., V_{k-1})$. We call this second version as *DBO*; (3) the last method performs the optimal locally selection based on a permutation of the vertices (DBP). Table 4 presents the results obtained on this new benchmark. The table indicates the number of items satisfied (Γ) for each algorithm, and relative best cost found. For these experiments, it has been used the following parameters $d = 100$, $dup = 15$, and $\tau_B = 15$. For GA, instead, we have used the best tuning of parameters obtained after several experiments: $pop\_size = 100, P_c = 1.0$, and $P_m = 0.3$. Moreover, in all experiments we have used as stop criterion a maximum number of fitness function evaluations $T_{max} = 5 \times 10^4$ for all graphs with $|V| < 100$, and $T_{max} = 5 \times 10^5$ otherwise. Finally, 30 independent runs have been performed for each instance. *MIA* is able to satisfy all requests received, with respect deterministic algorithms. *MIA* and *GA* have been able to satisfy all request received on different dimensions of the problem (from 36 to 256 vertices). However, comparing MIA and GA is possible to see how the proposed algorithm is able to find better costs in all tested instances, which means that MIA is able to produce more compact groups from a topologically point of view. In the table 4, if one of the algorithms has not been able to satisfy all requested, the relative costs have been not included in the table (labelled as $-$), because the fitness value produced is high due to the penalty factor β (equation 1).

5 Conclusion

From the shown results, the reduced DFS produces good solutions in term of quality and homogeneity of assignments. The designed approach seems to be very promising. Currently, our research is primarily directed, (1) on the study of the best tuning of the DFS (i.e., the radius R_{DFS}); (2) design a good refinement operator, such to improve the convergence speed of MIA; (3) and finally take into account the dynamical environment where the weights either on the vertices and on the edges may change during the time.

All results have been obtained using as capacity, the lower bound Λ increased by 0.2%, i.e. as small as possible, to properly understand the search ability of MIA. The proposed algorithm has been compared with standard GAs and with three different versions of a deterministic algorithm. MIA outperforms the compared algorithms; MIA satisfies always all the requests, as opposed to compared algorithms, which fail on several instances.

References

1. Graph Colouring Instances, `http://mat.gsia.cmu.edu/COLOR/instances.html`
2. Bretthauer, K., Shetty, B.: The nonlinear resource allocation problem. Operations Research 43(4), 670–683 (1995)
3. Coffman, E.G., Garey, M.R., Johnson, D.S.: Approximation Algorithms for Bin Packing: A Survey. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 46–93. PWS Publishing, Boston (1997)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
5. Cutello, V., Nicosia, G., Pavone, M.: A Hybrid Immune Algorithm with Information Gain for the Graph Colouring Problem. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 171–182. Springer, Heidelberg (2003)
6. Cutello, V., Nicosia, G., Pavone, M.: Exploring the capability of immune algorithms: a characterization of hypermutation operators. In: Nicosia, G., Cutello, V., Bentley, P.J., Timmis, J. (eds.) ICARIS 2004. LNCS, vol. 3239, pp. 263–276. Springer, Heidelberg (2004)
7. Cutello, V., Nicosia, G., Pavone, M.: An Immune Algorithm with Hyper-Macromutations for the Dill's 2D Hydrophobic - Hydrophilic Model. In: Proc. of Congress on Evolutionary Computation (CEC 2004), vol. 1, pp. 1074–1080. IEEE Press, Los Alamitos (2004)

8. Cutello, V., Nicosia, G., Pavone, M.: An immune algorithm with stochastic aging and kullback entropy for the chromatic number problem. Journal of Combinatorial Optimization 14(1), 9–33 (2007)
9. Cutello, V., Nicosia, G., Pavone, M., Narzisi, G.: Real Coded Clonal Selection Algorithm for Unconstrained Global Numerical Optimization using a Hybrid Inversely Proportional Hypermutation Operator. In: SAC 2006, vol. 2, pp. 950–954 (2006)
10. Cutello, V., Nicosia, G., Pavone, M., Timmis, J.: An Immune Algorithm for Protein Structure Prediction on Lattice Models. IEEE Transaction on Evolutionary Computation 11(1), 101–117 (2007)
11. Fukunaga, A.S., Korf, R.E.: Bin completion algorithms for multicontainer packing, knapsack, and covering problems. Journal of Artificial Intelligence Research 28, 393–429 (2007)
12. Ibaraki, T., Katoh, N.: Resource Allocation Problems – Algorithmic Approaches. MIT Press, Cambridge (1988)
13. Lin, X., Johansson, M., Boyd, S.P.: Simultaneous routing and resource allocation via dual decomposition. IEEE Transactions on Communications 52(7), 1136–1144 (2004)
14. Morris, M.D.: Factorial sampling plans for preliminary computational experiments. Technometrics 33(2), 161–174 (1991)
15. Raidl, G.R., Kodydek, G.: Genetic Algorithms for the Multiple Container Packing Problem. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 875–884. Springer, Heidelberg (1998)
16. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: On the Capacitated Vehicle Routing Problem. Mathematical Programming 94, 343–359 (2003)
17. Toth, P., Vigo, D.: The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications (2002)