# A Variable Local Search Based Memetic Algorithm for the Load Balancing Problem in Cloud Computing

Nasser R. Sabar[1(✉)], Andy Song[1], and Mengjie Zhang[2]

[1] School of Computer Science and I.T., RMIT University, Melbourne, Australia
{nasser.sabar,andy.song}@rmit.edu.au
[2] Victoria University of Wellington, Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

**Abstract.** Load balancing (LB) is an important and challenging optimisation problem in cloud computing. LB involves assigning a set of services into a set of machines for which the goal is to optimise machine usages. This study presents a memetic algorithm (MA) for the LB problem. MA is a hybrid method that combines the strength of population based evolutionary algorithms with local search. However the effectiveness of MA mainly depends on the local search method chosen for MA. This is because local search methods perform differently for different instances and under different stages of search. In addition, invoking local search at every generation can be computationally expensive and compromise the exploration capacity of search. To address these issues, this study proposes a variable local search based MA in the context of LB problem. The proposed MA uses multiple local search mechanisms. Each one navigates a different area in search space using a different search mechanism which can leads to a different search path with distinct local optima. This will not only help the search to avoid being trap in a local optima point, but can also effectively deal with various landscape search characteristics and dynamic changes of the problem. In addition, a diversity indicator is adopted to control the local search processes to encourage solution diversity. Our MA method is evaluated on instances of the Google machine reassignment problem proposed for the ROADEF/EURO 2012 challenge. Compared with the state of the art methods, our method achieved the best performance on most of instances, showing the effectiveness of variable local search based MA for the Load Balancing problem.

**Keywords:** Local search · Memetic algorithms · Load balancing · Cloud computing · Meta-heuristics

## 1 Introduction

Cloud computing is a fast growing technology that provides on-demand computing services over the Internet [3,6]. It offers network access to a various shared pool of configurable computing resources including storage, processing,


G. Squillero and P. Burelli (Eds.): EvoApplications 2016, Part I, LNCS 9597, pp. 267–282, 2016.
DOI: 10.1007/978-3-319-31204-0_18

bandwidth and memory. A cloud provider, such as Google and Amazon, manages a data centre of which the computing resources are to be shared by end users. With the rapid growth of the demand in cloud services, optimal resources allocation becomes one the most important targets in cloud computing [6].

Load balancing (LB) is one of the cloud resource allocation tasks seeking for the best arrangement of services into a set of machines so the usage of these machines can be improved. In this paper, we consider the LB problem introduced by Google for the ROADEF/EURO 2012 challenge [1]. The task is named as *Machine Reassignment Problem* (MRP). The goal of MRP is to improve the usage of resources by reassigning a set of processes into a set of machine, while all problem constraints must be satisfied. A range of methods have been proposed to solve MRP. These include variable neighbourhood search [8], constraint programming-based large neighbourhood search [15], large neighbourhood search [4], multi-start iterated local search [14], simulated annealing [20] and restricted iterated local search [13].

In this study, we propose a memetic algorithm (MA) based method for this load balancing problem. MA is a stochastic optimisation search method which combines population based algorithm with local search. The rationale of MA is to synergise the exploration power of population based algorithms with the exploitation capability of local search [16]. MA has been proven very successful in solving various difficult optimisation problems [17]. However the success of MA is not automatic [21,22,25]. There are two important aspects that have to be considered when designing MA for a particular problem [18]. Firstly, the choice of local search is important. The performance of MA heavily depends on the selected local search algorithm. It is difficult for one local search to fit with diverse features of different instances of different problems. Even for the same instance, the characteristic of search space under different stages may vary significantly [24]. That makes the choice of local search method difficult and critical. Secondly, MA often faces the challenge of how to preserve the diversity of a search process [23]. Excessive use of local search may consume more computation on exploitation compromising the effort on exploration. To address these two issues, we propose a variable local search based memetic algorithm. It combines genetic algorithm (GA) with multiple local search algorithms in which each one can navigate a different area in the search space. Different search mechanisms can lead to a different search path with distinct local optima. Furthermore a diversity indicator is adopted to control the invocation of local search to prevent lost of diversity in the population of solutions. With the proposed method, there is no need to examine the nature of a load balancing problem and to choose an appropriate local search for the problem. The need of tuning the local search is also unnecessary in the proposed MA approach.

The proposed algorithm are evaluated on small and large scale instances of the machine reassignment problem from ROADED/EURO 2012 challenge. For comparison purposes, the state of the art algorithms for this challenge are included as well. In Sect. 2, this challenge is described in details. The proposed variable local search based MA is presented in Sect. 3. Section 4 shows the experiment settings while the results are listed in Sect. 5. Section 6 concludes this study.

## 2 Problem Description

The so-called machine reassignment problem (MRP) introduced by Google [1] for the ROADEF/EURO 2012 challenge is load balancing problem. It is a challenging combinatorial optimisation problem which is to find the optimal way to assign processes to machines in order to improve the usage of a given set of machines. One machine consists of a set of resources such as RAM and CPUs. One process can be moved from one machine to another to improve overall machine usage. The allocation of processes must not violate the following hard constraints:

- *Capacity constraints*: the sum of requirements of resource of all processes should be less than or equal to the capacity of the allocated machine.
- *Conflict constraints*: processes of the same service should be allocated into different machines.
- *Transient usage constraints*: if a process is moved from one machine to another, it requires adequate amount of capacity on both machines.
- *Spread constraints*: the set of machines is partitioned into locations and processes of the same service should be allocated to machines in a number of distinct locations.
- *Dependency constraints*: the set of machines are partitioned into neighbourhoods. Then, if there is a service depends on another service, then the process of first one should be assigned to the neighbouring machine of second one or vice versa.

A solution to MRP is a process-machine assignment which satisfies all hard constraints and minimises the weighted cost function as much as possible which is calculated as follows:

$$\begin{aligned} f = & \sum_{r \in R} weight_{loadCost}(r) \times loadCost(r) \\ & + \sum_{b \in B} weight_{balanceCost}(b) \times balanceCost(b) \\ & + weight_{processMoveCost} \times processMoveCost \\ & + weight_{serviceMoveCost} \times serviceMoveCost \\ & + weight_{machineMoveCost} \times machineMoveCost \end{aligned} \tag{1}$$

where $R$ is a set of resources, $loadCost$ represents the used capacity by resource $r$ which exceeds the safety capacity, $balanceCost$ represents the use of available machine, $processMoveCost$ is the cost of moving a process from its current machine to a new one, $serviceMoveCost$ represents the maximum number of moved processes over services and $machineMoveCost$ represents the sum

of all moves weighted by relevant machine cost. $weight_{loadCost}$, $weight_{balanceCost}$, $weight_{processMoveCost}$, $weight_{serviceMoveCost}$ and $weight_{machineMoveCost}$ define the importance of each individual cost.

The detailed explanation of the constraints, the costs and their weights can be found on the challenge documentation [1]. Note that the quality of a solution is evaluated by the given solution checker, which returns fitness measure to the best solution generated by our MA. Another important aspect of this challenge is the time limit. It was stated that *"The maximum execution time will be fixed to 5 min by instance on a core2duo E8500 3.16 MHz with 4Go RAM on debian 64 or Win7 64 bits."* All methods have to finish within the 5 min timeframe to ensure the fairness of the comparison.

## 3 Methodology

Hybridised algorithms have recently received increased interest from the optimisation research community [17]. It is expected that integrating the components of multiple algorithms under one framework may result in a more effective and efficient optimisation method [19]. One of such hybridisation frameworks is the Memetic Algorithms (MAs) [16,17]. MA is a class of search methods that merge the strengths of population-based algorithms and local search algorithms. Local search is to improve the convergence of traditional population-based algorithms by exploiting the surrounding area of the evolved solutions in the search space. MAs can not only produce high quality solutions but also converge faster than other methods. However, as mentioned before the performance of a MA highly depends on its local search method of which the suitability is problem dependent. In addition the excessive invocation of local search may harm the exploration. The balance between exploration and exploitation, that is the balance between population-based search versus local search, should be carefully maintained.

Our proposed MA is to address these aforementioned issues. It introduces a set of local search algorithms, which are invoked according to the search process, and a diversity indicator to balance the exploration and exploitation. Figure 1 shows the overall flowchart of the proposed MA, which is based on steady-state genetic algorithms (GA). This choice is mainly due to the 5 min time limit imposed on this machine reassignment challenge. The process shown in the figure is actually similar to that of classic steady-state GA. The main different is the addition of the diversity control after mutation. The diversity condition of each new solution will be checked at this step. If the condition is satisfied then the solution will be sent to the variable local search component for improvement. Basically solutions are to be improved by a sequence of local search. A solution that can not be improved will be abandoned. Otherwise it will be added back to the population.

A main purpose of the variable local search is to address the question of "*which local search should be used on which solution?*". Multiple local search strategies are to be applied depending on the individual situation. The detail of the process is explained in Sect. 3.3. Other main components of the proposed MA are also presented in details in the following subsections.

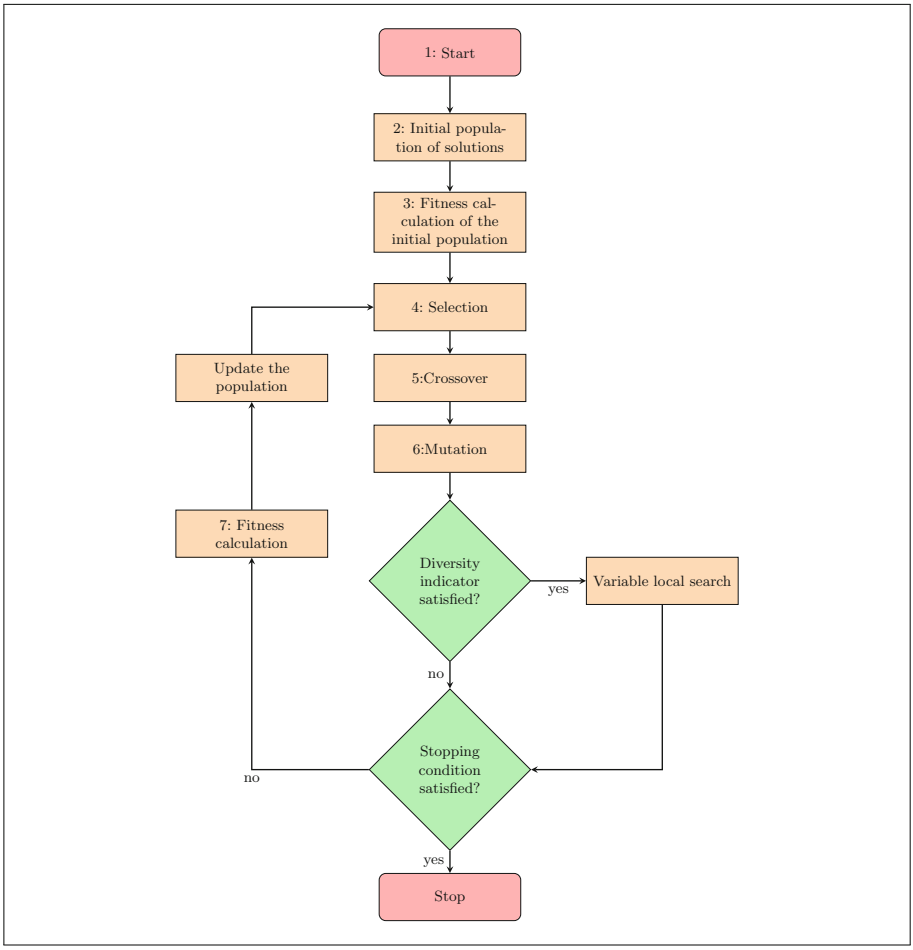


**Fig. 1.** The overall flowchart of the proposed MA

### 3.1 Genetic Algorithm

The proposed MA method is based on steady-state genetic algorithm which is a variation of GA, the well-known nature inspired population based meta-heuristic that mimics the process of natural selection [9].

A solution for one MRP instance is represented as a chromosome as shown in Fig. 2. The number of alleles is the number of available machines. Each allele encodes the processes that are to be executed by the corresponding machine. For example machine $M1$ on the figure will run processes $p1, p9, p3, p20$ in sequence. In this example there are 22 processes in total. A valid solution should contain all of these process while there is no duplicates. Furthermore the solution must satisfy the constraints that mentioned in Sect. 2.

| *Machine* | ***M1*** | ***M2*** | ***M3*** | ***M4*** | ***M5*** | ***M6*** | ***M7*** | ***M8*** | ***M9*** | ***M10*** |
|---|---|---|---|---|---|---|---|---|---|---|
| | *p2* | *p7* | *p14* | *p16* | *p1* | *p11* | *p12* | *p13* | *p5* | *p23* |
| *Processes* | *p9* | *p8* | *p15* | *p17* | *p10* | *p4* | | *p21* | | *p19* |
| | *p3* | | | | *p22* | | | *p18* | | |
| | *p20* | | | | | | | *p6* | | |

**Fig. 2.** Solution representation for MRPs

As for the evolutionary aspect, the majority of the algorithm is inherited from steady-state GA. The main steps are shown below:

* **Step 1:** Start. GA parameters are initialised including:
  - Population size: the number of solutions in the population;
  - Maximum generations: unspecified, the last generation before reaching the 5 min limit;
  - Crossover rate: the percentage of selected solutions participating in crossover;
  - Mutation rate: the percentage of selected solutions participating in mutation.

* **Step 2:** Initial population of solutions. A set of solutions are randomly created and the feasible ones are used to fill the initial population. For MRP, Google provides an initial solution for every problem instance [1]. The initial population are generated by randomly modifying this solution many times.
* **Step 3:** Fitness calculation of the initial population. The fitness value of each solution is its weighted cost calculated by Eq. (1).
* **Step 4:** Selection. Roulette wheel selection is used in this study where the probability of selecting a solution is proportional to its fitness.
* **Step 5:** Crossover. This is to create new individuals by exchanging genetic materials between two selected solutions. Order crossover operator (OX) is used here to avoid generating invalid solutions for MRP. In OX, two cut points are randomly selected. The alleles between the cut points on the first parent are copied to one child. These identical alleles are removed from the second parent which will contribute its remaining alleles to fill the other parts of the child. The same process can repeat by copying alleles between cut points from the second parent and other alleles from the first parent to generate another child. The individuals generated by OX has no missing alleles or duplications. They are the rearrangement of their parents. This is particular suitable for MRP.
* **Step 6:** Mutation. The mutation operator modifies one solution by randomly selecting two different alleles from the gene and then swap their positions, for example a solution of $M2$ running $\{p7, p8\}$ and $M7$ running $p12$ becomes a new solution that allocates $\{p7, p8\}$ to $M7$ and $p12$ to $M2$ after the mutation.
* **Step 7:** Fitness calculation. The fitness value of the generated solutions are again calculated using Eq. (1).

### 3.2 Diversity Indicator

An important aspect of MA is to avoid exploitation dominating exploration. To attain high quality solutions, balance should be maintained between the exploration from GA and the exploitation from local search. Excessive invocation of local search, for example running it at every generation, is costly and may increases the risk of the premature convergence, due to bias towards exploitation. Therefore local search should be applied only when it is necessary. In this study a diversity indicator is introduced to control when local search is needed by measuring the diversity of the current population. The measurement is from the work of Neri and Cotta [18]:

$$\text{Diversity} = 1 - \left| \frac{f_{\text{avg}} - f_{\text{best}}}{f_{\text{worst}} - f_{\text{best}}} \right| \tag{2}$$

where $f_{avg}$ is the average fitness of the population of solutions, $f_{best}$ is the fitness of the best solution and $f_{worst}$ is the fitness of the worst solution in that population. If the *Diversity* is greater than a pre-defined value *threshold*, local search will be invoked to improve the solution generated after crossover and mutation.

### 3.3 Variable Local Search

Local search starts with a single solution and attempts to improve it by exploiting local solution space until the predefined termination criterion is reached [2,27]. In other words, to improve a given solution, local search will try to find the best neighbour. One major drawback of traditional local search is being trapped in the so-called local minima. In the literature there are different mechanisms to guide search away from non-improving areas and continue the search beyond the local optima. Examples of the well-known local search include simulated annealing [11], great deluge [7] and the late acceptance hill climbing [5].

It is not known in advance which local search would be the most suited to perform well on all problem instances for a memetic algorithm [12]. Therefore we propose a new approach in which multiple local search will be executed in sequence. Each one can utilise different rules to navigate different area in the search space. The integration of these local search can help avoid local optima and lead to high quality solutions for majority of instances.

Four local search algorithms are used in our variable local search mechanism. Each solution that is sent to this component for improvement will be assigned to a sequence of local search algorithms. This sequence consists of all four algorithms but in a random order. If the first local search resulted in an improved solution then the search would stop. The improved solution will be added into the population. The index of this local search is also recorded. If the stopping condition of the first search algorithm is met but no improvement can be made, then the second local search will be applied. This process repeats until an improved solution is found or the last local search in the sequence is terminated. Note the sequence of local search is circular. For example if a solution starts with local

search No.3, then the sequence of execution is local search No. 4, No.1 and No.2, if no improvement can not found early.

In the case of no better solution found after trying all four local search mechanisms, the original solution will return to population with no further modification. The index of the last successful local search will remain unchanged. Then the local search process will start over again on the next solution.

The four local search algorithms to construct the sequence are:

- **Steepest Descent (SD)** [26]. Given an initial solution, $S_0$, generates a neighbourhood solution, $S_1$, by randomly selecting one process and then move it into a different machine. Replace $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$. If not, $S_1$ will be rejected and a new iteration will be started. The search process will be terminated after a fixed number of non-improving iterations. In our preliminary experiment that number is set as 10.

- **Simulated Annealing (SA).** SA was introduced by [11]. It tries to escape local optima by accepting worse solutions. Given an initial solution, $S_0$, generates a neighbourhood solution, $S_1$, by randomly selecting one process and then move it into a different machine. Replace $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$ or satisfying the probability condition, $R < PA$, where $R$ is uniform random number in [0,1] interval and $PA$ is calculated as follows:

$$PA = \exp(-\delta/t) \tag{3}$$

  where $\delta$ is the change in the fitness values of $S_0$ and $S_1$, $t$ is the temperature which is set to 50 % of the fitness value of $S_0$. The $t$ value controls the acceptance ration of worse solutions and its gradually decreases by $\alpha$ ($\alpha = 0.85$) during the search process. The search process will terminate when $t = 0$.

- **Great Deluge (GD).** GD was introduced by [7]. Similar to SA, GD also accepts worse solutions in order to get out of a local optima point but using different rules. Given an initial solution, $S_0$, generates a neighbourhood solution, $S_1$, by randomly selecting one process and then move it into a different machine. Replace $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$ or lower than the *level*. Initially, the value of *level* is set equal to the fitness value of the initial solution. At each iteration, the value of *level* is decreased by $\varepsilon$ as follows:

$$level = level - \varepsilon \tag{4}$$

  and

$$\varepsilon = (f(S_1) - f(best_{sol}))/NI \tag{5}$$

  where $NI$ is the number of iterations which is fixed to 1000 (the 1000 was determined based on a preliminary test). The search process will stop when the *level* value is lower than the best solution found so far.

– **Late Acceptance Hill Climbing.** This is an improved variation of hill climbing [5]. It also attempts to escape from local optima point by accepting worse solutions. Its main idea is to accept the generated solution if it is not worse than the quality of a saved solution which was the current one several steps before. Given an initial solution, $S_0$, generates a neighboured solution, $S_1$, by randomly selecting one process and then move it into a different machine. Replace $S_1$ with the $S_0$ if the quality of $S_1$ is better than $S_0$ or better than $f_v$ where $f_v$ is the quality of $v^{it}$ solution saved in list $L$ that contains qualities of current solutions during a number of recent iterations. $v$ is calculated as follows:

$$v = I \mod L_{size} \tag{6}$$

where $L_{size}$ is the size of the list $L$ and $I$ is the iteration counter. At each iteration, LA will insert the quality of the current solution into the beginning of $L$ and removes the last one from the end. In this work, $L_{size}$ size was set to 20 and the search will terminate after 10 consecutive none improving iterations.

## 4 Experimental Settings

This section briefly introduces the MRP instances from the ROADEF/EURO 2012 challenge and presents the parameter settings of our proposed MA.

### 4.1 Problem Instances

Two groups of instances from the ROADEF/EURO 2012 challenge are named as $a$ and $b$. Group $a$ has two subgroups $a1$ and $a2$. Each group contains 10 instances with diverse characteristics in terms of the number of machines, the number of processes, neighbourhood, and so on. Table 1 shows the main characteristics of these instances. In the table, $R$ is the number of resources; $TR$ is the number of resources that need transient usage; $M$ is the number of machines; $P$ is the number of processes; $S$ is the number of services; $L$ is the number of locations; $N$ is the number of neighbourhoods; $B$ is number of triples and $SD$ is the number of service dependencies.

### 4.2 Parameters Settings

The proposed MA contains mainly six parameters, as shown in Table 2. The setting of these parameters was determined by our preliminary experiments. In particular, for each parameter, different values were tested and the one that leads to the best trade-off between the computational time and solution quality is selected [10].

**Table 1.** The characteristics of the problem instances

| Instance | R | TR | M | P | S | L | N | B | SD |
|---|---|---|---|---|---|---|---|---|---|
| a1_1 | 2 | 0 | 4 | 100 | 79 | 4 | 1 | 1 | 0 |
| a1_2 | 4 | 1 | 100 | 1000 | 980 | 4 | 2 | 0 | 40 |
| a1_3 | 3 | 1 | 100 | 1000 | 216 | 25 | 5 | 0 | 342 |
| a1_4 | 3 | 1 | 50 | 1000 | 142 | 50 | 50 | 1 | 297 |
| a1_5 | 4 | 1 | 12 | 1000 | 981 | 4 | 2 | 1 | 32 |
| a2_1 | 3 | 0 | 100 | 1000 | 1000 | 1 | 1 | 0 | 0 |
| a2_2 | 12 | 4 | 100 | 1000 | 170 | 25 | 5 | 0 | 0 |
| a2_3 | 12 | 4 | 100 | 1000 | 129 | 25 | 5 | 0 | 577 |
| a2_4 | 12 | 0 | 50 | 1000 | 180 | 25 | 5 | 1 | 397 |
| a2_5 | 12 | 0 | 50 | 1000 | 153 | 25 | 5 | 0 | 506 |
| b_1 | 12 | 4 | 100 | 5000 | 2512 | 10 | 5 | 0 | 4412 |
| b_2 | 12 | 0 | 100 | 5000 | 2462 | 10 | 5 | 1 | 3617 |
| b_3 | 6 | 2 | 100 | 20000 | 15025 | 10 | 5 | 0 | 16560 |
| b_4 | 6 | 0 | 500 | 20000 | 1732 | 50 | 5 | 1 | 40485 |
| b_5 | 6 | 2 | 100 | 40000 | 35082 | 10 | 5 | 0 | 14515 |
| b_6 | 6 | 0 | 200 | 40000 | 14680 | 50 | 5 | 1 | 42081 |
| b_7 | 6 | 0 | 4000 | 40000 | 15050 | 50 | 5 | 1 | 43873 |
| b_8 | 3 | 1 | 100 | 50000 | 45030 | 10 | 5 | 0 | 15145 |
| b_9 | 3 | 0 | 1000 | 50000 | 4609 | 100 | 5 | 1 | 43437 |
| b_10 | 3 | 0 | 5000 | 50000 | 4896 | 100 | 5 | 1 | 47260 |

**Table 2.** Parameter settings of MA

| Parameter | Tested range | Suggested value |
|---|---|---|
| Population size | 5–50 | 30 |
| Crossover rate | 0.1–0.9 | 0.7 |
| Mutation rate | 0.1–0.9 | 0.2 |
| threshold value $th$ | 0.1–0.99 | 0.5 |
| Maximum number of non-improving iterations for SD and LA | 5–50 | 10 |
| Number of iterations for GD | 100–1500 | 1000 |

## 5 Results and Comparison

In this section, we first evaluate the effectiveness of the proposed MA by comparing it with basic GA and its counterpart that did not use the proposed variable local search enhancement. Secondly, the performance of the proposed MA is compared with the state of the art algorithms for MRP.

**Table 3.** The $p$-values of comparing MA with GA, MA1-5 on all instances

| MA vs. | GA | MA1 | MA2 | MA3 | MA4 | MA5 |
|---|---|---|---|---|---|---|
| Instance | p-value | p-value | p-value | p-value | p-value | p-value |
| a1_1 | 0 | 0.01 | 0.06 | 0.03 | 0.05 | 0.07 |
| a1_2 | 0 | 0.02 | 0.02 | 0.01 | 0 | 0.06 |
| a1_3 | 0 | 0 | 0.01 | 0.02 | 0.01 | 0.03 |
| a1_4 | 0 | 0.04 | 0 | 0 | 0 | 0.04 |
| a1_5 | 0 | 0.04 | 0.01 | 0.01 | 0.01 | 0.5 |
| a2_1 | 0 | 0 | 0 | 0 | 0 | 0 |
| a2_2 | 0 | 0 | 0.01 | 0 | 0 | 0 |
| a2_3 | 0 | 0 | 0 | 0 | 0 | 0.03 |
| a2_4 | 0 | 0.01 | 0.04 | 0 | 0 | 0 |
| a2_5 | 0 | 0.02 | 0 | 0 | 0.03 | 0.02 |
| b_1 | 0 | 0 | 0 | 0 | 0 | 0.01 |
| b_2 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_3 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_4 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_5 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_6 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_7 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_8 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_9 | 0 | 0 | 0 | 0 | 0 | 0 |
| b_10 | 0 | 0 | 0 | 0 | 0 | 0 |

### 5.1 Comparing the Proposed MA with GA and Other MAs

To evaluate the effectiveness of the proposed MA, it is compared with following algorithms which are similar to our method but with missing parts:

- **GA:** steady-state GA as described in Sect. 3.1.
- **MA1:** uses Steepest Descent only.
- **MA2:** uses Simulated Annealing only.
- **MA3:** uses Great Deluge only.
- **MA4:** uses Late Acceptance Hill Climbing only.
- **MA5:** no diversity indicator scheme described in Sect. 3.2.

The proposed MA and all the methods for comparison, including GA, MA1, MA2, MA3, MA4 and MA5, were tested with 31 independent runs on all instances from both group $a$ and group $b$. All runs have the same computational resources, terminating within 5 min.

The final results of each method over these 31 runs are statistically compared using the Wilcoxon statistical test with a significance level of 0.05. The $p$-values of MA versus all of these methods are shown in Table 3. In the table, a $p$-value less than 0.05 means MA is statistically better than the compared algorithm. A value greater than 0.05 indicates the good performance of our proposed MA is not so significant. This table does not include the actual cost achieved by these methods. The costs obtained by our proposed MA can be found in the Sect. 5.2.

As can be seen from the table, MA is statistically better than GA with no local search on all 20 instances. MA is also significantly better than other memetic algorithms. It outperformed MA1 (steepest descent only) and MA3 (great deluge only) on all 20 instances. Comparing with MA2 (simulated annealing only) and MA4 (late acceptance hill climbing), there is only one instance ($a1\_1$) that our MA is not significantly better. Comparing with MA5, which is the most similar to the proposed MA just without the diversity indicator, the proposed MA achieved significant better results on 17 instances.

This positive result clearly justifies the benefits of the proposed variable local search enhancements. In particular, the proposed MA outperformed the basic GA. That demonstrates the benefit of local search on evolutionary search, the combination of exploitation with exploration. The comparison with MA5 shows the benefit of the diversity indicator method.

### 5.2 Comparing with the State of the Art Methods

In this section, the results obtained by the proposed MA are compared with those obtained by the state of the art algorithms. The six algorithms are:

1. **VNS:** Variable neighbourhood search [8].
2. **CLNS:** CP-based large neighbourhood search [15].
3. **LNS:** Large neighbourhood search [4].
4. **MILS:** Multi-start iterated local search [14].
5. **SA:** Simulated annealing [20].
6. **RILS:** Restricted iterated local search [13].

Table 4 presents the comparison results which are the cost of the best solution obtained by our proposed MA and that of the other six algorithms (VNS, CLNS, LNS, MILS, SA) on all 20 instances. Note that the computational resources of these seven methods are identical as all of them have to complete the search within 5 min. For MRPs the lower the cost the better the solution. The best solution for one instance is highlighted in bold. There might be multiple results shown in bold for the same instance as all of them reached the same best value. The results of VNS, CLNS, LNS, MILS, SA are reported by the authors. Among all these methods, RILS did not have results on group $a$ instances.

**Table 4.** The results MA compared to the state of the art algorithms

| Instance | MA | VNS | CLNS | LNS | MILS | SA | RILS |
|---|---|---|---|---|---|---|---|
| a1_1 | **44,306,501** | **44,306,501** | **44,306,501** | 44,306,575 | **44,306,501** | 44,306,935 | – |
| a1_2 | **777,533,308** | 777,536,907 | 778,654,204 | 788,074,333 | 780,499,081 | 777,533,311 | – |
| a1_3 | **583,005,810** | 583,005,818 | 583,005,829 | 583,006,204 | 583,006,015 | 583,009,439 | - |
| a1_4 | **250,866,958** | 251,524,763 | 251,189,168 | 278,114,660 | 258,024,574 | 260,693,258 | - |
| a1_5 | **727,578,310** | **727,578,310** | 727,578,311 | 727,578,362 | 727,578,412 | 727,578,311 | - |
| a2_1 | **164** | 199 | 196 | 1,869,113 | 167 | 222 | - |
| a2_2 | **720,671,537** | 720,671,548 | 803,092,387 | 858,367,123 | 970,536,821 | 877,905,951 | - |
| a2_3 | 1,193,311,432 | **1,190,713,414** | 1,302,235,463 | 1,349,029,713 | 1,452,810,819 | 1,380,612,398 | - |
| a2_4 | 1,680,596,746 | 1,680,615,425 | 1,683,530,845 | 1,689,370,535 | 1,695,897,404 | **1,680,587,608** | - |
| a2_5 | 312,124,226 | 309,714,522 | 331,901,091 | 385,272,187 | 412,613,505 | **310,243,809** | - |
| b_1 | **3,302,947,648** | 3,307,124,603 | 3,337,329,571 | 3,421,883,971 | 3,516,215,073 | 3,455,971,935 | 3,511,150,815 |
| b_2 | **1,011,789,473** | 1,015,517,386 | 1,022,043,596 | 1,031,415,191 | 1,027,393,159 | 1,015,763,028 | 1,017,134,891 |
| b_3 | 158,102,214 | **156,978,411** | 157,273,705 | 163,547,097 | 158,027,548 | 215,060,097 | 161,557,602 |
| b_4 | 4,677,819,137 | 4,677,961,007 | **4,677,817,475** | 4,677,869,484 | 4,677,940,074 | 4,677,985,338 | 4,677,999,380 |
| b_5 | 923,311,250 | 923,610,156 | 923,335,604 | 940,312,257 | 923,857,499 | **923,299,310** | 923,732,659 |
| b_6 | **9,525,857,758** | 9,525,900,218 | 9,525,867,169 | 9,525,862,018 | 9,525,913,044 | 9,525,861,951 | 9,525,937,918 |
| b_7 | 14,836,237,140 | **14,835,031,813** | 14,838,521,000 | 14,868,550,671 | 15,244,960,848 | 14,836,763,304 | 14,835,597,627 |
| b_8 | **1,214,411,947** | 1,214,416,705 | 1,214,524,845 | 1,219,238,781 | 1,214,930,327 | 1,214,563,084 | 1,214,900,909 |
| b_9 | **15,885,546,811** | 15,885,548,612 | 15,885,734,072 | 15,887,269,801 | 15,885,617,841 | 15,886,083,835 | 15,885,632,605 |
| b_10 | 18,051,241,638 | **18,048,499,616** | 18,049,556,324 | 18,092,883,448 | 18,093,202,104 | 18,049,089,128 | 18,052,239,907 |

It can observed that the proposed MA is very competitive in comparison with these state-of-the-art methods. It achieved the lowest cost on 12 out of 20 instances. Among these 12 cases, 11 of them are the new best, meaning being higher than all other methods. On instance $a1\_1$, the proposed MA achieved the best result equivalent to that of VNS, CNS and MILS.

The second best is the VNS method which is the leader on 6 instances. Method SA championed 3 instances. Method CLNS was the best on 2 instances while method MILS achieve the best cost on instance $a1\_1$. Method RILS did not lead on any of the instances. Our proposed MA outperformed VNS on 14 instances, SA on 17 instances, CLNS on 18 instances, MILS on 19 instances, LNS and RILS on all tested instances. Overall, the comparison results clearly show that the proposed MA is an effective method for the MRP.

## 6 Conclusions

This study proposed a memetic algorithm for the load balancing problems. It combines the strengths of genetic algorithm and local search. The proposed algorithm integrates two important aspects to improve the performance of traditional memetic algorithms. Firstly, it uses multiple local search mechanisms to avoid getting stuck in a local optimum and to effectively deal with various types of search space. Four different local search algorithms were used in a sequential manner to improve the solutions evolved by genetic algorithm. Secondly, a diversity indicator was used to control the invocation of local search in order to avoid losing diversity caused by excessive exploitation. The performance of the proposed algorithm was evaluated using the Google machine reassignment benchmark instances that were used during the ROADEF/EURO 2012 challenge. The proposed MA method outperformed general GA, GA with single local search and memetic algorithm without diversity indicator. In comparison with the state-of-the-art algorithms, our proposed MA method is also very competitive achieving the best performance on most instances. We conclude that the variable local search based memetic algorithm is a good method for solving load balancing problem.

## References

1. Roadef/euro challenge 2012: Machine reassignment. http://challenge.roadef.org/2012/en/
2. Emile Aarts, H.L., Lenstra, J.K.: Local Search in Combinatorial Optimization. Princeton University Press, Princeton (2003)
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
4. Brandt, F., Speck, J., Völker, M.: Constraint-based large neighborhood search for machine reassignment. Ann. Oper. Res., 1–29 (2012)
5. Burke, E.K., Bykov, Y.: A late acceptance strategy in hill-climbing for exam timetabling problems. In: PATAT 2008 Conference, Montreal, Canada (2008)

6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Experience **41**(1), 23–50 (2011)
7. Dueck, G.: New optimization heuristics: the great deluge algorithm and the record-to-record travel. J. Comput. Phys. **104**(1), 86–92 (1993)
8. Gavranović, H., Buljubašić, M., Demirović, E.: Variable neighborhood search for google machine reassignment problem. Electron. Notes Discrete Math. **39**, 209–216 (2012)
9. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975)
10. Kendall, G., Bai, R., Błazewicz, J., De Causmaecker, P., Gendreau, M., John, R., Li, J., McCollum, B., Pesch, E., Qu, R., et al.: Good laboratory practice for optimization research. J. Oper. Res. Soc. (2015)
11. Kirkpatrick, S., Daniel Gelatt, C., Vecchi, M.P., et al.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
12. Krasnogor, N., Smith, J.: A memetic algorithm with self-adaptive local search: tsp as a case study. In: GECCO, pp. 987–994 (2000)
13. Lopes, R., Morais, V.W.C., Noronha, T.F., Souza, V.A.A.: Heuristics and matheuristics for a real-life machine reassignment problem. Int. Trans. Oper. Res. **22**(1), 77–95 (2015)
14. Masson, R., Vidal, T., Michallet, J., Penna, P.H.V., Petrucci, V., Subramanian, A., Dubedout, H.: An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. Expert Syst. Appl. **40**(13), 5266–5275 (2013)
15. Mehta, D., O'Sullivan, B., Simonis, H.: Comparing solution methods for the machine reassignment problem. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 782–797. Springer, Heidelberg (2012)
16. Moscato, P., et al.: On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech concurrent computation program, C3P Report, 826:1989 (1989)
17. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: a literature review. Swarm Evol. Comput. **2**, 1–14 (2012)
18. Neri, F., Tirronen, V., Karkkainen, T., Rossi, T.: Fitness diversity based adaptation in multimeme algorithms: a comparative study. In: IEEE Congress on Evolutionary Computation, CEC 2007, pp. 2374–2381. IEEE (2007)
19. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Hybrid evolutionary computation methods for quay crane scheduling problems. Comput. Oper. Res. **40**(8), 2083–2093 (2013)
20. Ritt, M.R.P.: An Algorithmic Study of the Machine Reassignment Problem. Ph.D. thesis, Universidade Federal do Rio Grande do Sul (2012)
21. Sabar, N.R., Ayob, M.: Examination timetabling using scatter search hyper-heuristic. In: 2nd Conference on Data Mining and Optimization, DMO 2009, pp. 127–131. IEEE (2009)
22. Sabar, N.R., Ayob, M., Kendall, G., Qu, R.: A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. IEEE Trans. Cybern. **45**(2), 217–228 (2015)
23. Sabar, N.R., Song, A.: Dual population genetic algorithm for the cardinality constrained portfolio selection problem. In: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (eds.) SEAL 2014. LNCS, vol. 8886, pp. 703–712. Springer, Heidelberg (2014)

24. Sabar, N.R., Zhang, X.J., Song, A.: A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 830–837. IEEE (2015)
25. Sabar, N.R., Ayob, M., Kendall, G., Qu, R.: Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. IEEE Trans. Evol. Comput. **19**(3), 309–325 (2015)
26. Talbi, E.-G.: Metaheuristics: From Design to Implementation, vol. 74. John Wiley and Sons, Hoboken (2009)
27. Xie, J., Mei, Y., Song, A.: Evolving self-adaptive tabu search algorithm for storage location assignment problems. In: Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference, pp. 779–780. ACM (2015)