CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	A Large Neighborhood Search approach for the Machine Reassignment Problem in data centers
Authors	Souza, Filipe;Grimes, Diarmuid;O'Sullivan, Barry
Publication date	2022-12-08
Original Citation	Souza, F., Grimes, D. and O'Sullivan, B. (2023) 'A large neighborhood search approach for the data centre machine reassignment problem', AICS2022, in L. Longo and R. O'Reilly (eds) Artificial Intelligence and Cognitive Science. Cham: Springer Nature Switzerland, pp. 397–408. https://doi.org/10.1007/978-3-031-26438-2_3
Type of publication	Conference item
Link to publisher's version	10.1007/978-3-031-26438-2_31
Rights	© 2023 The Author(s). Open Access. This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made - https://creativecommons.org/licenses/by/4.0/
Download date	2026-10-31 12:34:04
Item downloaded from	https://hdl.handle.net/10468/14038



University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

A Large Neighborhood Search Approach for the Data Centre Machine Reassignment Problem*

Filipe Souza[1,2], Diarmuid Grimes[2,3], and Barry O'Sullivan[1,2]

[1] Insight SFI Research Centre for Data Analytics, University College Cork, Ireland
f.desouza@cs.ucc.ie
http://www.ucc.ie/
[2] SFI Centre for Research Training in Artificial Intelligence
http://www.crt-ai.cs.ucc.ie
[3] Munster Technological University, Ireland
http://www.mtu.ie/

Abstract. One of the main challenges in data centre operations involves optimally reassigning running processes to servers in a dynamic setting such that operational performance is improved. In 2012, Google proposed the Machine Reassignment Problem in collaboration with the ROADEF/Euro challenge. A number of complex instances were generated for evaluating the submissions. This work focuses on new approaches to solve this problem.
In particular, we propose a Large Neighbourhood Search approach with a novel, domain-specific heuristic for neighborhood selection. This heuristic uses the unbalanced resource usage on the machines to select the most promising processes in each iteration. Furthermore, we compare two search strategies to optimise the sub-problems. The first one is based on the concept of Limited Discrepancy Search, albeit tailored to large scale problems; and the second approach involves the standard combination of constraint programming with random restart strategies.
An empirical evaluation on the widely studied instances from ROADEF 2012 demonstrates the effectiveness of our approach against the state-of-the-art, with new upper bounds found for three instances.

Keywords: LNS · Neighbourhood Selection · Machine Reassignment Problem · Limited Discrepancy Search.

1 Introduction

There has been a significant increase in data centers over the past two decades. Today there are nearly 3000 in the US alone and over 70 in Ireland[4]. With the increase in streaming services, and the default data acquirement of most websites,

* Supported by SFI Centre for Research Training in Artificial Intelligence under Grant No. 18/CRT/6223 and SFI under Grant No. 12/RC/2289-P2, co-funded under the European Regional Development Fund.

[4] https://www.statista.com/topics/6165/data-centers

etc. this is only expected to increase. In the research community, much research has naturally focused on reducing the environmental impact of data centers, with less focus on improving operational performance within data centers.

In 2012 Google proposed a challenge with this latter issue of operation performance in mind, via the ROADEF/Euro Challenge 2012 (*Roadef12*). The general goal is the optimisation of a data centre environment for virtualisation and service configuration. In particular, they proposed the Machine Reassignment Problem (MRP), which aims to reallocate a set of processes to a set of machines in order to minimise a multi-objective function subject to a number of constraints. During the competition a range of optimisation approaches were proposed and they are summarised by Afsar et al. in [9]. Due to its complexity and specificity, the MRP has been the focus of many works in the literature in the decade since, as discussed by Canales et al. [2].

In this paper we present a Large Neighbourhood Search (LNS) with some specific components to solve the MRP. The main contributions of this work are: (i) a novel domain specific neighbourhood selection heuristic; and (ii) a novel search strategy to optimise each LNS sub-problem. With regard to the search strategy, it relies on the known issue that a heuristic decision is more likely to be wrong in the beginning of the search because it has less information than deeper in search.

Bringing this to the MRP problem, when we assign the first process to the best machine based on the heuristic, perhaps this machine is the best machine at this moment because other processes are not assigned yet. While the last process to be assigned is more likely to be assigned correctly because the heuristic has the information of all processes already assigned. As the backtracking algorithm uses depth first search, it spends a long search time relying on the first decision taken by the heuristic. To overcome this issue, we investigate two search strategies that do not spend a large amount of time investigating alternatives values for heuristics decisions that are more likely to be correct.

2 Related Work

Large neighborhood search was first proposed by Shaw in 1998 [13] as a means of applying constraint programming (CP) techniques to large vehicle routing problems. In its basic form, an initial solution is generated and then refined in successive iterations. Each iteration involves firstly selecting a subset of variables (the neighborhood), whose assignment is relaxed while all other variables have their assignment fixed to the value in the current solution. The neighborhood of unassigned variables can then be solved using a systematic approach, like CP or MIP, to find the optimal solution to the neighborhood given the assignment of the non-neighborhood variables.

2.1 Machine Reassignment Problem

The Machine Reassignment Problem has received considerable attention in the literature, in particular the problem as defined by Google in Roadef12. Given

an assignment of processes to machines, the problem involves reassigning the processes to minimise a multi-objective cost function related to the migration of processes being reassigned. While some recent works (e.g [11]) have tackled the problem as a multiobjective optimisation problem, we consider the problem in its classical format as defined by Google. Here, the cost function is converted to a single objective function using a weighted sum of costs.

The costs are associated with: the resource load above safety capacity on machines including transient resource usage, where a process uses resources on both the original machine and the machine of its reassignment; the balance of resource usage on machines; and costs associated with migrating processes of services between machine pairs. This problem is further subject to constraints such as capacity of machines, relationships between process subsets and machine subsets, etc.

A recent study [2] shows that most current state-of-the-art approaches apply some variation of local search techniques to address the MRP. They also observed the superiority of the approaches that adapt the search strategy to the characteristics of each instance.

One of the most effective local search approaches to address the MRP came from Gavranovic and Buljubasic in [3]. An important component of the technique was a noising method to help avoid local optima. When the algorithm got stuck in a local minima, the weight of one of the objectives in the multiobjective cost function was changed. The search can then escape the local optima as it was specific to the previous objective function. When a new local optimal is reached, the approach returns to the original objective function to escape. This approach was the winner of Roadef12.

A number of LNS approaches for the MRP have been proposed, which are of particular relevance to this work. Indeed the second place entry was a CP-based LNS [8]. Mehta et al. investigated both a CP-based LNS and a MIP-based LNS for the problem, and found the CP-based LNS approach significantly more effective, particularly on the large-scale instances. This LNS approach does however have a number of parameters that are highly sensitive to the problem instance characteristics.

An improved method was proposed in subsequent work [7] to counter the issue of parameter sensitivity, using a non-model based portfolio approach (ISAC [6]) to tune the LNS parameters to clusters of similar instances. More recently another LNS approach was proposed by Brandt et al. [1] where four domain specific neighbourhood selection heuristics were evaluated, and only small sub-problems (less than 10 processes) were considered. However, this did not achieve the same level of performance as that of Mehta et al which is the state of the art LNS approach.

In terms of overall state-of-the-art, Turky [14] recently proposed two bi-level hyper-heuristic approaches, the first involving local search and the second involving ant colony optimization. The results presented demonstrated that this method was able to outperform most approaches on the Roadef12 instances, achieving a number of new upper bounds for instances.

3 Problem Definition

The Machine Reassignment Problem aims to reallocate a set of processes to a set of machines in order to minimise a multi-objective function subject to a number of constraints. We describe the problem as follows (due to space limitations, the reader is referred to the original problem specification [5] for more details).

We have a set of *machines* $m \in M$ that can be assigned processes. Each machine has strict capacity restrictions $C_{m,r}$ on its different resources ($r \in R$), e.g. CPU, Load, Disk. Machines also have associated safety capacities $SC_{m,r}$ per resource that can be exceeded but incur a penalty in so doing. Machines are further grouped into *locations* ($l \in L$), which are disjoint sets of machines. Similarly machines are grouped into *neighborhoods* ($n \in N$). Locations and neighborhoods handle different requirements that will be described subsequently.

Running on the machines are a set of *processes* ($p \in P$). Each process has associated resource requirements ($RE_{p,r}$). Processes are grouped into *services* ($s \in S$). Two processes of the same service cannot be assigned to the same machine. Processes of the same services have to be spread across machines in a minimum number of locations ($spreadMin_s$). Furthermore, services can have dependencies, if service s_1 depends on service s_2 then processes of s_1 must be assigned to machines in the *neighborhood* of machines handling processes of s_2.

Unbalanced resource usage is penalised according to balance rules $b \in B$. b is characterized by $(b_{r1}, b_{r2}, b_{target})$, the two related resources and the acceptable imbalance between them. A solution A is an assignment of $\forall p \in P$ to a machine $m \in M$. We will formally represent the assignment by $MC_p = m$ for each process p, while the original machine assignment is denoted by MO_p.

The MRP multiobjective function to minimise involves five different costs. The first cost is the load cost, the usage of machine resources above the safety capacity. The second cost is the balance cost, i.e. the imbalance of resources usage in each machine.

The final three costs are related to the cost of migrating a process from its original machine MO_p to a new machine MC_p. The first of these is the process move cost, where each process has an associated fixed cost to deter moving it. The second migration cost is the service move cost, which aims to penalise solutions that don't have moves balanced across services. The final cost is the machine move cost, which has a penalty depending on the pair of machines involved in a move.

The MRP problem can thus be modeled as:

- $LC = \sum_{r \in R} weightLC_r * (\sum_{m \in M} \max(0, U_{m,r} - SC_{m,r}))$
- $BC = \sum_{b \in B} weightBC_b * (\sum_{m \in M} \max(0, b_{target} * (C_{m,b_{r1}} - U_{m,b_{r1}}) - (C_{m,b_{r2}} - U_{m,b_{r2}})))$
- $PrMC = weight_{PMC} * (\sum_{p \in P \ \wedge \ MO_p \neq MC_p} PMC_p)$
- $SMC = weight_{SMC} * \max_{s \in S} \sum_{p \in s \ \wedge \ MO_p \neq MC_p} 1$
- $MaMC = weight_{MMC} * \sum_{p \in P} MMC_{MO_p, MC_p}$

[5] https://www.roadef.org/challenge/2012/files/problem_definition_v1.pdf

$$\text{minimize } \sum LC + BC + PrMC + MaMC + SMC \tag{1}$$

$$\text{subject to: } MC_p \in M \qquad \forall p \in P \tag{2}$$

$$U_{m,r} + TU_{m,r} \leq C_{m,r} \qquad \forall m \in M, \forall r \in R \tag{3}$$

$$MC_p \neq MC_j \qquad \forall p, j \in s, \forall s \in S, p \neq j \tag{4}$$

$$\sum_{l \in L} \min(1, \sum_{p \in S \ \wedge \ MC_p \in l} 1) \geq spreadMin_s \qquad \forall s \in S \tag{5}$$

$$\min(1, \sum_{p1 \in s1 \ \wedge \ MC_p 1 \in n} 1) \leq \min(1, \sum_{p2 \in s2 \ \wedge \ MC_p 2 \in n} 1) \quad \forall n \in N, \ s1 \ depends \ s2 \tag{6}$$

Constraint 2 enforces that each process is assigned to a machine. Constraint 3 ensures that the machine resources are not overloaded, where resource usage of resource r on machine m is given by $U_{m,r} = \sum_{p \in P \ \wedge \ MC_p = m} re_{p,r}$ and transient resource usage is defined by $TU_{m,r} = \sum_{p \in P \ \wedge \ MC_p \neq m \ \wedge \ MO_p = m} re_{p,r}$. Constraint 4 establishes that processes from the same service cannot be assigned for the same machine. Constraint 5 defines that the set of processes from a service have to be assigned to machines in a minimum number of different locations. Constraint 6 assures that if service s_1 has a dependency of service s_2, each process in s_1 has to be assigned for machines in the same neighbourhood of those process of s_2.

4 Proposed Algorithm

We implemented a Large Neighbourhood Search (LNS) for the MRP and compared two approaches for optimising the sub-problems within a CP solver. The first method is a random restart strategy (*RRS*) where search is restarted based on a failure threshold. We added a stochastic component to the variable/value ordering heuristics by choosing randomly amongst the top x choices of the heuristic. Therefore each restart is likely to explore a different part of the search space. This approach runs a backtracking search for a number of times, in each of them the failure threshold is increased based on $maxFails$. The termination criteria is a maximum failure threshold.

The second approach is a variation of Limited Discrepancy Search (LDS) [4], which we refer to as Restricted Domain Search (*RDS*). Note in our case we do not have the objective of *proving* optimality in each neighbourhood, our objective is to find the best possible solution for the neighbourhood in a small execution time. Thus, the idea behind this approach is to equally investigate every variable in the sub-problem.

Algorithm 1 describes the proposed RDS approach. It is a recursive function that, in each call, selects one process and investigates the D best machines to

assign the selected process to. When all processes are assigned, it checks whether the current solution is better than the best solution so far. If so, it updates the best solution with the current solution. Note this is based on a similar logic to LDS but rather than return to the top of the search tree at each increase of deviation, RDS performs its deviations in depth first search. The reason for this difference with LDS is the cost of propagation of assignments at the top of the search tree for problems of this nature.

Algorithm 1: RestrictedDomainSearch()

```
if qttUnassignedProcesses == 0 then
    if oldObjectiveCost > currentObjectiveCost then
        solutionSubProblem ← saveSolution() ;
    end
else
    process ← selectAndRemoveProcess() ;
    UnassignedProcesses − − ;
    updateDomain(process) ;
    if domain(process) == 0 then
        Failures + + ;
    else
        discrepancy ← 0 ;
        while domain(process) > 0 & checkTime() & discrepancy <=
         MaxDiscrepancy & qttFailures <= MaxFailures do
            machine ← selectAndRemoveMachine(process) ;
            if isConsistent(process, machine) then
                assignProcessToMachine(process, machine) ;
                propagateConstraints(machine) ;
                RestrictedDomainSearch() ;
                discrepancy + + ;
                unassignProcess(process, machine) ;
            else
                Failures + + ;
            end
        end
    end
    UnassignedProcesses + + ;
end
```

4.1 Adaptive Neighbourhood Size

An important parameter of Large Neighbourhood Search is the size of each neighbourhood. A very large neighbourhood requires too much time to be optimised which results in a poor investigation of other neighbourhoods. On the other hand too small a neighbourhood can result in getting stuck in local minima.

There are many approaches in the literature that address this problem by implementing an adaptive large neighborhood search, e.g. [10, 12, 5]. In this work a simple adaptive neighbourhood size method is used primarily for escaping local minima. A relatively small initial neighbourhood size is used until search reaches a local minimum, whereupon the neighbourhood size is increased until it leaves the local minimum, and the original neighbourhood size is then restored.

4.2 Neighbourhood Selection:

To define which variables should be relaxed on each iteration, our LNS approach focuses on the unbalanced usage of resources in each machine. The idea is that if a machine has an unbalanced usage, where the proportion of capacity used by one resource differs greatly from other resources, it is likely a better solution can be achieved by reassigning processes of this machine.

However, this heuristic does not consider all components of the multi-objective function. Therefore, we alternated with a heuristic based on the maximum machine cost. Algorithms 2 and 3 show the process to create the sub-problems.

Algorithm 2: Create subProblem Unbalanced Machine

```
numMachine ← (random()%(subProblemSize/2)) + 1 ;
machine ← getUnbalancedMachine() ;
while unassignedProcessQtt < numProcesses do
    if (numProcesses >= (subProblemSize/numMachine)) then
        numProcesses ← 0 ;
        machine ← getUnbalancedMachine() ;
    end
    if isHeuristicUsed then
        process ← getMaxCostProcess(machine) ;
    else
        process ← randomProcess(machine) ;
    end
    unassignProcess(process) ;
    addToSubProblem(process) ;
end
```

4.3 Variable And Value Ordering Heuristic

To select the variable we used the well known Fail First heuristic, that simply orders the variable based on minimising domain size. This heuristic is robust and widely used in CP solvers. Furthermore, this heuristic also indirectly incorporates the knowledge of "Big Processes First" highlighted by the winner of Roadef12 [3], larger processes are harder to assign to machines due to the capacity constraint, therefore the domain of possible machines for these processes is smaller.

For the value ordering heuristic we implemented an approximation of the minimum cost of assigning the process to a machine. This heuristic has a high

accuracy at the bottom of the search tree but loses accuracy as we approach the root. To alleviate this issue towards the root, ties were broken based on the number of remaining unassigned processes that could be placed on the machine.

Algorithm 3: getUnbalancedMachine()

```
if machineIndicesSize == 0  or  solutionWasImproved then
    solutionWasImproved ← FALSE ;
    machineIndicesSize = machineIndices.size() ;
    if useFirstSort then
        useFirstSort ← FALSE ;
        machineIndices ← machinesSortedByResourcesAvaliable() ;
    else
        useFirstSort ← TRUE ;
        machineIndices ← machinesSortedByMachineCost() ;
    end
end
if isHeuristicUsed then
    isHeuristicUsed ← FALSE ;
    machineIndicesSize ← machineIndicesSize − 1 ;
    machine ← machineIndices[machineIndicesSize] ;
else
    isHeuristicUsed ← TRUE ;
    machine ← randomMachine() ;
end
```

4.4 Early Search Noise Strategy

The double use of transient resources when migrating a process is an important aspect that must be considered in this problem. On some instances it can result in local optimal solutions that are very difficult to escape, as many machines will be overloaded with double use of transient resources. To avoid a premature convergence for some of those intermediate solutions, we added an extra component to the objective function to discourage process moves with a high level of transient resource usage. The weight of this component is reduced each time the algorithm reaches a fixed threshold of iterations without improvement, and set to 0 on the last 20% of the search runtime.

5 Evaluation

The experiments were run on a Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-70-generic) with 16 Core and 32GB. All runs had a runtime cutoff of 5 minutes per instance. Furthermore, as the proposed approach has stochastic components, the presented results are the average of 5 runs with different seeds. Table 1 presents the parameter configurations that were used to run the experiments.

Table 1: Configurations parameters for the benchmark experiment.

Parameter	Value
Runtime	300 seconds
Initial Neighbourhood Size	10 processes
Threshold of Non-improvement	50 iterations
Failure Threshold	400
Limit of Deviation	2

The experiments used the three sets of instances from Roadef12, where each set has 10 instances. The 'A' instance set is composed of smaller instances with a maximum of 1,000 processes and 100 Machines. The other two sets of instances are more complex and larger, with up to 50,000 processes and 5,000 machines.

5.1 Results

We first investigated the neighborhood search method, comparing Restricted Domain Search (RDS) with Random Restarted Search (RRS) on a range of neighbourhoods across the 30 instances. We observed that on a total of 271,219 neighbourhoods where one of these approaches managed to find a better (and improving) solution, RDS found better solutions in 54% of the neighbourhoods while RRS found better solutions in 46% of the neighbourhoods.

Furthermore, as we can see in Figure 1, RDS is consistently faster than RRS across all Roadef12 instances when the same neighborhoods were explored. This behaviour can be explained by the fact that every time the search is restarted in RRS, the algorithm has to assign many variables and propagate constraints before the search starts to have complete solutions or failures, which considerably increases the run-time when compared with RDS. We finally tested each search method independently (5 runs, 5 mins per run per instance) and found that RRS, while performing relatively well, never found better solutions on an instance when compared to RDS. These results demonstrate the quality of our novel RDS approach.

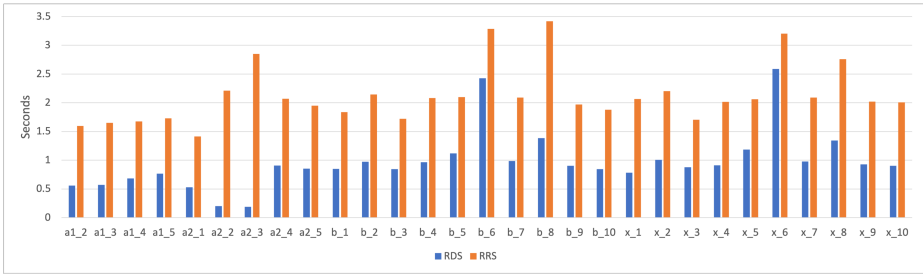


Fig. 1: Comparing the average Run-time of RDS and RRS in the same neighbourhoods.

Table 2: **Average cost** results and % gap to best known solution for RDS-LNS compared with state of the art. Best known solution taken from [14, 2]. All approaches had a 5 minute cutoff per run. CP-LNS and RDS-LNS run on same machine, results are average of 5 runs. NLS and Ant-HH results taken from paper, former is average of 100 runs, latter is average of 31 runs. Bold indicates best amongst the four comparison approaches according to the given metric.

		Cost				% Gap = 100* (Cost - BK)/BK			
Instances	BK Cost	Ant-HH$_{[14]}$	NLS$_{[3]}$	CP-LNS$_{[8]}$	RDS-LNS	Ant-HH	NLS	CPLNS	RDS
a1_1	44306501	**44306501**	**44306501**	**44306501**	**44306501**	**0.0**	**0.0**	**0.0**	**0.0**
a1_2	777532177	**777532179**	778142261	778654913	777680794	**0.0**	0.0	0.0	0.0
a1_3	583005715	**583005715**	583006320	583005829	583005836	**0.0**	0.0	0.0	0.0
a1_4	244875200	**244875200**	259815285	254185892	249753518	**0.0**	6.0	4.0	2.0
a1_5	727578306	**727578306**	727578311	727578311	727578311	**0.0**	0.0	0.0	0.0
a2_1	161	165	333	201	**159**	2.0	107.0	25.0	**-1.0**
a2_2	720671511	**720671511**	740140535	803912789	765776193	**0.0**	3.0	12.0	6.0
a2_3	1182260491	**1190713410**	1210207120	1304726522	1245045531	**1.0**	2.0	10.0	5.0
a2_4	1680368560	**1680368560**	1680629156	1683592281	1683447864	**0.0**	0.0	0.0	0.0
a2_5	307150821	**307150821**	317804454	339471433	362894481	**0.0**	3.0	11.0	18.0
b_1	3291069365	**3291069365**	3343410128	3339134760	3352472777	**0.0**	2.0	1.0	2.0
b_2	1010949451	**1015482891**	1015561513	1024629389	1028734683	**0.0**	0.0	1.0	2.0
b_3	156519816	**156691279**	157737166	157512118	159219336	**0.0**	1.0	1.0	2.0
b_4	4677792536	**4677792536**	4677981438	4677833576	4677858641	**0.0**	0.0	0.0	0.0
b_5	922944510	**922944510**	923905512	923721068	930511569	**0.0**	0.0	0.0	1.0
b_6	9525851389	**9525851389**	9525934654	9525870196	9525853048	**0.0**	0.0	0.0	0.0
b_7	14834456020	**14834456193**	14835328102	14853146933	14878044937	**0.0**	0.0	0.0	0.0
b_8	1214291129	**1214291141**	1214453127	1214589052	1214526543	**0.0**	0.0	0.0	0.0
b_9	15885437252	**15885437252**	15885693227	15885768231	15885669118	**0.0**	0.0	0.0	0.0
b_10	18048187105	**18048187105**	18048711483	18069108773	18130155888	**0.0**	0.0	0.0	0.0
x_1	3030246091	**3044411001**	3065081130	3106902032	3102713793	**0.0**	1.0	3.0	2.0
x_2	1002379317	**1002379317**	1003356104	1015807185	1010454229	**0.0**	0.0	1.0	1.0
x_3	69970	**75155**	341508	811958	136036	**7.0**	388.0	1060.0	94.0
x_4	4721586142	**4721586142**	4721856521	4721635533	4721697878	**0.0**	0.0	0.0	0.0
x_5	54132	57974	160418	100449	**53899**	7.0	196.0	86.0	**-0.0**
x_6	9546936159	**9546936159**	9546972261	9546952069	9546938475	**0.0**	0.0	0.0	0.0
x_7	14252476500	**14252476500**	14253212517	14397486190	14349370657	**0.0**	0.0	1.0	1.0
x_8	29193	32014	147269	65953	**30678**	10.0	404.0	126.0	**5.0**
x_9	16125531142	**16125531142**	16125760293	16125916363	16125823486	**0.0**	0.0	0.0	0.0
x_10	17815045320	17815981156	**17815072367**	17839540583	17903620248	0.0	**0.0**	0.0	0.0

The results of RDS-LNS is given, along with comparison results from other approaches in the literature, in Table 2. In particular we present results for the best LNS approach from the literature (CP-LNS [8]), the winner of Roadef12 (NLS [3]), and the current state of the art (Ant-HH [14]). We also provide the reference best known solution (*BK_Cost*) for each instance.

For the CP-LNS approach, we ran their code with the same experimental setup on the same machine as our experiments. The results for the other two methods were taken from their respective references. Both used a 5 minute runtime cutoff but NLS is an average of 100 runs, while Ant-HH is an average of 31 runs.

The results for each method on each instance are given in terms of average objective value across runs, and in terms of gap to the best known solution. Looking at the gap, the first point to note is that RDS-LNS was within 2% of the best known solution in its average performance on 25/30 instances, and less than 1% for 17 of these.

Comparing our approach with the other methods in the table, we first consider the other LNS approach. This is most directly comparable both because it's a CP-based LNS approach, and because experiments were performed under identical conditions. RDS-LNS found better results than CP-LNS in 17/30 instances, and had identical averages for two others. Of the remaining 11 instances that CP-LNS had better results, RDS-LNS was within 1% for all but one (instance a2_5). Results were also impressive in comparison Roadef12 winner NLS[3], our approach outperformed their approach on 13/30 instances, while for the majority of the other 17 instances the RDS-LNS results are within 2%. Finally, comparing against the current SOA, Ant-HH[14], RDS-LNS only managed to perform better on 3/30 instances. However, RDS-LNS was within 1% for all but nine instances, and within 10% for all but two of those nine.

Even more importantly, despite the amount of research that has been dedicated to these instances in the past decade, upon further analysis of our results we found that RDS-LNS improved on the best known solution [14, 2] for three instances (a2_1, x_5 and x_8). Indeed, as can be seen in Table reftable:Result, the *average* performance alone was better than the best known. Based on the best solution among our 5 runs, the new upper bounds found were 153, 45922, and 28564 respectively (equating to roughly 5%, 15%, and 2% reductions in cost compared to the current best known cost). This is even more noteworthy given that some of the other approaches in the literature had a 30 minute cutoff [1], or had many more runs, e.g. 31 [14] and 100 [3] and did not find such best solutions for these three instances.

6 Conclusion

In this paper, we proposed a new Large Neighbourhood Search approach for the Machine Reassignment Problem with a novel domain specific neighbourhood operator, and novel search strategy for the subproblems. Our empirical evaluation demonstrated the quality of the approach on a well studied problem set, resulting in improvement on best known solution for three of the thirty instances. Comparison of our results against other state-of-the-art demonstrated the effectiveness of RDS-LNS. Further analysis of the sub-problem optimisation strategies showed the superiority of the proposed Restricted Domain Search when compared with a standard random restart strategy.

References

1. Brandt, F., Speck, J., Völker, M.: Constraint-based large neighborhood search for machine reassignment. Annals of Operations Research **242**(1), 63–91 (2016)
2. Canales, D., Rojas-Morales, N., Riff, M.C.: A survey and a classification of recent approaches to solve the google machine reassignment problem. IEEE Access **8**, 88815–88829 (2020)
3. Gavranović, H., Buljubašić, M.: An efficient local search with noising strategy for google machine reassignment problem. Annals of Operations Research **242**(1), 19–31 (2016)
4. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: IJCAI (1). pp. 607–615 (1995)
5. He, K., Tole, K., Ni, F., Yuan, Y., Liao, L.: Adaptive large neighborhood search for circle bin packing problem. arXiv preprint arXiv:2001.07709 (2020)
6. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC—instance-specific algorithm configuration. In: ECAI 2010, pp. 751–756. IOS Press (2010)
7. Malitsky, Y., Mehta, D., O'Sullivan, B., Simonis, H.: Tuning parameters of large neighborhood search for the machine reassignment problem. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 176–192. Springer (2013)
8. Mehta, D., O'Sullivan, B., Simonis, H.: Comparing solution methods for the machine reassignment problem. In: International Conference on Principles and Practice of Constraint Programming. pp. 782–797. Springer (2012)
9. Murat Afsar, H., Artigues, C., Bourreau, E., Kedad-Sidhoum, S.: Machine reassignment problem: the roadef/euro challenge 2012 (2016)
10. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4), 455–472 (2006)
11. Saber, T., Gandibleux, X., O'Neill, M., Murphy, L., Ventresque, A.: A comparative study of multi-objective machine reassignment algorithms for data centres. Journal of Heuristics **26**(1), 119–150 (2020)
12. Sacramento, D., Pisinger, D., Ropke, S.: An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. Transportation Research Part C: Emerging Technologies **102**, 289–315 (2019)
13. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: International conference on principles and practice of constraint programming. pp. 417–431. Springer (1998)
14. Turky, A.: Bi-level hyper-heuristic approaches for combinatorial optimisation problems. Ph.D. thesis, RMIT University (2019)