Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Discrete Mathematics 39 (2012) 209–216

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

Variable Neighborhood Search for Google Machine Reassignment problem

Haris Gavranović [a,1] Mirsad Buljubašić [b,2] Emir Demirović [b,3]

[a] *International University Sarajevo, Bosnia and Herzegovina*

[b] *Faculty of Natural Sciences, University of Sarajevo, Bosnia and Herzegovina*

Abstract

We present a hybrid method to efficiently solve Google Machine Reassignment problem (MRP), the problem proposed at ROADEF/EURO Challenge 2012 competition. We study, implement, combine and empirically examine different local search neighborhoods to solve the set of available instances. Intensification and diversification of search is achieved through the suitable change of the objective function and sorting the processes. We present results obtained with the solver that respect the given computational time of 5 minutes. Some of the obtained results are proven to be optimal or near optimal. With the presented method we were ranked first at ROADEF/EURO Challenge 2012 competition.

Keywords: Machine Reassignment, Generalized Assignment Problem, Local Search, Variable Neighborhood Search

[1] Email: haris.gavranovic@gmail.com

[2] Email: mirsad_bulj@yahoo.com

[3] Email: emir.demirovic@gmail.com

1571-0653/$ – see front matter
doi:10.1016/j.endm.2012.10.028

1 Introduction

In this work we consider Google Machine Reassignment problem, proposed at ROADEF/EURO Challenge 2012 (http://challenge.roadef.org). The aim of this problem is to improve the usage of a set of machines. A machine has several resources as for example RAM and CPU, and runs processes which consume these resources. Initially each process is assigned to a machine. In order to improve machine usage, processes can be moved from one machine to another. Possible moves are limited by hard constraints, as for example resource capacity constraints, and have a cost. A solution to this problem is a new process-machine assignment which satisfies all hard constraints and minimizes a given objective cost. The problem at hand is similar to the Generalized Assignment Problem (GAP) [2] with some specific, and often hard to satisfy, constraints and some of the presented local moves (e.g. shift, swap) could be found for example in Yagiura et al. [5], [6]. Detailed description of the problem is given in the ROADEF/EURO Challenge subject [4], and we will use the same names for data as given in this document. The next section presents the simple yet important calculation of solution lower bounds. In the third section we present the components of local search and how they are composed in a general solution method. The fourth section presents computational results on 20 given instances. The paper is concluded with possible extensions of the work and refinement of the presented method. Note that this is a recently proposed problem and there are no previous works published on this topic, to our knowledge.

1.1 Notation

The notation used in the announcement of the problem can be summarized in the following list.

- M - set of machines
- P - set of processes
- R - set of resources
- B - set of balance triples
- $M(p)$ - machine process p is assigned to
- $M_0(p)$ - initial machine process p is assigned to
- $C(m, r)$ - the capacity of resource $r \in R$ for machine $m \in M$
- $SC(m, r)$ - the safety capacity of resource $r \in R$ for machine $m \in M$
- $R(p, r)$ - the requirement of resource $r \in R$ for process $p \in P$
- $U(m, r)$ - the usage U of a machine m for a resource r defined as:

$$U(m,r) = \sum_{p \in P, M(p)=m} R(p,r)$$

2 Lower Bound

In this section we present a lower bound on total solution cost. This bound is equal to the sum of load cost lower bound and balance cost lower bound.

2.1 Load Cost Lower Bound

Solution load cost is

$$LC = \sum_{r \in R} weight_{loadCost}(r) * loadCost(r),$$

We have:
$loadCost(r) = \sum_{m \in M} max(0, U(m,r) - SC(m,r)) \geq \sum_{m \in M} (U(m,r) - SC(m,r)) =$
$\sum_{m \in M} U(m,r) - \sum_{m \in M} SC(m,r) = U(r) - SC(r),$
where $U(r) = \sum_{m \in M} U(m,r)$ is total requirement for resource r and $SC(r) = \sum_{m \in M} SC(m,r)$ is total safety capacity. Total load cost lower bound is then equal to

$$\sum_{r \in R} weight_{loadCost}(r) * (U(r) - SC(r))$$

2.2 Balance Cost Lower Bound

In similar way we obtain a lower bound on solution balance cost. Solution balance cost is

$$BC = \sum_{b \in B} weight_{balanceCost}(b) * balanceCost(b),$$

We have:

$balanceCost(b) = \sum_{m \in M} max(0, target * A(m,r_1) - A(m,r_2)) \geq \sum_{m \in M} (target * A(m,r_1) - A(m,r_2)) = target * \sum_{m \in M} A(m,r_1) - \sum_{m \in M} A(m,r_2) = target * A(r_1) - A(r_2),$
where $A(m,r) = C(m,r) - U(m,r)$, $b = (r1, r2, target) \in B$ and $A(r) = \sum_{m \in M} A(m,r)$ is total available amount of resource r. Total balance cost lower bound is then equal to

$$\sum_{b \in B} weight_{balanceCost}(b) * (target * A(r_1) - A(r_2))$$

Solution cost lower bound is equal to the sum of these two lower bounds. Lower bounds for challenge instances are presented in Table 1 and Table 2.

3 Method Description

3.1 General framework

We propose a local search method, which given an initial assignment, iteratively tries to replace the current assignment with a better one. At every iteration, the current assignment has a smaller cost than the previous one and each assignment generated this way is feasible. Different local search neighborhoods are explored [3]. Local search starts from the given initial solution. Load cost is the most important part of the solution cost. Reassignment of small processes (respect to requirements), while improving objective function value for some, small, value, consume a certain amount of remaining safety capacities of machines. Thereafter, the reassignment of big processes will be much harder to accomplish. This is why the proposed method tries to reassign big processes first. All processes are sorted by absolute or relative requirements and reassignment is done in the following way:

- Set number of processes to consider to zero at beginning ($N = 0$).
- Increase number of processes to consider (N) and improve solution considering (reassigning) only processes from position 0 to N in sorted list of processes. ($local\_search(0, N)$) Repeat this until all processes are considered.

In what follows we describe more precisely a main procedure ($local\_search()$).

3.2 Neighborhoods

The local search procedure consists of exploring the following neighborhoods:

Shift - Given a solution s, the shift neighborhood, N_{shift} , is defined to be the set of solutions that can be obtained from s by reassigning one process from one machine to another. More formally,

$$N_{shift}(X) = \{X' : X' \text{ is obtained from } X \text{ by changing the assignment of one process }\}$$

Swap - The swap neighborhood, N_{swap}, is the set of solutions that can be obtained by interchanging the assignments of two processes, originally assigned

to different machines More formally,

$$N_{swap}(X) = \{X' : X' \text{ is obtained from } X \text{ by exchanging the assignments of two processes }\}$$

Chain - $N_{chain}(s)$ is the set of solutions s' obtainable from s by shifting l processes $p_1, p_2, ..., p_l$ ($l \in \{1, 2, 3, ..., |M|\}$) simultaneously, such that:

$$s'(p_k) = s(p_{k+1})\ k \in \{1, 2, 3, ..., l-1\}$$
$$s'(p_l) = s(p_0)$$

BPR - $N_{bpr}(s)$ is the set of solutions s' obtainable from s by shifting process p to a machine m and shifting some processes from m to some other machines. This neighborhood showed to be useful in reassigning big processes.

Shift and swap neighborhoods are the most simple and can be found in almost every paper concerning solving problems similar to MRP, for example Generalized Assignment Problem (GAP) and Multi Resource Generalized Assignment Problem (MRGAP).

Chain neighborhood is less frequent in the literature while for this problem, coupled with other local moves, shows its strength. Chain neighborhood is explored using an oriented graph G with weights on the arcs. Vertex of the graph G represents one process and no two vertices represent the processes assigned to the same machine in the current solution. Let vertex v_i correspond to the process p_i assigned to machine m_i. The weight on the arc (v_i, v_j) is the change in objective function with removing p_j from m_j and assigning p_i to m_j. It is obvious that chain (cycle or path) move is improving the objective function only if corresponding cycle (path) has a negative weight. Therefore, the goal is to find cycles or paths with negative weight. Number of vertices is 30-100 and the processes represented by these vertices are chosen randomly from a given range. Since keeping all possible (feasible) edges would be practically prohibitive and time-consuming we keep only negative edges in the graph. In that way every cycle (path) would improve the solution.

It is not always possible to reassign big processes using shift, swap or chain moves, especially if these processes are much bigger then the other ones. That's why we use BPR neighborhood. In shift, swap and chain moves only one new process can be assigned to a machine or removed from a machine, while in BPR neighborhood few processes are removed from a machine big process is to be assigned to.

The neighborhoods are explored in the following order: BPR, shift, swap, chain. Running time of the search and minimum solution improvement are used as a stopping criteria in exploring each of these neighborhoods. When stopping criteria is met for a current neighborhood, the search continues with exploring the next neighborhood. Each neighborhood is explored only once in each local search iteration (for given range of processes and current objective function).

3.3 Search Diversification

One of the strategies used for improving the solution quality is search diversification which is done by shaking the method. The strategy we use is called the Noising method. We refer the reader to [1]. Shaking is done by changing the objective function. Namely, we escape from the local optimum by changing the objective function in the following way:

- choose resource r
- increase load cost weight of the resource r

We optimize the new objective and then continue with optimization of original objective. This is repeated for few different resources, which are chosen by importance (distance to the load cost lower bound). The final algorithm is given in 1.

Algorithm 1 Local Search

```
make the list of processes sorted by sum of requirements (L)
NMB_ITERATIONS = 5;
for i = 1 to NMB_ITERATIONS do
    for r = 1 to NMB_RESOURCES_TO_USE do
        LS with original objective
        LS with changed objective (increase weight or resource r)
    end for
    increase number of processes to consider
end for
```

4 Computational Results

The method is tested on the computer equipped with Intel i7 920 processor (2.66 GHz, 8M Cache, RAM 6GB). In Tables 1 and 2 we present our computational results for provided set of instances.

Results A -100 runs

Instance	Average	Best	Best Q	LB
a1_1	44 306 501	44 306 501	44 306 501	44 306 390
a1_2	778 265 189	777 536 907	777 532 896	777 530 730
a1_3	583 006 320	583 005 818	583 005 717	583 005 700
a1_4	260 903 327	251 524 763	252 728 589	242 387 530
a1_5	727 578 312	727 578 310	727 578 309	727 578 290
a2_1	333	199	198	0
a2_2	748 528 290	720 671 548	816 523983	13 590 090
a2_3	1 218 013 414	1 190 713 414	1 306 868 761	521 441 700
a2_4	1 680 740 350	1 680 615 425	1 681 353 943	1 680 222 380
a2_5	317 804 454	309 714 522	336 170 182	307 035 180

Table 1

The table shows the results for data set A instances. Average and best objective values are reported with running the program for 100 different seeds with 5 minutes running time. Fourth column (Best Q) represents the best solutions from qualifying phase of competition and the last column represents solution lower bounds.

Results B - 100 runs

Instance	Average	Best	LB
B1	3 345 152 832	3 307 124 603	3 290 754 940
B2	1 015 561 513	1 015 517 386	1 015 153 860
B3	157 737 166	156 978 411	156 631 070
B4	4 677 981 438	4 677 961 007	4 677 767 120
B5	923 905 512	923 610 156	922 858 550
B6	9 525 934 654	9 525 900 218	9 525 841 820
B7	14 835 328 102	14 835 031 813	14 833 297 940
B8	1 214 510 885	1 214 416 705	1 214 153 440
B9	15 885 693 227	15 885 548 612	15 885 064 440
B10	18 048 711 483	18 048 499 616	18 048 006 980

Table 2

The table shows the results and lower bounds for data set B instances.

5 Conclusion

The proposed solution is still sensitive to the choice of the parameters and the appropriate choice of processes and machines participating in the moves. The computational tests show that these choices in the initial phase of the method greatly influence the quality of the final solution. Nevertheless, some of the obtained results are quasi optimal while the others are competitive with the world best known results. The challenge remains to construct essentially different type of local search moves. We believe it would be very useful to dispose an efficient procedure to calculate the optimal assignment of processes on two given machines, taking into consideration only the processes already assigned to those machines. While the whole problem is defined as an improvement problem for a given solution the construction of an initial solution from scratch would bring a new insight in the data and the method of solution and could improve the presented local search itself.

References

[1] Charon, I., and O. Hurdy, *The noising methods: A generalization of some metaheuristics*, EJOR **135** (2001), 86–101.

[2] Diaz, J. A., and E. Fernandez, *A Tabu search heuristic for the generalized assignment problem*, Technical Report DR. **98/8** (1998).

[3] Hansen, P., N. Mladenovic, and J. A. M. Perez, *Variable neighbourhood search: methods and applications*, Annals of Operations Research **175** (2010), 367–407.

[4] *Google ROADEF/EURO challenge 2011-2012: Machine reassignment*, URL: http://challenge.roadef.org/2012/files/problemdefinitionv1.pdf

[5] Yagiura, M., T. Ibaraki, and F. Glover, *An ejection chain approach for the generalized assignment problem*, Technical Report **99013** (1999).

[6] Yagiura, M., T. Yamaguchi, and T. Ibaraki, *A variable depth search algorithm with branching search for the generalized assignment problem*, Optimization Methods and Software **10** (1998), 419–441.