Metaheuristics for task assignment

Roberto Meroni* Hugo Sanz González†

November 2023

Consider the problem of assigning the workload required to complete a set of task T to a set of computers C. Each task $t \in T$ requires $r_t \geq 0$ units of work to complete, while each computer $c \in C$ has $r_c \geq 0$ compute units available. Define the load of a computer $c \in C$ as the total number of compute units handled divided by its capacity r_c, and let P_2 be the problem of minimizing the load of the highest-loaded computer. We evaluate the performance and efficiency of two search strategies based on the GRASP and BRKGA metaheuristics on instances of P_2 with different characteristics.

1. *Prepare a pseudocode for the GRASP constructive algorithm. Specify the greedy function and the RCL.*

The GRASP algorithm consists of two schemes that are applied alternately until a maximum number of iterations is reached. Let us first describe these two phases and then state the procedure in full.

Constructive phase. The first part of GRASP attempts to construct a feasible solution by traversing the set of tasks T by the number of required units in descending order. For each task, a restricted candidate list (RCL) containing the task–computer assignments that reduce the load of the highest-loaded computer by the greatest amount is built. Then the scheme chooses an assignment from the RCLat random. Without loss of generality, let $C = \{c_1, \ldots, c_m\}$ be the set of computers. The formal description of this procedure follows:

1. [Initialize.] Set $S \leftarrow \varnothing$, $i \leftarrow 1$; sort the tasks in T by their required compute units in descending order, and let the resulting sequence be $t_1, \ldots, t_n$. Then set $l_j \leftarrow 0$ for all $1 \leq j \leq m$. (Throughout the algorithm, S is the partial solution built so far, i is an index into the sequence of sorted tasks, and l_j is the number of compute units assigned to computer c_j.)

2. [Start the search.] If $i = n$, go to step 8. Otherwise set $F \leftarrow \varnothing$, $j \leftarrow 1$. (Here $F \subseteq C$ is the set of computers to which task t_i may be assigned while keeping the load of all computers under their respective capacities, and j is an index into the set of computers.)

3. [Obtain feasible assignments.] If $j = m$, jump to step 6. Otherwise if $l_j + r_{t_i} \leq r_{c_j}$, set $F \leftarrow F \cup \{c_j\}$. Next, set $j \leftarrow j + 1$ and repeat this step.

*roberto.meroni@estudiantat.upc.edu
†hugo.sanz@estudiantat.upc.edu

4. [Infeasible?] The set F is empty whenever no more tasks can be assigned; if so, go to step 8.

5. [Create RCL.] If F is empty, jump to step 8. Set $m_{c'} \leftarrow \max_{c \in C}(l_c + \mathbf{1}_{c=c'} r_{t_i})/r_c$ for each $c' \in F$. (Here $m_{c'}$ is the load of the highest loaded computer if the assignment (t_i, c') were added to S.) Set $m \leftarrow \min_{c \in F} m_c$, $M \leftarrow \max_{c \in f} m_c$ and $\beta \leftarrow m + \alpha(M - m)$. Then remove all items c from F such that if t_i were assigned to c in S then the load m_c of the highest-loaded computer would be greater than β.

6. [Select.] Choose a random computer c from F, and set $S \leftarrow S \cup \{(t_i, c)\}$, $l_c \leftarrow l_c + r_{t_i}$. Increase i by 1 and go to step 4.

7. [Done.] Terminate with S as the answer.

The quality function $q(t, c') = \max_{c \in C}(l_c + \mathbf{1}_{c=c'} r_t)/r_c$ in step 5, where $\mathbf{1}.$ is the indicator function, represents the load of the highest loaded computer if task t were assigned to computer c' on top of the partial solution S.

Local search. The second stage of GRASP refines the solution found during the constructive phase. Given a partial solution S, the program considers two kinds of neighborhoods:

- For each assignment $(t, c) \in S$, construct a new solution with exactly the same assignments as S but with task t assigned to a computer in C different from c with enough free resources.
- For each pair of assignments $(t_1, c_1), (t_2, c_2) \in S$, create a new solution with the same assignments as S except that (t_1, c_1) is replaced by (t_1, c_2) and (t_2, c_2) is replaced by (t_2, c_1) provided that the new assignments do not exceed the capacity of c_1 nor c_2.

Also, the program can be configured to select an element from the neighborhood of S in two ways: *a*) explore all near solutions until one with a better objective is found, or *b*) calculate the load of the highest-loaded computer for all solutions in the neighborhood and pick one that minimizes this quantity.

If $i_{\max}$ denotes the maximum number of local search iterations, then the local search scheme proceeds as follows:

1. [Construct initial solution.] Let S be the partial solution built in the constructive phase, let z be the load of the highest-loaded computer in S, and set $i \leftarrow 1$.

2. [Explore.] Let S' be a solution in the neighborhood of S such that the load z' of the highest-loaded computer in S' is less than z. If such an S' does not exist, go to step 3. Otherwise set $S \leftarrow S'$ and $z \leftarrow z'$.

3. [Done?] Increase i by 1; if $i \leq i_{\max}$ return to step 2. Otherwise terminate with S as the answer.

In step 2 the solution S' is picked according to one of the exploration strategies described above from one of the two kinds of neighborhoods.

We are now ready to state the functioning of the GRASP scheme in full:

#	Computers	Tasks
1	100	500
2	120	500
3	100	550

Table 1: Some characteristics of the generated instances.

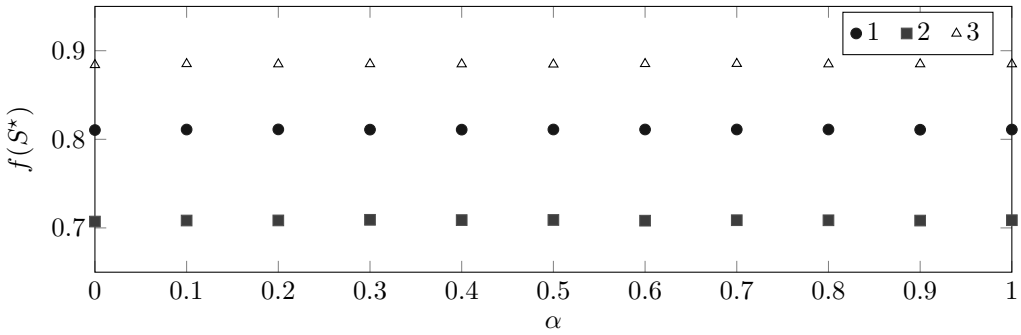

Figure 1: The average objective value of the solutions found by 5 trials of the GRASP algorithm with parameter α on the 3 generated instances summarized in Table 1.

ALGORITHM G (GRASP). Given a maximum number of iterations $k_{\max}$, this algorithm attempts to produce an assignment of computers to a subset of the given tasks that meets all restrictions on computer capacities while trying to minimize the load of the highest-loaded computer.

1. [Initialize.] Set $S^\star \leftarrow \varnothing$ and $k \leftarrow 1$.
2. Execute the constructive phase, and set S to be the built solution.
3. Update S with the refinement of S found by the local search procedure.
4. [Improvement?] If S is feasible and $f(S) < f(S^\star)$, set $S^\star \leftarrow S$.
5. [Done?] Increase k by 1. If $k \leq k_{\max}$ go to step 2. Otherwise terminate with $S^\star$ as the answer.

2. *Tune the α parameter.*

We use the instances generated during the previous lab session, whose main characteristics are summarized in Table 1, to optimize the RCL threshold parameter α. Next, we run the GRASP algorithm on these instances 5 times with a running time limit of 30 seconds. We use the average objective of the solutions found as the success metric; see Figure 1 for the results. Observe that the value of the parameter has little effect on the quality of the solutions obtained, so from now on we will take $\alpha = 0.3$.

3. *Prepare a pseudocode for the BRKGA decoder algorithm. Specify how the chromosome is used.*

A BRKGA decoder in the context of the computer-task assignment problem takes a $|T|$-dimensional vector $\mathfrak{c}$, called a *chromosome*, as input. Each entry of $\mathfrak{c}$ is a *gene*, and its value—also known as an *allele*—is a real number in $[0, 1]$.

The goal of the decoder is to convert this genetic material into a solution S to the optimization problem at hand and its corresponding objective value $f(S)$. Here each of the $|T|$ alleles in $\mathfrak{c}$ is used to weight the load of each task in T during the subsequent greedy search step in the BRKGA scheme. The following algorithm formalizes this notion of passing from the search space to the solution space and performing the greedy search:

1. [Initialize.] Set $S \leftarrow \varnothing$, $i \leftarrow 1$.

2. [Link chromosomes.] Set $r'_{t_i} \leftarrow \mathfrak{c}_i r_{t_i}$ for each task $t_i \in T$. (Here r'_{t_i} is the amount of compute required by task t_i, weighted by the corresponding allele of the input chromosome $\mathfrak{c}$.)

3. [Sort tasks.] Sort the tasks in T by their required weighted compute units in descending order, and let the resulting sequence be $t'_1, \ldots, t'_n$. Then set $l_j \leftarrow 0$ for all $1 \leq j \leq m$.

4. [Start the search.] If $i = n$, go to step 8. Otherwise set $F \leftarrow \varnothing$, $j \leftarrow 1$. (Here $F \subseteq C$ is the set of computers to which task t'_i may be assigned while keeping the load of all computers under their respective capacities, and j is an index into the set of computers.)

5. [Obtain feasible assignments.] If $j = m$, jump to step 6. Otherwise if $l_j + r_{t'_i} \leq r_{c_j}$, set $F \leftarrow F \cup \{c_j\}$. Next, set $j \leftarrow j + 1$ and repeat this step.

6. [Infeasible?] The set F is empty whenever no more tasks can be assigned; if so, go to step 8.

7. [Apply fittest assignment.] Let $c^\star$ be the computer $c' \in F$ that minimizes $\max_{c \in C}(l_c + \mathbf{1}_{c=c'} r_{t'_i})/r_c$. Then set $l_{c^\star} \leftarrow l_{c^\star} + r_{t'_i}$, $S \leftarrow S \cup \{(t'_i, c^\star)\}$, increase i by one and return to step 4.

8. [Done.] Terminate with S as the answer.

The BRKGA algorithm evaluates each weighted solution, and mixes the chromosomes of the best solutions found with the existing chromosomes. The resulting chromosomes have some genes in common with the best solutions found so far, so in principle this evolutionary process progresses towards a better incumbent.

4. *Study what is the best combination of BRKGA parameters (size of population, inheritance probability, elite set and mutant percentages).*

Let n be the number of tasks in T. We apply the BRKGA on the three instances described in Table 1 with elite individual proportions $e = 0.15, 0.2, 0.25$, mutant individual proportions $m = 0.05, 0.1, 0.15, 0.2$, inheritance probabilities $p_i = 0.65, 0.7$ and number of individuals $N = n, 3n/2, 2n$. The best results found appear in Table 2. The objective found was quite close for all choices of parameter in the three instances. In fact, the solution found by the algorithm for the third instance was the same for every choice of parameters; here we selected $(e, m, p_i, N) = (0.15, 0.15, 0.7, n)$ arbitrarily.

5. *Solve the same instances generated in the previous lab session using local search, GRASP (using only the constructive phase and doing local search), BRKGA and CPLEX. Plot the quality of the solutions and time to solve against the size of the instances.*

The objective value and the running time required to obtain the solution for all search algorithms appear in Figure 2; for the GRASP and BRKGA

Instance	e	m	p_i	N	Objective
1	0.15	0.15	0.7	n	0.687
2	0.15	0.10	0.7	n	0.539
3	0.15	0.15	0.7	n	0.790

Table 2: The BRKGA configurations that lead to the minimum objective found for the three instances described in Table 1.

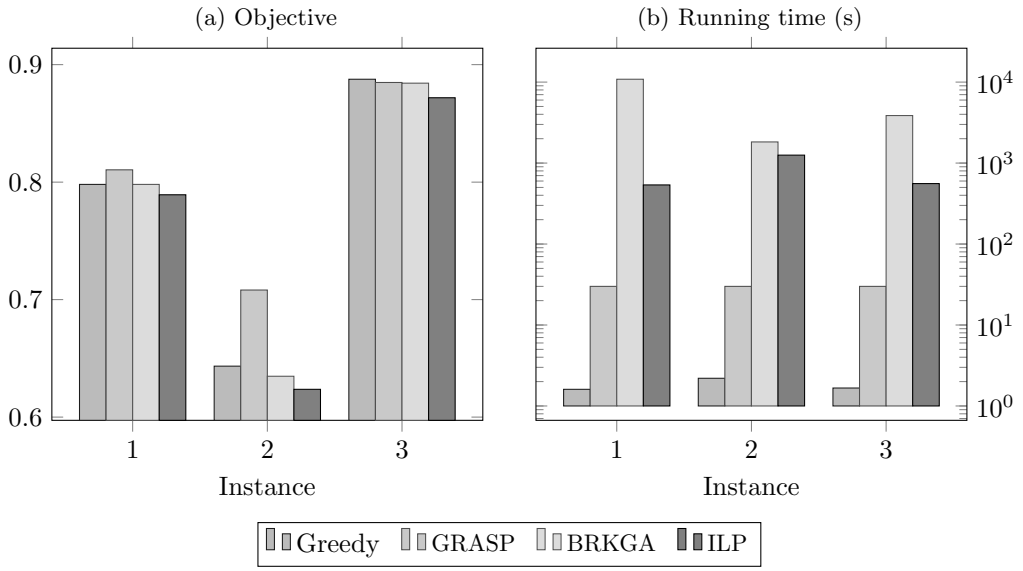

Figure 2: Solutions to the minimization problems generated in lab session 4 obtained by the search algorithms. The solution times appear in a logarithmic scale.

algorithms, we use the tuned parameters given in exercises 2 and 4. Note that greedy search without local search (as described in exercise 1) gives solutions of quality comparable to those found by BRKGA and CPLEX. Most importantly, it does so while requiring a fraction of the execution time. The GRASP algorithm gives worse solutions than all other algorithms for the first and second instances, and its running time is of the same order of magnitude as that of BRKGA and CPLEX. The BRKGA procedure does slightly better, but its solutions and running time are worse than those of CPLEX under the ILP formulation.