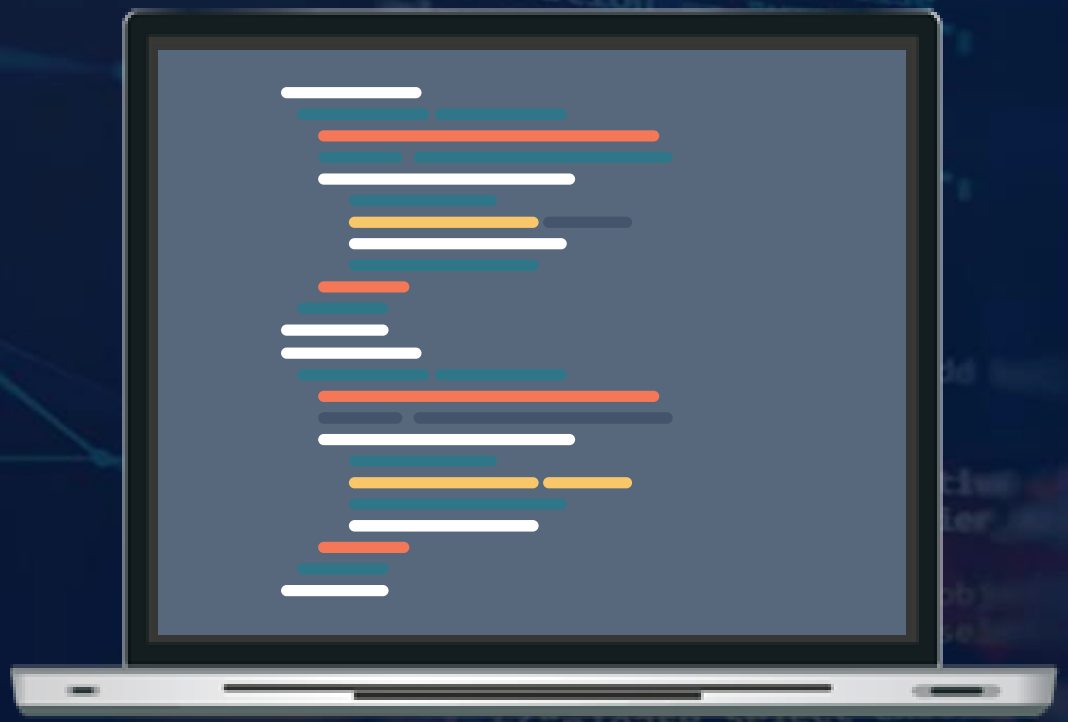
TABU SEARCH

BROGIOLO, Francisco / DE CASA, Davide / FORNERO, Lara Victoria

ALGORITHM AND FLOW CHART

TABU SEARCH



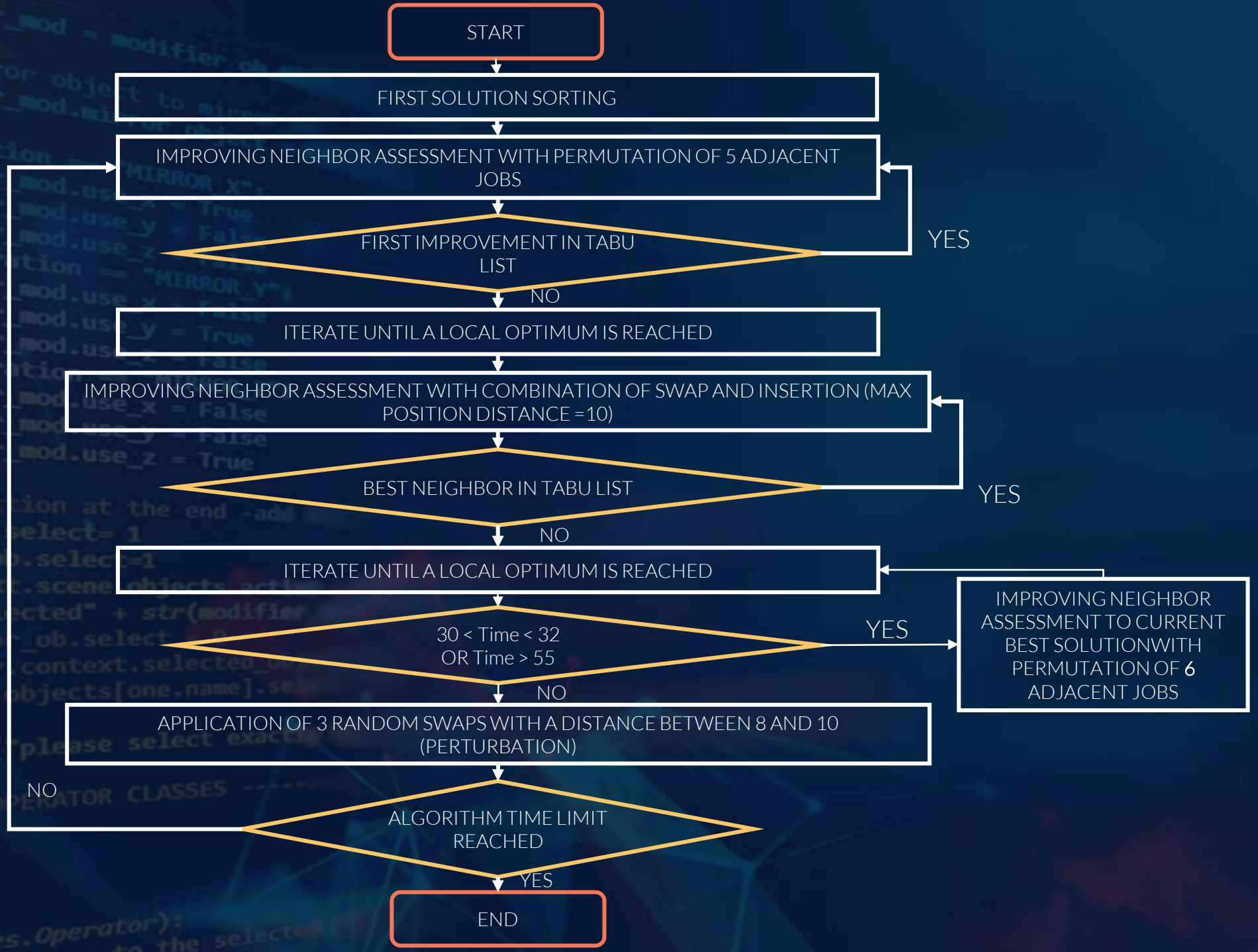
Created by Fred W. Glover in 1986, is a metaheuristic search method employing local search methods used for mathematical optimization.

One of the main components of Tabu Search is its use of adaptive memory, which creates a more flexible search behavior and enhances the performance of local search by relaxing its basic rule (to prevent being trapped in a local optima).

- At each step *worsening* moves can be accepted if no improving move is available
- *Prohibitions* (henceforth the term *tabu*) are introduced to discourage the search from coming back to previously-visited solutions.

Tabu Search approaches have set new records in finding better solutions to problems in production planning and scheduling, resource allocation, network design, routing, financial analysis, telecommunications, among many other areas.

FLOW CHART


START
FIRST SOLUTION SORTING
IMPROVING NEIGHBOR ASSESSMENT WITH PERMUTATION OF 5 ADJACENT JOBS
FIRST IMPROVEMENT IN TABU LIST
YES
NO
ITERATE UNTIL A LOCAL OPTIMUM IS REACHED
IMPROVING NEIGHBOR ASSESSMENT WITH COMBINATION OF SWAP AND INSERTION (MAX POSITION DISTANCE =10)
BEST NEIGHBOR IN TABU LIST
YES
NO
ITERATE UNTIL A LOCAL OPTIMUM IS REACHED
30 < Time < 32
OR Time > 55
YES
IMPROVING NEIGHBOR ASSESSMENT TO CURRENT BEST SOLUTIONWITH PERMUTATION OF 6 ADJACENT JOBS
NO
APPLICATION OF 3 RANDOM SWAPS WITH A DISTANCE BETWEEN 8 AND 10 (PERTURBATION)
NO
ALGORITHM TIME LIMIT REACHED
YES
END

CODING

COMPLETION TIME

The aim of the problem is to find the sequence of jobs that minimizes the sum of the completion time of the second machine. To calculate it, the following code was implemented:

```
completion_M1 = 0
completion_M2 = 0
 for x in range(len(list_)):
    completion_M1 = completion_M1 + list[job1][x]
    if completion_M1 > completion_M2:
            max = completion_M1
    else:
            max = completion_M2
    completion_M2 = max + list[job2][x]
    save completion_M2 in listCT
return(sum(listCT))
```

FIRST SOLUTION

A. JOHNSON'S ALGORITHM

Divide all the jobs in two different lists according to the following criteria; List 1: (t1>=t2) and List 2: (t2>t1) Sort first list in non-decreasing order considering duration in machine one, and sort second list in non-increasing order according to duration in machine. Finally we create one list by putting al jobs together, first those of List1 and then List2.

```
for jobs in list
    if t1>=t2:
            list1.append(job)
    else:
            list2.append(job)
sort(list1 by t1)
sort(list2 by t2, reverse order)
finallist=t1 add t2
```

B. SORT BY SUM OF TWO TIMES

Sort in non decreasing order by the sum of the times of the two jobs.

```
for job in list:
            sumlist.append(sum(job1+job2)
sort(sumlist)
```

FIRST SOLUTION

C. AS IT IS GIVEN

We take the ordered jobs as presented in the database as the first solution.

D. SORT BY SUM OF TWO TIMES AND ALTERNATE

Divide jobs in two lists, one in which t1>=t2 and other with t2>t1. Sort both lists by the sum of the times of the two jobs in a non decreasing order. The final list is the result of taking one job from list 1, then one from list 2, and so on until we have taken all jobs.

```
for jobs in list:
    if t1>=t2:
            list1.append(job1+job2)
    else:
            list2.append(job1+job2)
sort(list1)
sort(list2)
```

```
for i in max(len(list1),len(list2)):
        finallist.append(list1[i])
        finallist.append(list2[i])
        until i>min(len(list1),len(list2))
        then
        if len(list2)>len(list1)
                finallist.append(list2[i])
        else:
                finallist.append(list1[i])
```

O(nlogn)

FIRST SOLUTION

E. IMPROVEMENT OF SORT BY SUM OF TWO TIMES AND ALTERNATE

Considering the total time of the jobs: if one is much larger than the other then put the shortest one first.
Two parameters were changed: the relative distance of the sum of the two times between the two jobs being compared, and the number of "exceptions" to the rule allowed. The following table shows the results:

We tried to put some conditions to don't take just one and one from each list but to consider the relative difference of the sum of the times. We could find a combination of parameters that improved for some data the first result: maximum of exceptions to the rule allowed:2 and relative distance of the sum of the 2 times between 2 different jobs should be greater than 1.3. However, as the final result after applying the algorithm was always worse, we decided to go back to the original version.

Distance 1.3, exceptions 2	Distance 1.2, no restriction of n° of exceptions	Distance 1.3, no restriction of n° of exceptions
195800	196356	195800
180309	178476	180309
195895	201030	195895
179381	183065	181495
195425	197698	195379

FINAL RESULTS FOR THE IMPROVED FIRST SOLUTION (JUST FOR THE 3 DATA THAT IMPROVES)						
	improved	not improved	improved	not improved	improved	not improved
First result	180309	182294	179381	179392	195425	196771
Final result	175974	175534	173380	173380	187562	187511

FIRST SOLUTION

RESULTS AND COMPARISON

DATA	As it is given	Johnson's Algorithm	Sort by sum of two times	Sort by sum of two times and alternate (Final version)
1	277469	282k	196485	195800
2	252011	264k	187613	182294
3	269736	284k	204649	195895
4	240651	263k	190778	179392
5	266826	288k	209929	196771

NEIGHBORHOOD

A. SIMPLE SWAP

1. Neighborhood: for each distance of swap (from 1 till the maxdistance set) tries every possible swap in order to create neighbors.

```
for j in range(1, Maxdistancetoswap + 1):
  for i in range(len(listtoswap) - j):
     swap[i,j]
     if swap[i,j] improves currentsolution:
        save swap in list
return  sorted(list) #Sorted by the values of improvements
```

2.Update the current solution: Apply the best swap to the original list

```
do the best swap
repeat with updated currentsolution
```

O(n^2)

NEIGHBORHOOD

SIMPLE SWAP RESULTS

	Max Swaps allowed: Result(Iterations)								
Data	0	1	2	3	4	5	20	30	BEST
1	195800	192481	192488	192349	192377	192483	192512	192512	192349
2	182294	176110	176298	175762	176098	175892	175976	175976	175762
3	195895	193658	192356	192713	192526	192518	191740	191170	191170
4	179392	174446	174299	173713	173855	173734	173438	173438	173438
5	196771	188454	187700	187367	187974	187635	187549	187549	187367
Average	190030,4	185029,8	184628,2	184380,8	184566	184452,4	184243	184129	184129

NEIGHBORHOOD

B. INSERTION

1. Neighborhood: tries all possible insertions with a maximum distance of "x" in order to create neighbors.

```
for j in range(1,maxdistancetoinsert+1):
  for i in range(len(listoriginal) - j):
    insert[i,j]
   (if  j>1: #we dont insert twice when we insert with a distance of 1, which is the same to swap)
       insert[j,i]
    if insert improves currentsolution:
      save insert in list
return sorted(list)
```

2.Update the current solution: Apply the best insertion to the original list

```
do the best insertion
repeat with updated currentsolution
```

O(n^2)

NEIGHBORHOOD

INSERTION RESULTS

	Maximum distance of insertion									
Data	0	1	2	3	4	5	6	7	8	BEST
1	195800	192481	192342	192359	192325	192379	192357	192410	192368	192325
2	182294	176110	175791	175701	175691	175752	175898	175987	175882	175691
3	195895	193658	192915	192945	191998	191753	192562	192626	191007	191007
4	179392	174446	174063	173711	173442	173462	173305	173423	173285	173285
5	196771	188454	188402	187569	187246	187853	187424	187528	187954	187246
Average	190030,4	185029,8	184702,6	184457	184140,4	184239,8	184309,2	184394,8	184099,2	184099,2

NEIGHBORHOOD

C. HYBRID

1.Neighborhood: tries all possible swaps with a maximum distance of "x" and tries all possible insertions with a maximum distance of "y" in order to create neighbors.

```
for j in range(1, Maxdistancetoswap + 1):
  for i in range(len(listtoswap) - j):
     swap[i,j]
     if swap[i,j] improves currentsolution:
        save swap in list
 for j in range(1,maxdistancetoinsert+1):
   for i in range(len(listoriginal) - j):
     insert[i,j]
    (if  j>1: #we dont insert twice when we insert with a distance of 1, which is the same to swap)
         insert[j,i]
     if insert improves currentsolution:
       save insert in list
return sorted(list)
```

2.Update the current solution: Apply the best move to the original list

```
do the best move
repeat with updated currentsolution
```

O(n^2)

NEIGHBORHOOD

HYBRID RESULTS

DATA 1		INSERTION								
		0	1	2	3	4	5	6	7	8
SWAP	0	195800	195800	192382	192431	192387	192488	192465	192448	192438
	1	192481	192481	192342	192359	192325	192379	192357	192410	192368
	2	192488	192488	192352	192338	192293	192299	192277	192306	192348
	3	192349	192349	192303	192283	192268	192299	192257	192285	192300
	4	192377	192377	192290	192282	192261	192262	192258	192286	192289
	5	192483	192483	192442	192370	192306	192313	192313	192298	192298
	6	192583	192583	192545	192470	192444	192428	192428	192360	192360
	7	192597	192597	192433	192433	192433	192417	192418	192360	192394
	8	192541	192541	192381	192384	192384	192368	192368	192363	192401
								AVERAGE		192367
								MINIMUM		192257

NEIGHBORHOOD

HYBRID BEST RESULTS

DATA	SWAP	INSERTION	RESULT
1	3	6	192257
2	5	4	175600
3	2	8	190990
4	6	7	173140
5	2	4	187231

To set the best compromise pair of parameters (distances), we calculated the average for each pair and then the minimum is the best compromise solution for fixed parameters.

NEIGHBORHOOD

HYBRID AVERAGE

		INSERTION								
		0	1	2	3	4	5	6	7	8
SWAP	0	190030.4	190030.4	184631.0	184464.2	184165.0	184195.6	184372.8	184416.4	184136.4
	1	185029.8	185029.8	184702.6	184457.0	184140.4	184239.8	184309.2	184394.8	184099.2
	2	184628.2	184628.2	184534.4	184242.2	184098.6	184059.0	184171.4	184200.6	184062.0
	3	184380.8	184380.8	184259.4	184121.6	184106.6	184061.6	184035.4	183993.2	184064.6
	4	184566.0	184566.0	184413.6	184359.8	184303.2	183932.8	184209.2	184285.4	184205.0
	5	184452.4	184452.4	184070.6	184281.0	184128.8	184163.8	184132.8	184144.4	184201.0
	6	184408.6	184408.6	184352.4	184368.0	184269.2	184164.4	184234.4	184122.8	184238.8
	7	184389.4	184389.4	184251.4	184131.6	184009.8	184143.0	184046.4	184055.0	184209.8
	8	184452.6	184452.6	184379.2	184326.8	184256.0	184052.8	184034.8	184003.2	184087.2
									AVERAGE	184238.0
									MINIMUM	183932.8

BEST COMPROMISE RESULT

OPTIMUM COMBINATION: SWAP 4, INSERTION 5, OPTIMUM AVERAGE 183932.8

NEIGHBORHOOD

D. TRUNCATE

Neigborhood: Then we tried to truncate the search of neighbors to reduce time. The search for swaps that improve the solution stops when “x” improvements are found. The search for insertions that improve the solution stops when “y” (we decided to set y=x) improvements are found.

```
for swaps:
if len(solutionslist)>x:
break
for insertion:
if len(solutionslist)>x+y:
break
```

TRUNCATION	BEST	COMBINATION	CONCLUSION
First 5 swaps that improves and 5 first insertions that improve		Swap= 7 Insertion= 8	Bad because it leads to worse results and there is no considerable save of time
BEST AVERAGE:	184064.8		
First 3 swaps that improves and 3 first insertions that improve		Swap= 5 Insertion= 5	Still 100 worse on average
BEST AVERAGE:	184048.2		

NEIGHBORHOOD

E. DOBLE SWAP

1.Neighborhood: We tried to take the best “x” swaps (for example 5), and for that new swapped list do all the possible swaps and save the best improvement. Then the best 2 swaps (the 2 swaps that improve the solution the most) are done to the original list.

```
for j in range(1, Maxdistancetoswap + 1):
  for i in range(len(listtoswap) - j):
     swap[i,j]
     if swap[i,j] improves currentsolution:
        save swap in list
return  sorted(list) #Sorted by the values of improvements
```

2.Update the current solution: Apply the best swap to the original list

Repeat 1. for the best “x” swaps
do max(improvement1 + improvement2) swaps
repeat with updated currentsolution

O(n^2)

NEIGHBORHOOD

DOBLE SWAP RESULTS

	Max Swaps allowed:					
Datos	0	1	2	3	4	5
1	195800	192458	192387	192337	192421	192466
2	182294	176110	176386	175833	175960	175868
3	195895	NO IMPROVEMENTS, sometimes worse solutions and much more time				
4	179392					
5	196771					

PERTURBATION

After reaching the first local optima, since the time to compute the results was shorter than 5 seconds, we tried to take advantage of the remaining time by swapping the first local optimum and then apply the hybrid(4,5) to that new starting point.
The result was not improving if the distance of the swaps or the number of swaps was too big (for example, a distance greater than 10, and more than 15 swaps even with shorter distances)
As the swaps are done at random, it was not easy to fix the parameters, but came to the conclusion that for around 8 swaps of a distance between (4,6), the program is able to improve the most the results: on average on -60 for a total of one minute.

```
apply hybrid(4,5)
swaps=0
while swaps<8:
        start=randit(0,99)
        distance=randit(4,6)
        final=start+distance
        if final>len(localoptlist)
                final=start-distance
        swap(localoptlist[start, final])
apply hybrid(4,5)
```

TABU LIST

The implementation of tabu moves was more complicated cause we have different moves of the jobs in order to find different neighbors (permutation, swaps and insertion).

After having tried to implement tabu moves (in a simpler way), we found that no significant improvements were found in a sufficiently short time period (less than one minute) and, in comparison to an iterated local search with a tabu list of solutions, the results were always worse.

Moreover, although the time to check if a solution is or not in the tabu list is linear instead of constant (in the case in which tabu list consists on moves), the overall complexity remains the same because we only check for the best move and not for the whole neighborhood.

NEW NEIGHBORHOODS

After trying all previous explained neighborhoods, we thought about new ways to generate neighborhoods or to find better solutions.

Some attempts failed:

- **Shorterproblem:** we find the optimal solution for a shorter piece of the list to improve with size "k" (taken from the list to improve, from i=0 to n-k), then we see if applying the improvement of the shorter list, improves also the global CT. If yes that new sequence replace the last one. We do it until a local optimum is reached.

- **Comeback:** Once we found the first local optimum, we come back to the previous sequence before reaching the optima (we undo one move) and if there was other move to improve the solution, we do it. We continue with steepest descent until we reach a local optimum, and we repeat that logic until a time lapse has expired.

We found that there is no relationship between a local improvement (i.e. an improvement inside a list of size k<n) and a global improvement: they are quit independent. A local improvement doesn't imply a global improvement.

NEW NEIGHBORHOODS

Finally we tried to generate neighborhoods not only doing a single swap or insertion but generating all possible permutations for a shorter list of short size: 4, 5 and 6 (pieces of the list to improve), and for each one we calculate the new CT of the global list. We do it for i=0 to i=n-k. This case is similar to "shorter problem" but instead of looking for the local optima we analyzed all possible combinations for those 4/5/6 jobs. We do steepest descent with this neighborhood.
Now the number of neighbors grows to **k!(n-k).**

1. **Neighborhood:** all possible permutations for a list of size "k".

```
permutationslist=permutations([0,1,...,k])
for j in range(len(listtoimprove)-k):
   for change in permutationslist:
     if  change improves currentsolution:
          save change list
return sorted(list)
```

2.**Update the current solution**: Apply the best insertion to the original list

```
do the best move
repeat with updated currentsolution
```

NEW NEIGHBORHOODS

Below we present the results for a list of size 4 (k=4) applying that function combined with **hybrid or swaps** to reduce time:

	A	B	C
1	192219	192206	192235
2	175565	175558	175679
3	191890	191001	191052
4	173488	173441	173208
5	187525	187329	187230
AVERAGE	184137.4	183907	183880.8

Then we tried to expand the shortlist size to 5:

	D	E	F
1	192185	192206	192226
2	175564	175501	175489
3	191942	190933	190996
4	172915	172959	172904
5	187174	187323	187185
AVERAGE	183956	183784.4	183760

A: Starting from the initial solution obtained with the constructive method chosen, we apply steepest descent as indicated above for k=4. The results are worse than the average of doing hybrid(4,5).

B: Starting from the initial constructive solution first we apply steepest descent with just adjacent swaps. Then we apply the function of permutations with k=4. Finally we apply hybrid(10,10). The results are slightly better than the average of doing hybrid(4,5)

C: Starting from the initial constructive solution we apply steepest descent with the function hybrid(4,5). Then we apply the function of permutations with k=4. Finally we apply hybrid(10,10). The results are slightly better than the average obtained with B.

D: Starting from the initial solution obtained with the constructive method chosen, we apply steepest descent as indicated above for k=5. The results are worse than the average of doing hybrid(4,5).

E: Starting from the initial constructive solution we apply steepest descent with just adjacent swaps. Then we apply the function of permutations with k=5. Finally we apply hybrid(10,10). The results are better than the average of doing hybrid(4,5).

F: Starting from the initial constructive solution we apply steepest descent with the function hybrid(4,5). Then we apply the function of permutations with k=5. Finally we apply hybrid(10,10). The results are better than the average obtained with E.

NEIGHBORHOOD TRUNCATION

Even though the results were better, the times with k=5 were pretty long (around 10" on average), so we decided to apply 2 kinds of truncation:

1. Don't start from the very first element at each iteration, but start from the element (previous start-k) where k is the size of the short list (5), and previous start is the one from which we found the improvement on the last iteration (we applied this since we noticed that many times the following improvements were not present before the last improvement but after it).If from that new starting point to the last element of the list no improvement is found, then we compute the neighbors from the first element till we reach the new starting point (first element to previous start).

2. Do not calculate all the neighborhoods but pause at the first improvement (permutation "x" of element "i" that improves the solution). When reaching that point, we calculate all the permutations for that list to find the best improvement in that sorter list and then, stop the search and not continue with the element i+1.

The time was reduced to 3" on average and the results improved a bit as well: (F truncated)

1	2	3	4	5	AVERAGE
192226	175493	190921	172898	187185	183744.6

NEIGHBORHOOD TRUNCATION

But at this point, since most of the time was taken by the hybrid(4,5) and not by the permutations, we tried to apply directly the permutations neighborhood to the starting solution, and while the time remained exactly the same than F truncated, the results improved.

1	2	3	4	5	AVERAGE
192238	175542	191013	172744	187093	183726

And finally we apply the **perturbation** code to the first local optimum for one minute and we improved the results:

1	2	3	4	5	AVERAGE
192200	175408	190686	172703	186988	183597

Also applying **perturbation** with 3 swaps and a distance between 8 and 10 we reached:

1	2	3	4	5	AVERAGE
192180	175351	190635	172699	186996	183572.2

IMPLEMENTATION AND RESULTS

FINAL CODE

```
#OPEN FILE AND CREATE A LIST
        file=selectfile()
        list_=filetolist(file)
#INITIALIZE COUNTER
        start = time.time()
#FIRST SOLUTION
#INITIALIZE TABU VARIABLES
        tabulist=[]
        totaltabufound = 0
#PERMUTATION
#HYBRID
#CHECK RESULTS
        if currentsolution improves bestsolution;
        bestsolution=currentsolution
#PERTURBATION
#REPETITION
        repeat from permutations neighborhoods until the time of 60” is reached
```

FINAL CODE

#FIRST SOLUTION: SORT BY SUM OF TWO TIMES AND ALTERNATE

```
for jobs in list:
        if t1>=t2:
                list1.append(job1+job2)
        else:
                list2.append(job1+job2)
sort(list1)
sort(list2)
for i in max(len(list1),len(list2)):
        finallist.append(list1[i])
        finallist.append(list2[i])
        until i>min(len(list1),len(list2))

        if len(list2)>len(list1):
                finallist.append(list2[i])
        else:
                finallist.append(list1[i])
```

FINAL CODE

#PERMUTATIONS TRUNCATED : look for improvements by calculating the CT of a newlist in which we permutate a shortlist of size 5. When one improvement is found we stop. We start the search from the point in which we found the last improvement.

```
permutationslist=permutations([0,1,...,k])
for j in range(startfrom-5,len(listtoimprove)-4):
        for change in permutationslist:
            if change improves currentsolution:
                save change in list
        if at least one improve found:
            break
if no improve found:
for j in range(max(startfrom-5,0)):
        for change in permutationslist:
            if change improves currentsolution:
                save change in list
            if at least one improve found:
            break
return sorted(list)
```

```
if newsolution not in tabulist:
        do the best move
        tabulist.append(newsolution)
else:
        repeat from the 2nd best until one
solution not in tabulist is found

repeat with updated currentsolution until a local
optimum is reached
```

FINAL CODE

```
#HYBRID
for j in range(1, 10 + 1):
        for i in range(len(listtoswap) - j):
            swap[i,j]
            if swap[i,j] improves currentsolution:
                save swap in list
 for j in range(1,10+1):
     for i in range(len(listoriginal) - j):
         insert[i,j]
     if  j>1: #we dont insert twice when we insert with a distance of 1, which is the same to swap)
         insert[j,i]
     if insert improves currentsolution:
         save insert in list
return sorted(list)
if newsolution not in tabulist:
        do the best move
        tabulist.append(newsolution)
else:
        repeat from the 2nd best until one solution not in tabulist is found

repeat with updated currentsolution until a local optimum is found
```

FINAL CODE

```
#PERTURBATION

while swaps<3:
        start=randit(0,99)
        distance=randit(8,10)
        final=start+distance
        if final>len(currentsolution)
                final=start-distance
        swap(currentsolution[start, final])
```

COMPLEXITY

The maximum number of elementary operations that we would need to do in the worst case as a function of the number of jobs “n”

Algorithm Part	**Complexity**
First Solution	O(nlogn)
Calculation of the CT	O(n)
Permutations neighborhood	O(n^2)= (O(n)(neighbors)*O(n)(CT))
Hybrid	O(n^2) = (O(n)(neighbors)*O(n)(CT))
Perturbation	Repeat O(n^2)
Tabu List check	O(n)
Total:	**O(nlogn)+O(n^2)+O(n^2)=O(n^2)**

RESULTS

BEST COMPROMISE RESULTS: Permutation 5 truncated + hybrid 10,10 (10 neighbors for swap, 10 for insertion) and applying perturbation.

Data	BEST Result	BEST Result professor	Abs. Distance	Relative Distance
1	192164	192155	9	0,00468%
2	175351	175322	29	0,01654%
3	190566	190539	27	0,01417%
4	172605	172598	7	0,00406%
5	186977	186942	35	0,01872%
				0,01163%

RESULTS

AVERAGE RESULTS: In order to have a better picture of the efficiency of the code, we run it 5 times per each set of data.

	TRY					
Data	1	2	3	4	5	Average
1	192164	192190	192180	192167	192172	192174,6
2	175355	175367	175351	175406	175352	175366,2
3	190575	190658	190635	190846	190566	190656
4	172624	172605	172699	172649	172631	172641,6
5	186977	187057	186996	187024	186990	187008,8

AVERAGES:

Data	Average	Average professor	Abs. Distance	Relative Distance
1	192174,6	192157	17,6	0,00916%
2	175366,2	175336	30,2	0,01722%
3	190656	190544,8	111,2	0,05836%
4	172641,6	172647,2	-5,6	-0,00324%
5	186987	186952,6	34,4	0,01840%
	183565,08	183527,52		0,01998%

RESULTS

BEST RESULT FOR SET OF DATA 1

192164

Final list: [[1, 21], [8, 7], [9, 8], [12, 11], [17, 12], [12, 14], [10, 18], [11, 18], [27, 9], [8, 24], [27, 21], [21, 29], [28, 19], [19, 40], [36, 20], [12, 45], [52, 8], [1, 52], [46, 19], [30, 31], [40, 22], [3, 61], [68, 8], [18, 48], [35, 33], [41, 34], [27, 45], [57, 15], [10, 61], [62, 20], [5, 71], [62, 21], [37, 40], [50, 32], [34, 48], [50, 30], [2, 73], [62, 29], [64, 22], [14, 70], [75, 16], [14, 73], [75, 20], [21, 74], [67, 26], [39, 57], [56, 38], [33, 63], [56, 42], [55, 39], [35, 61], [51, 47], [42, 60], [76, 23], [17, 80], [88, 16], [11, 82], [76, 25], [37, 68], [47, 60], [80, 28], [21, 88], [77, 32], [50, 65], [52, 58], [68, 49], [36, 82], [88, 30], [37, 84], [72, 52], [63, 58], [63, 61], [54, 64], [62, 65], [59, 70], [83, 47], [43, 81], [85, 44], [34, 89], [90, 44], [50, 77], [75, 57], [63, 73], [67, 65], [69, 65], [59, 85], [85, 50], [56, 86], [88, 51], [46, 88], [88, 66], [60, 86], [88, 72], [75, 82], [86, 84], [63, 90], [89, 84], [90, 84], [87, 90], [90, 90]]

Sequence: [29, 7, 54, 22, 59, 79, 35, 63, 32, 61, 77, 21, 53, 75, 84, 82, 78, 12, 41, 70, 83, 67, 73, 97, 81, 64, 45, 65, 39, 17, 10, 6, 72, 27, 60, 4, 33, 16, 28, 40, 30, 98, 49, 100, 19, 55, 69, 43, 88, 95, 74, 62, 76, 71, 36, 50, 86, 96, 94, 25, 85, 99, 89, 56, 58, 13, 14, 52, 47, 18, 42, 1, 11, 34, 90, 44, 8, 66, 48, 80, 38, 68, 37, 93, 92, 24, 46, 15, 2, 9, 23, 87, 91, 31, 51, 3, 26, 5, 20, 57]

RESULTS

BEST RESULT FOR SET OF DATA 2

175351

Final list: [[2, 24], [3, 3], [9, 3], [11, 10], [14, 12], [8, 15], [18, 14], [11, 17], [20, 18], [22, 18], [17, 34], [32, 8], [3, 43], [32, 12], [25, 16], [14, 40], [31, 17], [34, 15], [15, 37], [26, 25], [36, 13], [4, 48], [47, 4], [6, 52], [43, 11], [27, 29], [21, 36], [34, 33], [18, 43], [65, 4], [8, 62], [62, 14], [14, 62], [36, 43], [67, 7], [6, 75], [47, 28], [56, 21], [15, 68], [47, 31], [60, 17], [10, 72], [52, 29], [50, 31], [30, 56], [63, 17], [8, 78], [78, 8], [7, 79], [66, 24], [42, 42], [42, 43], [43, 48], [48, 59], [63, 32], [33, 70], [65, 33], [21, 73], [84, 17], [21, 81], [76, 28], [23, 85], [78, 28], [46, 60], [53, 54], [41, 70], [76, 29], [43, 73], [62, 47], [41, 74], [83, 23], [34, 82], [72, 41], [50, 61], [52, 66], [66, 48], [59, 65], [64, 62], [60, 63], [52, 83], [89, 32], [43, 82], [74, 55], [59, 71], [74, 52], [54, 77], [71, 64], [68, 66], [68, 80], [74, 61], [72, 73], [74, 90], [78, 56], [77, 80], [79, 84], [83, 79], [85, 77], [87, 66], [89, 47], [89, 57]]

Sequence: [68, 17, 99, 74, 37, 93, 85, 8, 38, 52, 16, 45, 81, 71, 28, 62, 46, 67, 41, 65, 80, 78, 100, 35, 79, 27, 70, 66, 90, 18, 44, 9, 83, 24, 60, 30, 97, 63, 47, 15, 33, 89, 42, 13, 7, 87, 96, 64, 51, 1, 50, 14, 23, 11, 61, 36, 43, 53, 91, 29, 21, 75, 98, 92, 59, 72, 25, 12, 95, 10, 54, 34, 73, 20, 26, 6, 22, 82, 88, 84, 57, 49, 76, 32, 4, 31, 5, 2, 19, 69, 55, 3, 77, 48, 94, 39, 58, 40, 86, 56]

RESULTS

BEST RESULT FOR SET OF DATA 3

190566

Final list: [[1, 51], [29, 6], [18, 8], [6, 16], [25, 11], [9, 40], [36, 2], [9, 35], [29, 17], [15, 33], [35, 8], [11, 40], [40, 12], [15, 37], [18, 32], [48, 2], [1, 53], [54, 4], [1, 54], [49, 7], [12, 43], [38, 24], [32, 29], [34, 30], [14, 51], [51, 12], [27, 39], [21, 50], [52, 13], [11, 66], [71, 2], [16, 64], [47, 24], [39, 40], [41, 38], [23, 58], [61, 16], [28, 62], [48, 29], [42, 49], [29, 59], [67, 21], [33, 55], [50, 37], [43, 47], [29, 65], [76, 7], [5, 84], [75, 14], [23, 71], [76, 15], [18, 77], [79, 19], [14, 85], [76, 24], [38, 58], [51, 41], [42, 55], [59, 40], [37, 61], [48, 53], [62, 41], [29, 66], [83, 22], [21, 82], [86, 31], [8, 90], [81, 34], [64, 46], [36, 71], [78, 44], [37, 72], [85, 40], [23, 86], [90, 40], [36, 77], [90, 51], [40, 79], [84, 54], [60, 72], [75, 61], [61, 71], [61, 71], [79, 63], [64, 74], [75, 64], [46, 79], [80, 68], [78, 70], [74, 82], [84, 68], [70, 80], [80, 78], [77, 84], [85, 78], [33, 87], [86, 81], [64, 87], [50, 89], [87, 90]]

Sequence: [99, 81, 29, 31, 8, 12, 69, 77, 94, 93, 26, 24, 25, 18, 33, 41, 67, 73, 34, 92, 30, 79, 78, 97, 76, 23, 40, 45, 62, 70, 1, 37, 83, 16, 85, 20, 91, 42, 58, 32, 21, 50, 71, 9, 44, 38, 13, 64, 98, 80, 57, 86, 5, 47, 88, 82, 55, 51, 22, 15, 63, 100, 68, 48, 17, 35, 66, 10, 14, 65, 46, 60, 2, 59, 53, 75, 6, 90, 3, 56, 43, 27, 95, 54, 28, 7, 39, 11, 61, 89, 19, 72, 49, 96, 4, 36, 52, 87, 84, 74]

RESULTS

BEST RESULT FOR SET OF DATA 4

172605

Final list: [[3, 4], [4, 38], [17, 3], [13, 4], [15, 13], [10, 32], [18, 4], [17, 7], [10, 35], [24, 2], [14, 35], [22, 12], [21, 12], [16, 43], [36, 1], [7, 50], [32, 15], [31, 8], [4, 60], [39, 7], [30, 23], [21, 33], [28, 34], [42, 7], [8, 55], [48, 9], [16, 63], [46, 3], [22, 55], [33, 31], [52, 4], [3, 88], [53, 16], [53, 7], [8, 78], [51, 18], [38, 38], [40, 30], [32, 57], [58, 9], [12, 80], [66, 2], [12, 88], [58, 16], [46, 33], [37, 43], [40, 45], [40, 48], [54, 20], [15, 84], [65, 9], [32, 64], [53, 25], [37, 57], [48, 44], [43, 46], [52, 32], [16, 86], [76, 6], [36, 71], [53, 41], [52, 42], [49, 37], [39, 67], [57, 38], [48, 57], [37, 76], [72, 17], [34, 79], [43, 72], [79, 12], [47, 68], [59, 50], [56, 60], [63, 40], [41, 86], [67, 38], [53, 65], [67, 49], [45, 78], [58, 87], [79, 22], [63, 63], [62, 73], [70, 60], [65, 65], [66, 79], [76, 50], [66, 90], [80, 11], [67, 86], [81, 14], [78, 84], [81, 28], [82, 16], [82, 57], [82, 41], [82, 87], [90, 87], [90, 16]]

Sequence: [99, 31, 37, 26, 11, 82, 46, 52, 42, 17, 74, 60, 15, 76, 68, 33, 85, 1, 54, 88, 32, 75, 78, 92, 94, 83, 6, 30, 91, 97, 58, 14, 81, 47, 53, 84, 63, 70, 4, 72, 23, 34, 3, 45, 89, 12, 95, 57, 93, 28, 44, 55, 19, 73, 77, 36, 39, 71, 50, 38, 51, 100, 20, 90, 80, 65, 9, 25, 2, 56, 29, 59, 43, 48, 86, 69, 67, 7, 27, 96, 79, 41, 61, 22, 49, 21, 62, 8, 10, 35, 40, 66, 16, 5, 18, 98, 87, 24, 64, 13]

RESULTS

BEST RESULT FOR SET OF DATA 5

186977

Final list: [[1, 24], [17, 2], [7, 9], [3, 18], [17, 13], [20, 2], [4, 25], [22, 3], [1, 52], [23, 4], [22, 15], [27, 10], [1, 57], [36, 9], [35, 12], [7, 51], [37, 17], [42, 12], [12, 50], [49, 2], [5, 59], [44, 10], [24, 44], [44, 21], [12, 58], [48, 13], [29, 46], [48, 20], [10, 63], [51, 19], [35, 38], [44, 36], [30, 51], [59, 29], [13, 63], [77, 9], [10, 66], [67, 21], [17, 62], [55, 32], [38, 45], [49, 39], [40, 43], [18, 64], [86, 3], [1, 77], [67, 23], [25, 57], [48, 41], [32, 59], [80, 24], [31, 61], [58, 45], [49, 49], [23, 64], [85, 26], [23, 65], [54, 50], [71, 36], [35, 66], [66, 38], [33, 68], [52, 56], [52, 60], [75, 41], [52, 57], [55, 58], [46, 70], [84, 44], [44, 70], [70, 53], [53, 63], [47, 71], [83, 44], [43, 73], [75, 61], [63, 63], [38, 75], [88, 60], [55, 73], [81, 68], [78, 66], [56, 74], [74, 72], [63, 74], [65, 74], [59, 79], [55, 80], [77, 80], [69, 81], [76, 81], [79, 82], [25, 84], [86, 86], [55, 86], [80, 86], [2, 87], [1, 88], [26, 90], [5, 90]]

Sequence: [100, 94, 92, 2, 65, 73, 88, 26, 29, 7, 54, 5, 71, 68, 99, 33, 6, 61, 16, 75, 30, 8, 36, 48, 93, 78, 58, 28, 15, 20, 32, 51, 83, 66, 45, 35, 59, 38, 14, 62, 27, 11, 89, 31, 53, 67, 81, 60, 98, 63, 17, 43, 37, 25, 52, 42, 50, 70, 76, 22, 23, 91, 40, 57, 18, 82, 13, 77, 41, 72, 21, 12, 85, 24, 84, 96, 49, 9, 19, 64, 10, 74, 69, 80, 95, 46, 1, 55, 86, 79, 3, 87, 47, 4, 39, 44, 56, 97, 34, 90]

CONCLUSIONS

Our algorithm takes advantage from:

- Variable neighborhood search: since we first apply permutations 5, then we extend the neighborhood to hybrid 10,10 and finally we apply (two times in different moments) permutation 6.
- Iterated local search: since we perturbate the first local optimum and we reapply local search until we reach the stop condition (one minute).
- Tabu search: since we don't allow to go back to an already visited solution.

The complexity of the algorithm is polynomial O(n^2), so we consider that it is not so high and acceptable for the results we obtain.

This allows us to find approximately 30 different local optimums in a period of one minute.

The results obtained seem to be accurate and with good quality.