4th International Conference on Industrial Engineering and Industrial Management
XIV Congreso de Ingeniería de Organización
Donostia- San Sebastián , September 8th -10th 2010

Parallel CLM starting solutions comparison*

Jose Miguel León Blanco[1], Jose L. Andrade Pineda[1], Manuel Dios Rubio[1], Rafael Ruiz-Usano[1]

[1] Dpto. de Organización de Empresas. Escuela Técnica Superior de Ingenieros. Universidad de Sevilla. Cº de los Descubrimientos s/n, 41092 Sevilla. miguel@esi.us.es, jlandrade@esi.us.es, mdios@us.es, usano@us.es

Abstract

Local search methods are sensitive to the way initial solutions are generated. In this communication the differences in terms of quality of solutions and execution time due to the method used to generate starting solutions are shown. Guided initial solutions usually lead to higher quality final solutions but the algorithm can converge quickly to poor quality local optima. Using randomly generated starting solutions could supply more diversified solutions at a cost of a greater computation time. In this communication, both methods are compared when applied in a parallel algorithm to the permutation flow shop problem.

Keywords: parallel algorithms, local search

1. Introduction

Complete Local search with Memory or CLM, Ghosh and Sierksma (2002), is a local search based metaheuristic that takes advantage of its configuration parameters resembling the behaviour of other metaheuristics like tabu search and simulated annealing. We have implemented two parallel metaheuristics based in this method, Leon et al. (2006), obtaining good results in terms of efficiency in its application to the permutation flow shop problem. In this communication, as a complement to that communication we study the differences in terms of quality of solutions and execution time due to the method used to generate starting solutions. We will compare the effect of start from randomly generated solutions and from solutions generated using a constructive heuristic. Guided initial solutions usually lead to higher quality final solutions but the algorithm can converge quickly to poor quality local optima. Using randomly generated starting solutions could supply more diversified solutions at a cost of a greater computation time. We will study the differences when both methods are applied in a parallel algorithm to a NP-Hard combinatorial optimization problem like the permutation flow shop problem*. In this problem, a number of jobs are processed on a number of machines. The flow shop permutation schema forces each job to be processed through all the machines in exactly the same order and this order must be the same on each machine. Each of the machines can process only one job simultaneously and each job may be processed only in one machine simultaneously. The objective function in this problem is to minimize the maximal completion time of all of the jobs or makespan.

* This work stems from the participation of the authors in a research project funded by the Spanish Ministry of Science and Innovation, grant DPI2007-61345, title SAGIP "Advanced Systems for Integrated Order Management", and by the Andalusian Government, grant P08-TEP-3630, title SCOPE "Cooperative Systems for Order Programming and Fulfilment"

2. Parallel Local Search with Memory

Local search parallel algorithms have a good balance between simplicity and effectiveness, so they are well suited to obtain good quality solutions to NP-Hard problems like the one we are studying in a reasonable amount of time. There are algorithms that find good quality solutions by using very complex methods or combining several heuristics, like Zobolas et al. (2008) but the effort to implement them are hard compared to simpler ones.

Parallel algorithms can reduce computing time and increase the quality of solutions by sharing the computational effort between two or more processors and taking advantage of the communication between processes to increase the quality of solutions. There are several parallel algorithms applied to the permutation flow shop problem that have been successful in this task. In Okamoto et al. (1994) and Okamoto et al. (1995) exact methods are implemented in parallel. Simulated annealing is parallelized in Wodecki and Bożejko (2002) and in Bożejko and Wodecki (2004a). Tabu Search is used in Wodecki and Bożejko (2002) and in Bożejko and Wodecki (2004b). Genetic algorithms are used in Kohlmorgen et al. (1999) and Scatter Search in Bożejko and Wodecki (2008a), Bożejko and Wodecki (2008b) and Bożejko (2009).

2.1. Description of parallel algorithm

The studied algorithm follows a master/worker architecture as proposed by Talbi et al. (1998) amongst others. In this schema, one of the processes, the master, controls the search and the others, the workers, make the work of exploring the neighbourhood of the solution provided by the master. The parallel communication schema is a coarse grained one because the time spent searching the neighbourhood is much greater than the time spent by the communication between processes, as it is implemented in a cluster of workstations without shared memory. As another point of comparison the experiments have been run in a workstation with a multicore CPU (an Intel Quad Core one) to see differences between computer platforms. We have observed in Leon et al. (2006) the best values of efficiency of the parallel algorithm when using 3 or 4 processors, so it is interesting to study the performance of the algorithm in current multicore workstations.

Master process contains a memory or pool of solutions from where worker processes read and write solutions and parameters used in the search. The pool is modelled as a list ordered by makespan value, being the first solution in the list the best one in terms of makespan. In this list, each element or node contains a pointer to the next one. The nodes stored in master pool contain the following information (Table 1):

Table 1. Information in central pool nodes

Data	Comment
Process	Process where the solution has been found
Sequence	The sequence in a permutation flow shop problem
Makespan	The makespan associated with that solution
Mobility	A parameter related to the quality of the solution
Memsize	Top of memory in the CLM algorithm
K	Number of solutions explored in each CLM iteration
Iterations	Top of iterations in CLM algorithm
Next_pool	Pointer to next node in the list

This information is used by worker processes in order to intensify or diversify the search. Mobility is a concept used in the parallel algorithm developed by Crainic y Gendreau (2002). This concept guides the selection of a solution from the list to send it to a worker process. The mobility m_i increases in one when a new solution is added to the list after this one –has a

worse makespan- and keeps its value when added solution is before this one in the list. The solution S_i which is going to be sent to a worker is chosen from the list with a probability $P(S_i)$ (1) that depends on its mobility m_i the number of solutions in central pool, *npool*, the order of the solution in the list *i* and, inversely, on the sum of mobility of solutions with worse makespan in the list.

$$P(S_i) = \frac{m_i + npool + 1 - i}{\sum_i^{npool}(m_i + i)} \quad (1)$$

The information sent from master to workers and viceversa follows a similar schema (Table 2). There can be saved communication weight by sending only the data needed by workers to perform the search.

Table 2. Information communicated between processes

Data	Comment
Sequence	The sequence in a permutation flow shop problem
Makespan	The makespan associated with that solution
Memsize	Top of memory in the CLM algorithm
K	Number of solutions explored in each CLM iteration
Iterations	Top of iterations in CLM algorithm

The information needed by workers only contains the starting solution and parameters of CLM search.

2.1.2. Starting solutions

We will compare the performance of this parallel algorithm when the experimenter uses randomly generated solutions for each of the processes or guided starting solutions. Guided solutions are generated using a constructive heuristic based in the well known one, NEH by Nawaz, Enscore and Ham (1983). This heuristic has a good performance in terms of computing time and quality of solutions when applied to the permutation flow-shop problem, see Ruiz and Maroto (2005) or Rad et al. (2006).

2.2. Computational experience

We have carried out at least 30 experiments with each of the problems of well known Taillard (1993) benchmark from 20 jobs and 5 machines to 100 jobs and 20 machines in two systems. The first one is a network of workstations. The nodes of the cluster are workstations with one Intel Pentium IV processor each, running at 3.2 GHz. The operating system is Linux, Debian distribution. The parallel algorithm has been implemented using LAM-MPI library for message passing, Pacheco (1997). The second platform is a workstation with an Intel Quad Core processor running at 2.5 GHz. The operating system is Linux with the Debian based distribution Ubuntu.

The initial values for the parameters of the algorithm are shown in (table 1). These values have been selected taking into account the values proposed in the original sequential CLM algorithm from Ghosh and Sierksma (2002) and in other parallel local search algorithms, mainly the one by Niar and Freville (1996).

Table 1. Initial values of the parameters

Parameter	Value	Comments
α_0	0.1	Initial value of threshold parameter
β	0.1	Threshold parameter
K	2	Number of solutions which neighborhood is explored in each worker iteration
Pool_max	0.2	Max. size of central pool as a percentage of neighborhood size, also used in workers
Max_glob_it	20	Iterations in master process
B	5	Number of solutions in each communication message

In the original sequential CLM algorithm, the solutions found in the neighbourhood of another one, are stored only if their quality is near the best one. This difference of qualities is computed by a threshold (2).

$$\tau = (1+\alpha)mk \quad (2)$$

Being *mk* the best value of makespan found till that iteration. α is multiplied in each iteration by a factor λ (3) and so α value decreases in each iteration to intensify the search in the neighbourhood of good quality solutions.

$$\lambda = \alpha_0 / (1+\beta) \quad (3)$$

The results have been analyzed and compared using MS Excel datasheet. First of all are the results of ARPD (4).

$$ARPD = \frac{obtained\_makespan - top\_makespan}{top\_makespan} \quad (4)$$

Table 3. ARPD results with guided initial solutions in a multicore workstation

ARPD WS guided	np=2	np=3	np=4	np=5	np=6	Avg
20x5	1.3538%	1.1432%	0.8724%	0.9627%	0.8424%	**1.0349%**
20x10	1.8366%	1.9124%	1.7543%	1.9052%	1.9302%	**1.8677%**
20x20	1.6616%	1.6132%	1.5207%	1.6216%	1.5063%	**1.5847%**
50x5	0.1986%	0.2013%	0.1900%	0.1683%	0.2044%	**0.1925%**
50x10	3.1613%	3.1420%	3.1732%	3.2327%	3.2457%	**3.1910%**
50x20	4.0468%	4.1411%	4.0911%	4.1505%	4.2476%	**4.1354%**
100x5	0.2272%	0.2336%	0.1936%	0.1803%	0.1985%	**0.2066%**
100x10	1.3169%	1.3421%	1.2991%	1.3150%	1.3634%	**1.3273%**
100x20	3.6853%	3.9016%	3.9264%	3.9617%	4.1534%	**3.9257%**
Avg	**1.9431%**	**1.9589%**	**1.8912%**	**1.9442%**	**1.9658%**	**1.9406%**

The first results (table 3) are the average ARPD obtained in a workstation using guided initial solutions and the second (table 4) are the average ARPD obtained using randomly generated starting solutions. All of the results are grouped by the size of the problem (*jobs* x *machines*) and by the number of processes (*np*) employed in the parallel algorithm, from 2 to 6 processes. In the cluster, each process runs in a processor. In the case of the workstation, when the number of processes exceeds the number of cores, more than one process must run in the same core of the processor. This will decrease the performance of the algorithm, increasing the computing time when more than 4 processes are used.

Table 4. ARPD results with random initial solutions in a multicore workstation

ARPD WS random	np=2	np=3	np=4	np=5	np=6	Avg
20x5	4.1742%	6.4531%	6.7464%	6.2274%	7.5361%	**6.2274%**
20x10	6.9566%	7.8909%	8.6623%	9.0719%	10.9512%	**8.7066%**
20x20	5.5682%	6.2334%	6.6742%	7.4816%	7.6633%	**6.7242%**
50x5	1.6663%	2.6585%	3.3929%	3.9844%	4.0905%	**3.1585%**
50x10	6.5498%	9.3432%	10.8130%	11.4835%	12.1235%	**10.0626%**
50x20	7.9090%	11.7203%	12.7295%	13.9283%	14.4304%	**12.1435%**
100x5	3.5969%	4.0316%	4.4957%	5.0097%	4.8146%	**4.3897%**
100x10	4.4277%	7.5766%	8.4953%	9.4098%	9.9477%	**7.9714%**
100x20	9.1623%	12.8269%	14.9119%	14.9302%	15.3706%	**13.4404%**
Avg	**5.5568%**	**7.6372%**	**8.5468%**	**9.0586%**	**9.6587%**	**8.0916%**

Table 5. ARPD results with guided initial solutions in a cluster of workstations

ARPD Cluster guided	np=2	np=3	np=4	np=5	np=6	Avg
20x5	1.4531%	1.4295%	1.3001%	1.4299%	1.2916%	**1.3808%**
20x10	2.0420%	2.1399%	1.9581%	2.1437%	1.9990%	**2.0565%**
20x20	2.0264%	2.0700%	1.7636%	1.9627%	1.6677%	**1.8981%**
50x5	0.2761%	0.3354%	0.2607%	0.2526%	0.2453%	**0.2740%**
50x10	3.2509%	3.4289%	3.3048%	3.3828%	3.3834%	**3.3501%**
50x20	4.1752%	4.2735%	4.3212%	4.4443%	4.4545%	**4.3337%**
100x5	0.2012%	0.2883%	0.2156%	0.2477%	0.2033%	**0.2312%**
100x10	1.2408%	1.3427%	1.2614%	1.4167%	1.3838%	**1.3291%**
100x20	3.7051%	3.8784%	4.0851%	3.9219%	4.0271%	**3.9235%**
Avg	**2.0412%**	**2.1318%**	**2.0523%**	**2.1336%**	**2.0729%**	**2.0864%**

We have also run the algorithm using random initial solutions in the cluster with only two of the problem sizes (table 6) to confirm the differences between the quality of solutions obtained by these methods.

Table 6. ARPD results with random initial solutions in a cluster of workstations

ARPD Cluster random	np=2	np=3	np=4	np=5	np=6	Avg
100x10	5.0576%	7.8692%	9.5393%	9.9698%	10.4172%	**8.5706%**
100x20	9.4156%	13.2467%	14.7844%	15.4988%	16.0241%	**13.7939%**
Avg	**7.2366%**	**10.5579%**	**12.1619%**	**12.7343%**	**13.2206%**	**11.1823%**

The time in seconds needed to reach the ARPD shown before, using the different configurations of computer platform and algorithm is shown in the following tables. In a workstation using guided starting solutions (table 7) and random ones (table 8). In a cluster using guided starting solutions (table 9) and random ones (table 10).

Table 7. Time results with guided initial solutions in a multicore workstation

Time (s) WS guided	np=2	np=3	np=4	np=5	np=6	Avg
20x5	0.12	0.10	0.08	0.08	0.10	**0.10**
20x10	0.19	0.17	0.16	0.16	0.15	**0.17**
20x20	0.37	0.30	0.28	0.27	0.24	**0.29**
50x5	14.73	8.47	7.23	6.18	6.24	**8.57**
50x10	15.79	14.25	11.87	12.08	11.48	**13.09**
50x20	42.76	34.79	29.83	25.54	26.28	**31.84**
100x5	232.82	143.33	117.44	110.52	118.63	**144.55**
100x10	913.77	419.31	364.19	357.12	306.34	**472.15**
100x20	2602.64	1441.32	918.24	913.20	933.29	**1361.74**
Avg	**424.80**	**229.12**	**161.04**	**158.35**	**155.86**	**225.83**

Table 8. Time results with random initial solutions in a multicore workstation

Time (s) WS random	np=2	np=3	np=4	np=5	np=6	Avg
20x5	0.06	0.05	0.07	0.05	0.06	**0.06**
20x10	0.12	0.11	0.12	0.12	0.12	**0.12**
20x20	0.24	0.21	0.19	0.21	0.21	**0.21**
50x5	19.77	10.81	9.05	7.78	7.36	**10.96**
50x10	42.80	23.58	18.13	16.05	15.39	**23.19**
50x20	89.06	45.87	39.03	31.82	31.07	**47.37**
100x5	622.70	295.52	231.92	196.22	208.05	**310.88**
100x10	1756.85	860.72	668.97	550.97	543.35	**876.18**
100x20	3724.16	1744.63	1245.18	1144.25	1157.02	**1803.05**
Avg	**695.09**	**331.28**	**245.85**	**216.39**	**218.07**	**341.33**

Table 9. Time results with guided initial solutions in a cluster of workstations

Time (s) Cluster guided	np=2	np=3	np=4	np=5	np=6	Avg
20x5	0.10	0.06	0.05	0.04	0.04	0.06
20x10	0.16	0.10	0.09	0.07	0.07	**0.10**
20x20	0.32	0.21	0.17	0.14	0.14	**0.20**
50x5	21.08	9.99	8.11	6.96	5.60	**10.35**
50x10	34.21	23.22	17.25	14.18	11.33	**20.04**
50x20	78.10	47.27	38.62	31.22	25.42	**44.12**
100x5	332.13	243.07	185.31	167.23	134.84	**212.52**
100x10	1121.54	880.53	661.34	603.44	415.44	**736.46**
100x20	3929.37	2615.31	1623.97	1334.16	1145.69	**2129.70**
Avg	**613.00**	**424.42**	**281.66**	**239.72**	**193.17**	**350.39**

Like in the study of makespan, we present here (table 10) the execution time in seconds for the algorithm using random initial solutions running in the cluster with only two of the problem sizes to confirm the differences between the execution time needed by these methods.

Table 10. Time results with random initial solutions in a cluster of workstations

Time (s) Cluster random	np=2	np=3	np=4	np=5	np=6	Avg
100x10	2489.37	1307.97	912.71	676.22	555.43	**1188.34**
100x20	5634.25	2942.56	1871.53	1556.40	1308.51	**2662.65**
Avg	**4061.81**	**2125.27**	**1392.12**	**1116.31**	**931.97**	**1925.50**

We have also carried out a statistical study to validate the observed differences between implementations and platforms. First, we have compared ARPD variances between the workstation and the cluster using guided starting solutions. The F-test suggests the variances are equal in most of problem sizes and number of processes employed with a confidence level of 95%. This is due to the constructive nature of the starting point of this method. In this case, the average ARPD is different (table 11) with a confidence level of 95% only in some of the greater size problems.

Table 11. Differences between ARPD when using a workstation and a cluster and guided starting solutions

Differences	np=2	np=3	np=4	np=5	np=6
20x5	=	=	=	=	=
20x10	=	=	=	=	=
20x20	=	=	=	=	=
50x5	=	=	=	=	=
50x10	=	=	=	=	=
50x20	=	=	=	=	=
100x5	≠	=	=	=	=
100x10	≠	=	≠	=	=
100x20	=	≠	=	≠	≠

The F-test confirmed that the variances of ARPD using different starting methods when the algorithms run in the same computer are different with a confidence level of 95%. So we have employed a t-test to confirm the differences between the averages of ARPD obtained. With this test we can conclude that the quality of solutions obtained by a local search parallel algorithm improves, using the same number of iterations, when the starting point of the search is guided by a constructive heuristic instead of using a random starting solution.

The differences between execution time are greater in most of the cases. The time needed using guided starting solutions is generally lesser than the one using random starting solutions. This difference has been observed using a cluster or an individual computer. The only point is that the behavior is the opposite when problem size is smaller than 50 jobs. In this case, the time needed to construct the starting solution is greater than the time needed for the algorithm to improve it.

We have also noticed a better performance in terms of execution time of a today's workstation over the performance of a previous generation cluster of workstations when using similar number of processes to run a parallel local search algorithm. The different time is due to communication delays. The cluster needs to make use of a gigabit Ethernet connection between its nodes while the individual computer only uses its internal bus between memory and multicore CPU. The external network inserts latency and less speed than the internal bus. The difference is greater when using 4 processes, the number of cores of the workstation. With less processes, the communication is also lesser and with more than 4 processes, there are more than one running in each of the cores and so the performance of the cluster begins to take advantage of its greater number of processors.

3. Conclusions

We have observed differences between the behavior of our parallel algorithm not only due to the method used to generate starting solutions but also due to the computer environment. The quality of obtained solutions is better using guided initial solutions than using random initial solutions, also in parallel algorithms. The time used by the Quad Core workstation is better than the obtained using six nodes of a cluster except when the number of processes is greater than the number of cores. Even in this case the observed processing times are similar, so nowadays the advances in hardware makes it useful to use parallel computing even in personal computers. As a future research line we could check if the presented conclusions are applicable to other parallel algorithms or to other optimization problems.

References

Bożejko, W. (2009). Solving the flow shop problem by parallel programming. Journal of Parallel and Distributed Computing. Vol. 69, pp. 470–481.

Bożejko, W.; Wodecki, M. (2008a). Parallel scatter search algorithm for the flow shop sequencing problem. PPAM 2007. 7th International Conference on Parallel Processing and Applied Mathematics. Wyrzykowski, R.; Szymanski et B. Ed., LNCS, Springer. To appear.

Bożejko, W.; Wodecki, M. (2008b). Parallel path-relinking method for the flow shop scheduling problem. International Conference on Computational Science (ICCS 2008). LNCS. Springer. To appear.

Bożejko, W.; Wodecki, M. (2004a). The New Concepts in Parallel Simulated Annealing Method. L. Rutkowski *et al.,* Eds.): ICAISC 2004, LNAI 3070, pp. 853–859.

Bożejko, W.; Wodecki, M. (2004b). Parallel tabu search method approach for very difficult permutation scheduling problems. Proceedings of International Conference on Parallel Computing in Electrical Engineering: PARELEC '04, pp. 156–161.

Bożejko, W.; Wodecki, M. (2002). Solving the flow shop problem by parallel tabu search. Proceedings of International Conference on Parallel Computing in Electrical Engineering, 2002. PARELEC '02, pp. 189–194.

Crainic, T.G.; Gendreau, M. (2002). Cooperative Parallel Tabu Search for Capacitated Network Design. Journal of Heuristics. Vol. 8, pp. 601–627.

Ghosh, D.; Sierksma, (2002). Complete Local Search with Memory. Journal of Heuristics, Vol. 8, No. 6, 571-584.

Kohlmorgen, U.; Schmeck, H.; Haase, K. (1999). Experiences with fine-grained parallel genetic algorithms. Annals of Operations Research. Vol. 90, pp. 203–219.

León Blanco, J.M., Framiñán Torres, J.M., González Rodríguez, P.L., Pérez González, P., Ruiz Usano, R. (2006). Influencia de la Granularidad en las Prestaciones de Algoritmos Paralelos. X Congreso de Ingeniería de Organización. Valencia, pp. 217-218.

Nawaz, M.; Enscore Jr., E.E.; Ham, I. (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem. Omega. Vol. 11, no. 1, pp. 91–95.

Niar, S.; Freville, A. (1996). A Parallel Tabu Search Algorithm For The 0-1 Multidimensional Knapsack Problem. Baltzer Journals. ISSN 1063-7133/97.

Okamoto, S.; Watanabe, I.; Iizuka, H. (1995). A new parallel algorithm for the n-Job, m-machine flow-shop scheduling problem. Systems and Computers in Japan (Wiley Periodicals, Inc.). Vol. 26, no. 2, pp. 10–21.

Okamoto, S.; Watanabe, I.; Iizuka, H. (1994). A new optimal algorithm for the permutation flow-shop problem and its parallel implementation. Computers & Industrial Engineering. Vol. 27, no. 1–4, pp. 39–42.

Pacheco, P.S. (1997). Parallel Programming with MPI. Morgan Kaufmann Publishers, Inc. / Elsevier.

Rad, S.F., Ruiz, R., Boroojerdian, N. (2009). New High Performing Heuristics for Minimizing Makespan in Permutation Flowshops. OMEGA, The International Journal of Management Science. Vol. 37, no. 2, pp. 331–345.

Ruiz,R., Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research. Vol. 165, pp. 479–494.

Talbi, E.G., Hafidi, Z., Geib, J.M. (1998). A parallel adaptive tabu search approach. Parallel Computing. Vol. 24, pp. 2003–2019.

Wodecki, M.; Bozejko, W. (2002). Solving the Flow Shop Problem by Parallel Simulated Annealing. R. Wyrzykowski et al., Eds.): PPAM 2001, LNCS 2328, pp. 236–244

Zobolas, G.I., Tarantilis, C.D., Ioannou, G. (2008). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Computers and Operations Research. doi: 10.1016/j.cor.2008.01.007.