SCHEDULING IN FLEXIBLE MULTI-SERVER INTERNET SERVICES

Fawaz S. Al-Anzi[1] and Ali Allahverdi[2]

1: Assistant professor, Department of Computer Engineering, Kuwait University
2: Associate professor, Department of Industrial and Management Systems Engineering, Kuwait University.

P.O. Box 5969, Safat, Kuwait, Fax: 965 483-9461, Email: alanzif@eng.kuniv.edu.kw

ABSTRACT

In this paper, we consider the problem of scheduling on a multi-stage parallel-server architecture in computer centers to minimize the total or average completion time of a set of requests. The problem is modeled as a flexible flowshop problem for which few heuristics to minimize total completion time exist in the flowshop scheduling literature. We propose a new heuristic for this problem that consists of three phases. Genetic Algorithm is used in the first phase of the heuristic followed by Diffusion and Infusion greedy algorithms phase. The last phase is a local linear rippling procedure. An extensive computational experiment has been conducted to compare the existing heuristics with each other for the first time. Moreover, the proposed heuristic is compared with the existing heuristics. The results indicate that the proposed heuristic significantly outperforms the existing ones.

Keywords: *Scheduling, Multi-Server, Multi-Stage, Flexible flowshop, Completion time*

الملخص

في هذه الورقة العلمية نتناول مشكلة جدولة الطلبات لتنفذ على نظام متعدد المراحل حيث تتكون كل مرحلة من عدد من الخادمات المتوازية في مراكز الحاسوب بغرض تقليل الزمن الكلي المستغرق لتنفيذ الطلبات ، يتم تمثيل المشكلة بصورة الجدولة المرنة في الهندسة الصناعية حيث توجد عدد من الخوارزميات في محاولات سابقة لحلها، يتم عرض خوارزمية جديدة لحل المشكلة تتكون من ثلاثة مراحل (مرحلة خوارزمية الهندسة الجينية ، مرحلة الخوارزميات الطموحة ، مرحلة التحسين الخطي المحلي) ، كما يتم فحص و مقارنة اداء الخوارزميات السابقة مع بعضها و كذلك مع الخوارزمية الجديدة ، النتائج تدل على تفوق اداء الخوارزمية الجديدة على نظيراتها السابقة.

1. INTRODUCTION

One of the important factors in the determining the quality of an internet service is the average response time of the requests. Typically an internet service requires the processing of a request on a multi stage architecture where each stage usually consists of a pool of parallel servers performing the same operation. The number of parallel servers in a stage is determined by how critical this particular stage is to the overall quality of the service. This architecture is widely used in the internet since it provides flexibility in scalability and robustness.

When a service request arrives to such architecture it follows a predetermined order of different stages of the service. Usually the amount of processing time of a request can be known before hand. At any point of time, there is usually a set of service requests waiting to be processed. The order of providing the service to the available set of requests significantly affects the average response time of the service. Therefore, it is empirical to determine the best order for processing this set of requests.

This problem is precisely the same as the flexible multi-stage flowshop problem with the objective of minimizing the average completion time. This problem has been addressed in the literature in the context of machine scheduling, and it is known to be NP-hard. Therefore, implicit enumeration techniques, such as the branch-and-bound, have been proposed for a small number of jobs, see Linn and Zhang (1999). For larger number of jobs, efficient heurists are required. The literature survey reveals that Sridhar and Rajendran (1993), Brah and Loo (1999), and Azizoglu et al. (2001) presented different heuristics. The problem has also been addressed in the context of machine scheduling for other performance measures such as makespan, which denotes the completion time of the last request. This performance measure is useful in the context of multi-server internet services for situations where the set of requests are parts of a single transaction where the transaction cannot be considered complete unless all the requests are successfully executed. Recent work with makespan performance measure includes Cheng et al. (2000), Choi and Lee (2000), Moursli and Pochet (2000), Grabowski and Pempera (2000), Gendreau et al. (2001), and Negenman (2001).

In this paper, we propose a new heuristic for the problem with the total completion time performance measure. The proposed heuristic consists of three phases. Genetic Algorithm is used in the first phase of the heuristic followed by Diffusion and Infusion greedy algorithms phase. The last phase is a local linear rippling procedure. An extensive computational experiment has been conducted to compare the existing heuristics of Sridhar and Rajendran (1993), Brah and Loo (1999), and Azizoglu et al. (2001) with each other for the first time. Moreover, the proposed heuristic is compared with the existing

Top

heuristics. The results indicate that the heuristic by Sridhar and Rajendran (1993) is practically the best among the existing ones. The results also indicate that the proposed heuristic significantly performs better than the existing ones in terms of the average error, the number of best obtained solutions, and the standard deviation of the error.

2. PROBLEM DEFINITION

Consider the configuration where there are *m* stages. Stage j (j=1,2,…,m) consists of m_j identical parallel servers. There is a set of *n* requests available for scheduling. Every request *i* is to be processed by one of the servers in stage *1* then one of the servers in stage *2* and so on until it is processed by one of the servers in stage *m*. If all the servers at a stage are busy, then a request has to wait until one of the servers at that stage becomes available. A server at a given stage can only process one request at a time and no preemption is allowed.

Let $t_{i,j}$ and $C_{i,j}$ denote the processing time and completion time of request in position *i* at stage *j*, respectively. Also, let $A_j(t)$ denote the earliest time that a server in stage *j* is available at time t or later. Then, the completion time of request i at stage j can be computed as

$$C_{i,j} = \max\{C_{i,j-1}, A_j(C_{i,j-1})\} + t_{i,j}$$

where $C_{i,0} = 0$. The total completion time (TCT) of all requests can be expressed as

$$TCT = \sum_{i=1}^{n} C_{i,m}$$

The objective is to find a schedule of requests that minimizes TCT.

3. EXISTING HEURISTICS

The flexible multi-stage flowshop problem is the generalization of the multi-stage flowshop problem where there is only one server at each stage. It is known that even the two-stage flowshop problem with the total completion time performance measure is NP-hard, see Gonzalez and Sahni (1978). Therefore, the problem addressed in this paper is also NP-hard demonstrating the need for efficient heuristics. The literature review reveals that Sridhar and Rajendran (1993), Brah and Loo (1999), and Azizoglu et al. (2001) are the only ones establishing heuristics for our problem. In the following, the existing heuristics are described.

Top

3.1. ACK heuristic

Azizoglu et al. (2001) presented a branch-and-bound algorithm to find an optimal solution. They proposed three heuristics to find an upper bound for their branch-and-bound algorithm as following:

Heuristic 1. Stages are considered separately. At each stage, the problem is solved by using the Shortest Processing Time rule. The sequences found are imposed over all stages and the sequence that gives the minimum total completion time is selected.

Heuristic 2. The same as Heuristic 1 except that the sequence is only used at the first stage. For the other stages, among the available requests assign the request with the shortest processing time to the earliest available server.

Heuristic 3. The requests are ordered in a sequence according to a sequence proposed by Rajendran and Chaudhuri (1991) of a non-decreasing order of $\sum_{j=1}^{m}(m-j+1)t_{i,j}$. Requests are assigned to the earliest available server at each stage.

In this paper, all the three heuristics are evaluated and the best of the three is chosen. This is denoted as ACK heuristic.

3.2. HO heuristic

Brah and Loo (1999) have applied the existing heuristics designed for the regular flowshop problems to flexible flowshop problems with different performance measures including total completion time. They found that the heuristic by Ho (1995) performed the best among the others with respect to total completion time minimization. The HO heuristic is as follows:

1. Let i=n-4, and k=1
2. Obtain an initial sequence by arranging the requests in ascending order of $\sum_{j=1}^{m}(m-j+1)t_{i,j}$
3. Sort the initial solution using the bubble sort and call it the current solution.
4. Set Z_1=TCT
5. a. Sort the current solution by the insertion sort
   b. Sort the current solution by the non-adjacent pair-wise interchange method
6. Sort the current solution by the bubble sort
7. Set Z_2=TCT
8. If k<i, and $Z_1 \neq Z_2$ then k=k+1, and go to Step 4, otherwise the current solution is the final solution.

The insertion and bubble sort methods are well known sorting algorithms in computer science, see Kunth (1973).

Top

3.3. SR heuristic

Sridhar and Rajendran (1993) were the first to propose a heuristic for the problem. They proposed a three phase heuristic. An initial seed sequence is obtained in the first phase by using the Rajendran and Chaudhuri (1991) algorithm. This seed sequence is used as a starting sequence for the Simulated Annealing algorithm. Then, finally a local adjacent interchange scheme is applied. Their heuristic is referred to as SR Heuristic in this paper.

4. PROPOSED HEURISTIC (AA)

The proposed heuristic AA is described in this section. It consists of three phases. In phase one, an initial sequence is obtained by applying a genetic based algorithm. In phase two, the sequence obtained from phase one is improved by a repeated application of diffusion and infusion greedy algorithms. Finally, in phase three, the sequence acquired from phase two is further improved by a local linear rippling procedure. The description of each phase of AA heuristic is explained next.

Phase I: Genetic Algorithm

```
1. Initialize a population POP of random sequences
2. Compute the Total Completion Time (TCT) of each sequence in POP
3. Order the sequences in POP according to TCT from best to worst
4. Repeat Steps i to v for GEN times
     i. Repeat Steps a to d  for CP times
          a. Randomly choose two different compatible parents to mate
          b. Select compatible segments in the two parents
          c. Swap the segments
          d. Save the new sequences in CHILD and compute TCT of each
     ii. Order CHILD with respect to TCT
     iii. Replace the worst y sequences of POP with the best in
          CHILD sequences
     iv. Mutate each sequence in POP with the probability of p
     v. Compute TCT and order POP
5. Select the best solution $\pi_1$ from POP, and set it as the current sequence
```

POP is the population of sequences. CHILD is the population of sequences generated from POP in a single iteration. GEN is the total number of generations, CP is the number of times to search for child sequences from compatible parents to generate CHILD population, and p is the mutation probability.

Phase II: Diffusion and Infusion Algorithms

Here we describe the two greedy algorithms of phase II separately.

Diffusion Algorithm

1. Given an input sequence of n requests.
2. Set k=1 and current solution to be empty
3. Generate k candidate sequences by diffusing the current solution into two parts at all possible k positions and inserting the k^{th} request in the diffused sequence.
4. Compute partial TCT for all k candidate sequences. Among these candidates, select the one with the least TCT.
5. Update the one with the least TCT as a current solution.
6. Update k = k+1
7. If k = n + 1 then stop, else go to Step 3.

The Diffusion Algorithm described above was first used by Nawaz et al. (1983) for the makespan problem on m-machines and n-jobs which is a different problem and a different performance measure than the one we are considering here.

Infusion Algorithm

Our Infusion algorithm makes $\log_2(n)$ passes on the list of requests. Here, we assume that n is multiple of 2's. If not, we increase n to the closest multiple of 2 and fill the rest of request slots by zero processing time requests which will be removed for the final request sequence. In pass number i, the list is evenly divided into subsequences, each of size 2^{i-1}. For every two subsequences j and j+1 that are to be infused, we iterate the infusion of every element in the subsequence j+1 into the subsequence j maintaining the best sequence that has the least total completion time in the produced subsequences.

1. Given an input sequence of n requests.
2. Set the input sequence as current sequence
3. For i = 1, ..., $\log_2(n)$ Repeat Steps 4-10.
4. Divide the current sequence into subsequences of size 2^{i-1}.
5. For every two subsequences j and j+1 that will be infused by repeating Steps 6-9.
6. For every request x in subsequence j+1 from left to right.
7. Generate candidate sequences by infusing the x request into each possible slot of the j subsequence.
8. Compute the partial TCT for each candidate sequence. Among these candidates, select the one with the least TCT.
9. Update the one with the least TCT as the j sequence.
10. Join all subsequences resulting in Steps 7-9 into one sequence and let it be the current sequence.

Top

Phase III: Local Linear Rippling Procedure

```
1. Set i=1
2. Compute TCT of the current sequence, and call it $TCT_1$
3. Exchange the requests in position i and i+1
4. Compute TCT of the current sequence, and call it $TCT_2$
5. If $TCT_2<TCT_1$, then go to Step 7
6. Exchange the requests in position i and i+1
7. Set i=i+1
8. If i<n, then go to Step 2
```

Notice that the policy of assigning a sequence of the requests at each stage can be one of two policies. The first policy is to enforce the initial sequence (the sequence fed to stage one) in all the subsequent stages. The second policy is that a request departing from a previous stage is assigned to the earliest available server of the current stage. If there is more than one request available when a server is free, the priority is given to the one with the shortest processing time. In our proposed heuristic we adopt the second policy.

Setting the parameters for the genetic algorithm is essential in achieving a good performance for obtaining a good sequence with a reasonable time for phase one. It is known fact that parameters for a genetic algorithm are problem dependent and may not work well for different problem sets. After an extensive computational analysis, the parameters for our problem are set as given in Table 1.

Table 1. Parameters of the genetic algorithm

Parameter	Value
POP	5n
GEN	3n
CP	2n
y	50%
p	0.0135

In Phase II, the greedy algorithms are applied in the sequence of Diffusion-Infusion-Diffusion repeatedly. This process is repeated for five times since no significant improvement was observed beyond this. It should be noted that there is no guarantee that each time a better solution will be obtained. Therefore, among the five repetitions the best one is always kept as the current solution. Experimentation has shown that the proper sequence of greedy algorithms is Diffusion-Infusion-Diffusion since other permutations did not perform as well. Experimentation has also shown that using Diffusion or Infusion alone did not perform well.

Similarly Phase III is repeated and it has been observed that five repetitions of the local linear rippling procedure were sufficient to produce a good solution.

5. HEURISTIC COMPARISON

The three existing heuristics of ACK, HO, and SR along with the proposed heuristic AA were implemented in C on a Sun Sparc 20, and evaluated with respect to average error, standard deviation of the error, and the number of times yielding the best solution.

The processing times were randomly generated from a uniform distribution (1, 100). In the scheduling literature, most researchers have used this distribution in their experimentation, e.g., Wang et al. (1997), Pan and Chen (1997), Al-Anzi and Allahverdi (2001), and Allahverdi and Al-Anzi (2002). The reason for using a uniform distribution is that the variance of this distribution is large and if a heuristic performs well with such a distribution, it will certainly perform well with any other distribution.

Problem data are generated for different number of requests for the range of 20 to 100 in increment of 10. The number of stages is considered to be 2 to 6 while at each stage the number of parallel servers is randomly generated from a uniform discrete distribution 1 to 5. We compare the performance of the heuristics for thirty replicates using three measures: average percentage error (Error), standard deviation (Std), and the number of times the best solution is obtained (NBS). The percentage error is defined as 100* (Heuristic – Best Solution)/Best Solution.

Figures 1-6 show the results of running the existing and the proposed heuristics for all combinations of the number of requests and the number of stages. Each entry in the tables represents the average of the thirty replicates where for every replicate a configuration of the processing time and the number of servers per stage are generated randomly. Each heuristic is evaluated for the same configuration to ensure accurate assessment of the different heuristics. The results for HO are not reported for the number of requests greater than 40 because of its large CPU time requirement. For example, for 40 requests and 6 stages, the CPU times (Sec) of ACK, HO, SR, and AA were 0.029, 281, 32, and 39, respectively. The CPU time of HO increases significantly as the number of requests increases while the CPU times of the other heuristics increase moderately. The same can be said about the number of servers.

Figures 1-3 illustrate the performance of the heuristics with respect to the number of requests. Observe that the heuristic ACK is not included in Figures 1 and 2 because of its

Top

poor performance and to make the comparison of the others clearer. It is clear that among the existing heuristics in the literature SR and HO significantly outperform ACK. Even though HO outperforms SR (at least for the number of requests less than 40), the heavy CPU time requirement of HO makes it impractical to use it for larger problems. Hence, SR can be considered practically the best among the existing ones. As can be seen from Figure 1, the average error of the proposed algorithm AA gets better as the number of requests increases while that of SR deteriorates. The same can be said for the other performance measures (standard deviation and the number of best solutions) for the AA and SR heuristics. Therefore, the proposed AA heuristic is superior to the SR heuristic.

Figures 4-6 illustrate the performance of the heuristics with respect to the number of stages. Again, ACK is eliminated from Figures 4 and 5 due to its poor performance. In general, for all number of stages, the proposed AA heuristic outperforms all the existing heuristics in terms of all the three performance measures. It can be seen that the performance measures deteriorates as the number of stages increases for all the heuristics. However, the deterioration of SR is more significant.

The overall average of Error of ACK, HO, SR, and the proposed heuristic AA are 34.1, 0.32, 0.9, and 0.27, respectively. It should be noted that the performance of all the four heuristics with respect to the number of best solution and the standard deviation is similar to that of the average error. The overall average of NBS of ACK, HO, SR, and proposed heuristic AA are 0, 58.22, 21.11, and 73.41 while the overall average Std are 10.15, 0.53, 0.77, and 0.41, respectively.

6. CONCLUDING REMARKS

The problem of scheduling on a multi-stage parallel-server architecture in computer centers is addressed in this paper with respect to the total completion time of a set requests. This problem can be modeled as a flexible flowshop problem. The flexible flowshop literature review reveals that there exist few heuristics to minimize total completion time. In this paper, we compare the existing heuristics with each other. Moreover, we propose a new heuristic for this problem that consists of three phases; Genetic Algorithm, Diffusion and Infusion greedy algorithms, and a local linear rippling procedure. An extensive computational experiment has been conducted to compare the existing and proposed heuristics. The results indicate that the proposed heuristic significantly outperforms the existing ones.

Top

ACKNOWLEDGEMENTS

This research was supported by Kuwait University Research Administration project number EE 01/01.

REFERENCES

1. Al-Anzi, F. and Allahverdi, A., 2001, “The relation between three-tired client-server internet database and two-machine flowshop,” International Journal of Parallel and Distributed Systems and Networks, 4, pp 94-101.
2. Allahverdi, A. and Al-Anzi, F., 2002, “Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications,” Computers and Operations Research, 29, pp 971-994.
3. Azizoglu, M., Cakmak, E., and Kondakci, S., 2001, “A flexible flowshop problem with total flow time minimization,” European Journal of Operational Research, 132, pp 528-538.
4. Brah, S.A., and Loo, L.L., 1999, “Heuristics for scheduling in a flow shop with multiple processors,” European Journal of Operational Research, 113, pp 113-122.
5. Cheng, T.C.E., Lin, B.M.T., and Toker, A., 2000, "Makespan minimization in the two-machine flowshop batch scheduling problem,” Navel Research Logistics, 47, pp 128-144.
6. Choi, A.H., and Lee, J.S.L., 2000, "A sequence algorithm for minimizing makespan in multi-part and multi-machine flowshop case," Integrated Manufacturing Systems, 11, pp 62-73.
7. Gendreau, M., Laporte, G., and Guimaraes, E.M., 2001, "A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times," European Journal of Operational Research, 133, pp 183-189.
8. Gonzalez, T., and Sahni, S., 1978, “Flow shop and job shop schedules,” Operations Research, 26, pp 36-52.
9. Grabowski, J., and Pempera, J., 2000, "Sequencing of jobs in some production system," European Journal of Operational Research, 125, pp 535-550.
10. Ho, J.C., 1995, “Flowshop sequencing with mean flowtime objective,” European Journal of Operational Research, 81, pp 571-578.
11. Jansen, K., Mastrolilli, M. and Solis-Oba, R., 2000, “Approximation algorithms for flexible job shop problems”, Proceedings of Latin American Theoretical Informatics (LATIN), pp. 68-77.
12. D.E. Knuth, 1973, The Art of Computer Programming: Sorting and searching, Addison-Wesley, Reading, MA, USA.
13. Linn, R., and Zhang, W., 1999, “Hybrid flowshop scheduling: A survey,” Computers and Industrial Engineering, 37, pp 57-61.
14. Moursli, O., and Pochet, Y., 2000, "A branch-and-bound algorithm for the hybrid flowshop," International Journal of Production Economics, 64, pp 113-125.

Top

15. Nawaz, M., Enscore, E. and Ham, I., 1983, “A heuristic algorithm for m-machine, n-job flow-shop sequencing problem”, Omega, 11, pp 91-95.
16. Negenman, E.G., 2001, "Local search algorithms for the multiprocessor flow shop scheduling problem", European Journal of Operational Research, 128, pp 147-158.
17. Pan, C.H., and Chen, J.S., 1997, “Scheduling alternative operations in two-machine flow-shops,” Journal of the Operational Research Society, 48, pp 533-540.
18. Rajendran, C., and Chaudhuri, D., 1991, “An efficient heuristic approach to the scheduling of jobs in a flowshop,” European Journal of Operational Research, 61, pp 318-325.
19. Rajendran, C., and Chaudhuri, D., 1992, “A multi-stage parallel-processor flowshop problem with minimum flowtime,” European Journal of Operational Research, 57, pp 111-122.
20. Sridhar, J., and Rajendran, C., 1993, “Scheduling in a cellular manufacturing system: a simulated annealing approach,” International Journal of Production Research, 31, pp 2927-2945.
21. Stutzle, T., 1998, “An ant approach to the flow shop problem”, Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT’98), pp. 1560-1564.
22. Wang, M.Y., Sethi, S.P., and Van De Velde, S.L., 1997, “Minimizing makespan in a class of reentrant shops,” Operations Research, 45, pp 702-712.

Top

Figure 1. Average Error versus n

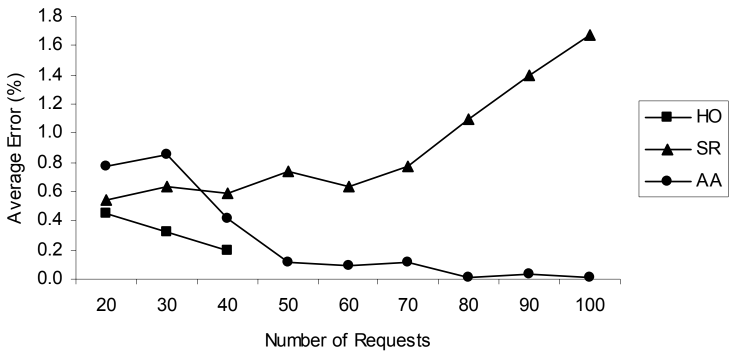


Figure 2. Standard Deviation versus n

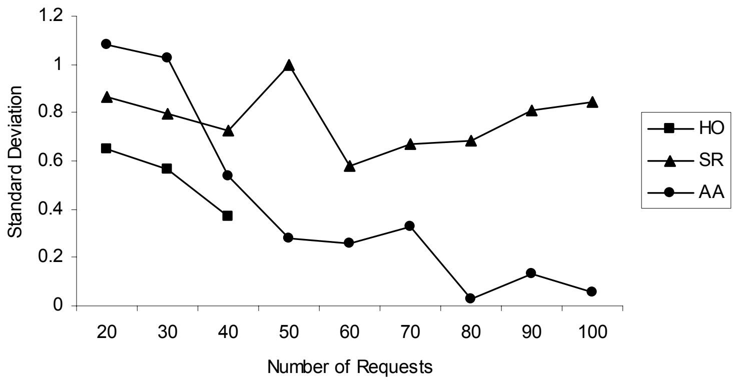
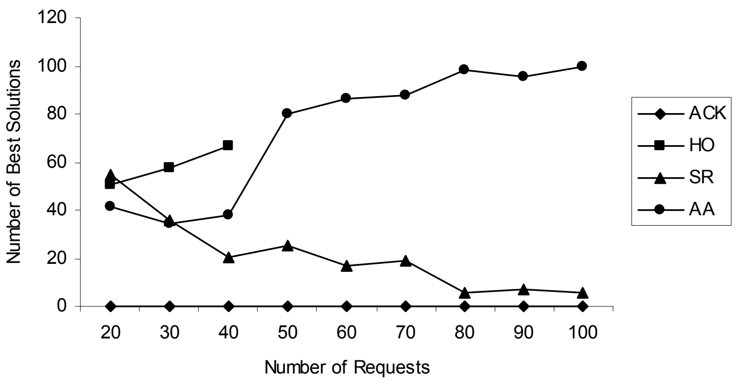


Figure 3. Number of Best Solutions versus n



Top

Figure 4. Average Error versus m

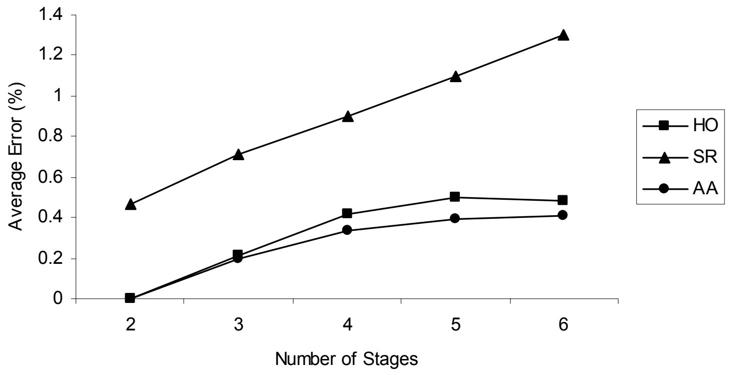


Figure 5. Standard Deviation versus m.

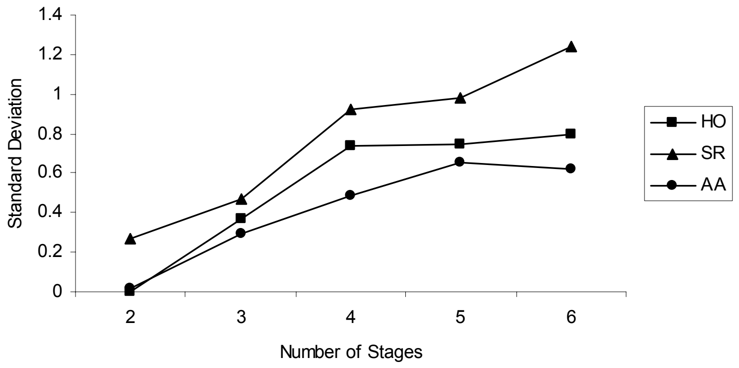
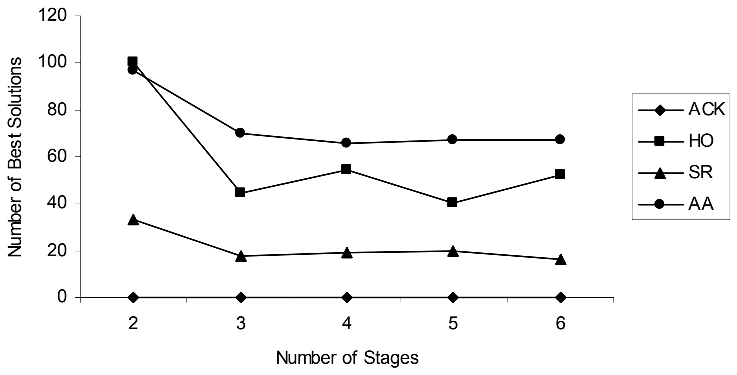


Figure 6. Number of Best Solutions versus m.



Top