A GRASP-Genetic metaheuristic applied on Multi-Processor Task Scheduling Systems

Abla Saad, Ahmed Kafafy, Osama Abd-El-Raof and Nancy El-Hefnawy
{abla.saad, ahmed.kafafi, osamaabd}@ ci.menofia.edu.eg , nancyabbas_1@hotmail.com
Operation Research & Decision Support Dept.- Faculty of computers and information- Menoufia university
Shebin El-Kom, Menoufia, Egypt

***Abstract*— In the last decades, parallel processing systems are used in most applications. In these systems, task scheduling problems are considered as an important issue in managing multiprocessors. The challenge in task scheduling problems is to get the best schedule that achieves the best efficiency and the minimum make span. In this paper, a new hybrid metaheuristic algorithm called GRASP-GA is proposed to handle such problems. In the proposed algorithm, the greedy randomized adaptive search procedure is adopted to construct a population of high quality solutions. Then, the genetic algorithm is applied on the constructed population to improve these solutions. Two heuristic functions are adopted to guide GRASP, bottom-level and top-level. The proposed GRASP-GA is verified against a set of the state-of-the-art algorithms using a set of test problems that considered as benchmarks. The experimental results indicated the superiority of the proposed GRASP-GA over all the tested algorithms. Since, it can achieve the best performance in all test problems used in this experiment. These results ensure the GRASP-GA is a good competitor and can be considered as a viable alternative.**

Keywords—Task scheduling problem; GRASP; GA;

I. INTRODUCTION

Parallel processing systems in most applications is one of the most critical issues in managing and completing the successful parallel programs. The evolution of High-Performance Computing Unit (HPCU) requires an efficient algorithm to execute this big scheduling problem in a reasonable amount of time. Traditional algorithms such as First Come First Served (FCFS), Opportunistic Load Balancing (OLB), Round Robin (RR), Minimum Execution Time (MET), Minimum Completion Time (MCT), Dominant Sequence Clustering (DSC) and Mobility Directed (MD) are time consuming and expensive [3, 4]. As well known, the task scheduling of multiprocessor systems is classified as an NP-hard problem [14]. Due to their capabilities, Metaheuristics based algorithms are considered as efficient approaches for solving many hard optimization problems. This issue excites many researchers to investigate these methods for handling task scheduling problems in multiprocessor systems. Kaur. *et.al.* [5] proposed a Heuristic Genetic Algorithm (HGA) to tackle task scheduling problems. Although HGA algorithm tends to minimize the completion time and increase the throughput of the system, HGA was only tested using one test problem.

Singh *et.al* [6] proposed a genetic algorithm for task scheduling in a heterogeneous parallel multiprocessor system. The proposed GA tries to minimize the finishing time for this schedule, but it consumes more computational time.

Awadall. *et.al* [7] designed a new chromosome representation for Genetic Algorithm. In this representation, task list and processor list are combined into a single chromosome. They called the approach "Task List Processor List Combination based GA (TLPLC). However, TLPLC enhances the processor efficiency and reduces runtime and memory requirements for this implementation, it failed to achieve minimum make span.

Mehrabi. *et.al*. [10] developed a new method to solve the task assignment problem. The developed method depends on loading balancing based on genetic algorithm (GA). Although this method adopts a repair function to achieve a legal chromosome during the process of the algorithm, it is tested using only one benchmark.

Sharma. *et.al* [11] introduced a new genetic algorithm for the multiprocessor-task-scheduling problem. This algorithm adopts a priority of tasks represented by directed acyclic graph (DAG) whose execution depends on the height of task in the graph.

The aim of this paper is to introduce a hybrid metaheuristic approach based on the greedy randomized adaptive search procedures (GRASP) and the genetic algorithm (GA). In this hybrid scheme, the capabilities of GRASP are exploited to generate an initial population of high quality solutions. Then, the Genetic algorithm is adopted to enhance these solutions. The remainder of this paper is organized as follows: the problem definition is introduced in section 2. A quick review on both greedy randomized adaptive search procedure (GRASP) and Genetic algorithms (GA) is presented in sections 3 and 4 respectively. In section 5, the propose approach is presented in detail. Section 6 highlights the experimental design and results. Finally, section 7 adopts the conclusions and the points for further research.

II. PROBLEM DEFINITION

The multiprocessor task scheduling systems can be classified into several classifications depending on the characteristics of the tasks to be assigned and the availability of processors. In this research, we are concerned with the deterministic task scheduling problem. In this case, the number of tasks, the execution time of all tasks, and their precedence are known. The multi-processor environment is homogenous, which means that all processors have the same computing speeds and processing capabilities. In general, the multiprocessor computing environment consists of m homogenous processor [12] as indicated in equation (1).

$$P = \{P_i : i = 1, 2, \dots, m\} \quad (1)$$

The Directed Acyclic Graph (DAG) is considered as a one of the famous methods used to represent the task scheduling problems [1]. In this model, the execution time of the tasks, the relationship among the tasks, and precedence constraints among the tasks are represented. Moreover, the data communication time between any two tasks is non-negligible due to the cost of messages' transition from a task on one processor to another (i.e. communication cost between two tasks at the same processor equals to zero). Furthermore, the multiprocessor system is non-preemptive, which means any processor completes the current task before a new one starts its execution [2]. The task scheduling problem is a bi-objective problem, the first objective function is to minimize the time needed to finish the latest job that called "make span", and the second one is to minimize the sum of finalization time of all tasks called "flow time". The two objectives can be formulated as follows:

$$\text{Makespan: } F_1 = min\{max\{\sum_{i=1}^{m} f_i\}\} \quad (2)$$

$$\text{Flowtime: } F_2 = min \sum_{i=1}^{m} f_i \quad (3)$$

Where, f_i represents the time when the i^{th} task finishes its execution [5]. The constraints of this model can be expressed by using the Directed Acyclic Graph (DAG) as follows:
The task graph $G = \{V, E\}$ include the set of tasks $V = \{t_i : i = 1, 2, \dots, n\}$, a set of edges E which represents the precedence relations among tasks. Each edge e_{ij} represents dependencies between the tasks t_i and t_j (i.e. t_i is the predecessor of t_j and t_j is the successor of t_i). All predecessor tasks must be completed their execution first before their successors begin. According to their dependences relations, the tasks are arranged in different levels. For example, the entry tasks have level 0 and so on [7]. The level emphasizes the precedence relation according to equation (4) as follows:

$$Level(\mathrm{T_i}) = \begin{cases} 0 & \text{if } PRED(T_i) = \emptyset \\ 1 + max\{Level(T_j) : T_j \in PRED(T_i)\}, & otherwise \end{cases} \quad (4)$$

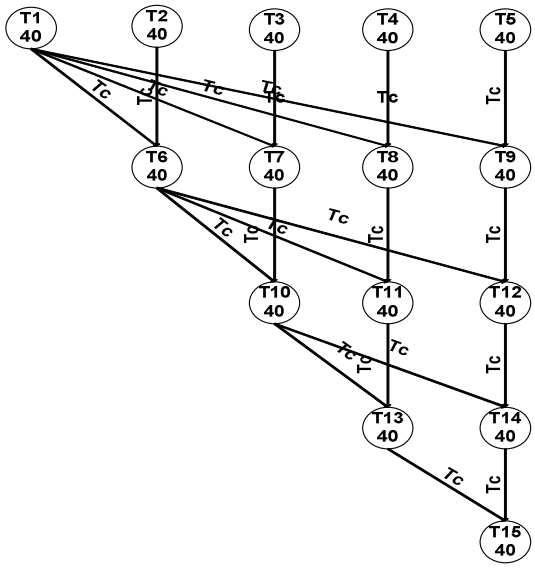

Fig. 1. DAG for Gauss-Jordan algorithm

III. GRASP METAHEURISTIC

The Greedy Randomized Adaptive Search Procedure (GRASP) is a memory less multi-start metaheuristic algorithm which is applied to solve many optimization problems [8,16]. In GRASP procedure, an empty or partial solution is generated then processed to become a complete solution. GRASP algorithm consists of two main phases, construction phase and local search phase. Firstly, in construction phase, a feasible solution is built using a greedy randomized heuristic function. Then, local search is applied to improve the initial constructed solution. The whole GRASP process is explained in some details as follows:

1. Construction Phase:
   This phase begins with an empty or partial solution *S*. Then, each component of *S* is evaluated according to a specific greedy heuristic function *f*. The empty components are arranged based upon their *f* values. After that, the arranged components are selected to complete the empty/partial solution *S* gradually one at a time in a greedy randomized fashion as follows:
   The unselected components are arranged according to their greedy function f values in a restricted candidate list (RCL). each component in RCL has f value according to equation (5) below.

$$f \in [\, f_{min},\ f_{min} + \alpha \times (\, f_{max} - \, f_{min})] \quad (5)$$

   Where, $\alpha \in [0,1]$ is a parameter used to control the balance between greediness and randomness. A component is selected randomly from RCL to be added to *S*. this process is repeated until we have a completed solution (*S*) to pass to the next phase.
2. Local search phase:
   In this phase, the complete solution S is used as a seed for the local search procedure. Here, the neighborhoods of the current solution S are investigated. The most popular two strategies often considered are the *first* and the *best* improvement strategies. In the first improvement, we choose the first neighborhood solution that improves the current solution S. In contrast, in the best improvement, all neighborhoods are investigated to choose the best one. After many iterations, the GRASP finishes and the improved solution is returned [17]. The algorithm below briefs the whole GRASP procedures.

GRASP procedure:

Begin:

1. $S \leftarrow \emptyset$ *//begin with empty solution*
2. $S \leftarrow Construction\,(S, \alpha, f)$.
3. $S^* \leftarrow LocalSearch\,(S, \beta, f)$.

Return S^* *//return the improved solution*

IV. GENETIC ALGORITHM

Genetic Algorithms (GAs) are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. Genetic Algorithms are inspired by Darwin's theory about evolution "survival of the fittest" [19]. They have

been mostly used in many optimization problems [13, 14, 15, 18]. In GAs, chromosome representation plays an important role, as it must be suitable to the problem we have considered. GA operates through a cycle of steps. It begins with initial population of a randomly generated individuals. Each individual represents a solution for the problem on hand. These individuals are evaluated according to a fitness function. After that, genetic operators are applied to improve these individuals. GA adopts three different genetic operators such as selection, crossover and mutation. Firstly, two or more parents' individuals are selected to perform recombination. Then, with a specified probability P_c, crossover is applied on the selected parents for producing new offspring. After that, mutation is applied on the new offspring to make a non-inherit change with a specified probability P_m. Consequently, the new offspring are evaluated according to a fitness function. Finally, the fittest individuals are elected by selection operator to construct a new population for the new generation. This cycle is repeated until stopping criteria are met. The whole process is descried in figure 2.

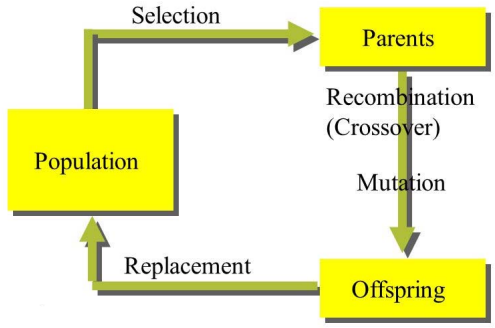

Fig. 2. GA Evolutionary cycle

V. THE PROPOSED APPROACH

In this section, the proposed GRASP-GA is introduced in detail. GRASP-GA is adopted to improve both make-span and flowtime for multiprocessor task scheduling problems. However, the proposed GRASP-GA consists of two basic phases, GRASP phase followed by GA phase. In the first phase, GRASP is responsible for constructing an initial population of high quality solutions. Then, GA operates on the constructed population in the second phase. In the following subsections, each phase will be explained in more details.

A. GRASP Phase

GRASP procedure is responsible for discovering new solutions in the most promising regions of the search space. As explained above, GRASP is two-phases procedure. The first phase is concerned with building a feasible solution using a greedy randomized heuristic. while the second phase performs local search on the solution obtained in the first phase. The key critical issue in GRASP is to design an appropriate heuristic function to the problem, as it controls GRASP performance. The main idea in this research is to adopt both *bottom level* and *top level* proposed in [5] as heuristic functions for GRASP. The *bottom level* (BL) and *top level* (TL) heuristic functions are shown in equations (6) and (7) below.

$$BL(T_i) = W_i + \max_{T_j \in Succ(T_i)} \{C_{ij} + BL(T_j)\} \quad (6)$$

$$TL(T_i) = \max_{T_j \in Pred(T_i)} \{TL(T_j) + W_j + C_{ij}\} \quad (7)$$

Now, we introduce the procedure of GRASP_TS proposed for task scheduling problems. the procedure can use *bottom level* or *top level* as a heuristic function as shown in Alg. 1 below. Firstly, the solution Sol and the candidate list CL are initialized. The procedure determines the heuristic function used whether it is *bottom level* or *top level*. In case of bottom level, the heuristic value of each task T_i is evaluated using equation (6). The Candidate List CL is built by adding each task T_i to CL in descending order according to the heuristic value. Otherwise, each task T_i is evaluated using *top level* as equation (7). Consequently, CL is built by adding each task T_i to CL in ascending order according to the heuristic value.

The Restricted Candidate List (RCL) is composed of the first $\alpha \times |CL|$ elements of CL. The randomness of GRASP occurs when T_i is chosen by chance from RCL. After that, the task T_i is added to Sol. Then, Sol and RCL are therefore updated. This adaptive part of GRASP procedure acts until a feasible solution is built. **Alg.2: GRASP_TS (**α, *HeurFun*)

Inputs:
α: parameter controls greediness /randomness.
HeurFun: The Heuristic function used in GRASP
Output: *Sol*:* best solution found after local search.
Begin: *//begin construction phase*
1. *Sol* ← Ø; CL←Ø; *//initialize Sol & Candidate list*
2. **If** $HeurFun = Bottom\_Level$ *//in case of bottom level*
3. **For** $i \in \{1, \dots, m\}$ **do:**
4. $BL(T_i) = Wi + \max_{Tj \in SUCC(Ti)} \{Cij + BL(Tj)\}$
5. $CL \leftarrow$ CL $\cup$ $\{BL(T_i)\}$. *//construct candidate list*
6. **End For**
7. **ElseIf** $HeurFun = Top\_Level$ *//in case of top level*
8. **For** $i \in \{1, \dots, m\}$ **do:**
9. $TL(T_i) = \max_{T_j \in Pred(T_i)} \{TL(T_j) + W_j + C_{ij}\}$
10. $CL \leftarrow$ CL $\cup$ $\{BL(T_i)\}$. *//construct candidate list*
11. **End For**
12. **EndIf**
13. $CL \leftarrow$ ArrangDesc(CL)*//arrange in descending order.*
14. $RCL \leftarrow$ {the first α × |CL| elements of CL}.
15. **For** $i \in \{1, \dots, |RCL|\}$ **do:**
16. Randomly pick T_i from RCL.
17. $Sol \leftarrow Sol \cup T_i$. *// put the task in the solution*
18. $RCL \leftarrow UpdateRCL(RCL)$*//replace item* T_i *by new one*
19. **End For**
20. **For** j ∈ {1, … , |Sol|} *//begin local search phase*
21. $Sol' \leftarrow Swap(Sol, T_j, T_{j-1})$. *//get new neighborhood sol*
22. $Sol'' \leftarrow Validate(Sol')$ *//validate swap*
23. **If** $F_1(Sol'') < F_1(Sol^*)$ **then**:
24. $Sol^* \leftarrow Sol''$ *//keep Sol with the best make-span*
25. **EndIf**
26. **End For**
27. **End While**
28. **Return** Sol^*; *//return the improved solution*

End

Secondly, local search phase is performed on Sol, the neighbor solution Sol' is obtained by swapping tasks T_j , T_{j-1} from solution Sol. The **validate** process is invoked to validate the

swap operation and to get the solution Sol'' as indicated by Alg.3. if Sol'' has better make-span value (F_1) than Sol^*, then Sol^* is updated by Sol'' to keep the solution with the best make-span F_1. Finally, the best solution Sol^* is returned.

Alg.3: Validate(Sol)**:**

Begin

1. **For** i ∈ {1, … , T} **do:**
2. **IF** ($Level(T_i) > Level(T_{i+1})$) **then:**
3. $Sol \leftarrow Swap(Sol, T_i, T_{i+1})$
4. **End-If**
5. **End-For**

Return Sol

B. Genetic Algorithm Phase

The whole process of the proposed GRASP_GA is detailed in Alg.4. In this procedure, GRASP_TS procedure is invoked N times to generate an initial population (Pop) of N high quality solutions according to Alg.2. This population is improved through applying Genetic algorithm on it until stopping criterion is met. In each GA iteration, tournament selection is used to select two parents X and Y for recombination. Then, crossover is applied on the two selected parents, with probability of crossover P_c, to produce two new offspring X' *and* Y'. Finally, mutation operator is applied on X' *and* Y' With probability of mutation P_m for producing X'' *and* Y''. The new offspring are saved in $NewPop$ which is evaluated and used to update the population Pop. The whole cycle is continued until the specified number of generations is achieved and the best solution Sol^* is returned as an output.

In the following, we will discuss in some details the different components of the proposed approach. These components involved chromosome representation used in our approach as well as fitness assignment, selection, crossover and mutation operators.

- Chromosomes representation:
  The proposed GRASP-GA adopts chromosome representation that depends upon TLPLC algorithm. In this representation, tasks list and processors list are combined in one array [7] as depicted in figure 3.

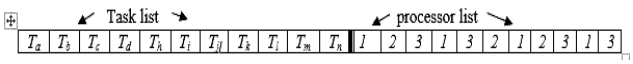

Fig. 3. Representation of chromosome

- Fitness assignment:

  As mentioned before, task schedule problems are bi-objective problem. They include the make-span (F_1) and flowtime (F_2) as defined in equation (2). In general, minimizing of make-span leads to minimize the flowtime. Consequently, the proposed GRASP-GA uses the fitness function F_1 to evaluate individuals.

- Tournament selection

  In tournament selection, a pool of individuals is constructed by choosing $TOURN\_SIZE$ individuals randomly from the population Pop. Then, the best two individuals X and Y are selected from the pool. Finally, the best two individuals in the pool X and Y are returned. the module in Alg.5. below expresses this process.

Alg.4: GRASP_GA_TS (N, $HeurFun, \alpha, P_c, P_m$)

Inputs:
N: Population Size.
HeurFun: The Heuristic function used in GRASP
α: parameter for GRASP
P_c: probability of Crossover
P_m: probability of Mutation.
Output: *Archive: all improved solutions found over generations.*
Begin:

1. $Pop \leftarrow \emptyset$. *//initialize Population*
2. **For** $i \in \{1, \dots, N\}$ **do:** *//Run GRASP N Times*
3. $Sol \leftarrow GRASP\_TS(\alpha, HeurFun, F_1)$
4. $Pop \leftarrow Pop \cup Sol$
5. **EndFor**
6. **While** Stopping criterion is not satisfied **do:**
7. $NewPop \leftarrow \emptyset$. *//initialize new Population*
8. **For** $i \in \{1, \dots, N/2\}$ **do:** *//Run GA iterations*
9. **X, Y** ← **TOUR_selection**($N/2, Pop$).
10. $r \leftarrow$ **random**(0,1);
11. **IF** ($r < Pc$) ***then***: *//apply crossover*
12. **X′, Y′** ← **Crossover**(**X, Y**).
13. **EndIF**
14. $r \leftarrow$ **random**(0,1);
15. **IF** ($r < Pm$)***then***:
16. **X″** ← **Mutate**(**X′**).
17. **Y″** ← **Mutate**(**Y′**).
18. **End IF**
19. $NewPop \leftarrow NewPop \cup \{\mathbf{X''}, \mathbf{Y''}\}$
20. **End For**
21. $NewPop \leftarrow$ **Evaluate**($NewPop$)
22. $Pop \leftarrow$ **UpdatePop**($Pop, NewPop$);
23. **End While**
24. Sol* ← **FindTheBest**(Pop)
25. **Return** Sol^*;

End

Alg.5: TOURN_Selection($TOURN\_SIZE$, Pop)**:**

Begin:

1. $pool \leftarrow \emptyset$.
2. $pool \leftarrow ChooseRandomly(\ TOURN\_SIZE\ ,\ Pop)$ //
3. $\boldsymbol{X}, \boldsymbol{Y} \leftarrow SelectTheBest(pool)$**.**
4. **Return** $\boldsymbol{X}, \boldsymbol{Y}$.

End.

- Crossover**:**
  Crossover is applied on parents' chromosomes selected by **TOURN_Selection** module with the chromosome representation explained above. With probability Pc, the process is performed on the parents' individuals by randomly selecting cross-points (x-cross). Here, we apply one cross-point and two cross-points in tasks list or one cross-point in processors list. Then, the new offspring are produced by exchanging the shaded sections between the parent chromosomes as depicted in figures 4, 5 and 6.

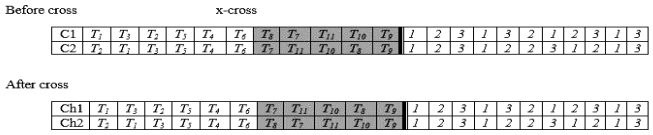

Fig. 4. One-point crossover in tasks list

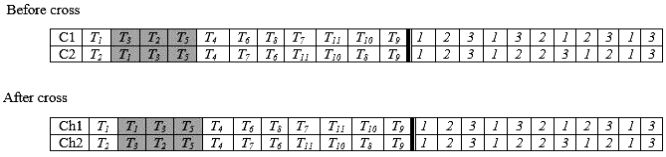

Fig. 5. Two-points crossover in tasks list

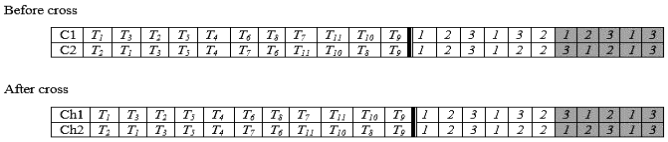

Fig. 6. One-point crossover in processors list

- Mutation**:**
  with probability P_m, Mutation operator is applied on offspring chromosomes obtained by crossover process. In this chromosome representation, swapping is very suitable to perform mutation process. The swapped genes (tasks, Processors) are randomly selected. Mutation is applied with lower probability value to avoid its negative impact on the best obtained solutions. Figures 6 and 7 show mutation process.

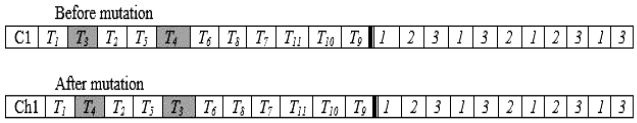

Fig. 7. mutation by swapping tasks

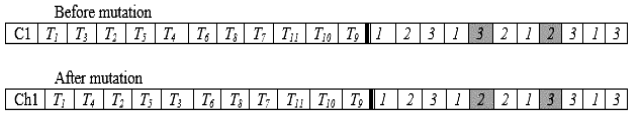

Fig. 8. mutation by swapping processors

VI. EXPERIMENTAL DESIGN

In this section, the experimentation designed to verify and test the performance of the proposed approach is introduced. The proposed GRASP-GA is verified using a set of five test problems commonly used in the literatures. These benchmarks are summarized in Table I.

In this experiment, GRASP-GA-TS is verified against two of the state-of-the-art algorithms [9]. They are Task List Processor List Combination (TL-PLC) [7] and Heuristic based Genetic Algorithm (HGA) [5]. In this experiment, two different heuristic functions are adopted *bottom-Level* and *top-Level*. Therefore, there are two GRASP schemes GRASPb and GRASPt. To study the effects of hybridization and to identify the role of each component, all schemes are involved in the experiment. As shown, there are two hybrid schemes GRASPb-GA and GRASPt-GA. The last included algorithm is the GA with random initial population. All these different schemes are depicted in figure 8.

TABLE I. THE SET OF TEST PROBLEMS

Problem	#tasks	Communication Costs	Description
P1[20]	15	25 (fixed)	Gauss-Jordan algorithm
P2[20]	14	20 (fixed)	LU decomposition
P3[7]	16	40 (fixed)	La place equation solver
P4[5]	11	Variable for each graph edge	Random
P5[21]	17	Variable for each graph edge	Random

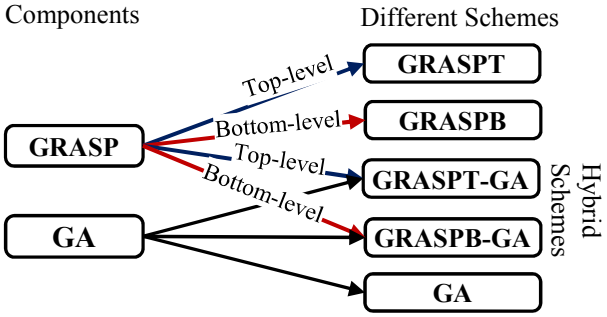

Fig. 9. Different schemes used in the study

In this experiment, each algorithm runs 15 times on each test problem using (2, 3 and 4 processors). The parameter setting used in our experiment is shown in the table II.

TABLE II. THE SET OF PARAMETER SETTINGS

GRASP parameter:α	0.4
Population size:	20
Max iteration for GA:	1000
crossover probability (pc):	0.6
Mutation probability (pm):	0.3

VII. EXPERIMENTAL RESULT

In this section, the different simulation results are shown in detail. The average make-span for each algorithm in each test problem is reported in table III. The best average make-span for each test problem in table III is highlighted in bold.

TABLE III. THE AVERAGE MAKE SPAN FOR TEST PROBLEMS

Algorithm	Test Problems				
	P1	P2	P3	P4	P5
GRASPb-GA	**252**	**250**	**760**	**14**	**36**
GRASPt-GA	254	265	796	**14**	37
GRASPb	294	284	803	19	54
GRASPt	294	299	906	19	55
GA	470	371	810	20	63
TLPLC-GA	300	270	**760**	**14**	37
HGA	440	270	**760**	18	61

According to the results in table III, it is clear that the proposed hybrid variant GRASPb-GA outperforms all the compared algorithms in most test problems. Although, HGA and TLPLC-

GA in some cases are comparable to GRASPb-GA (P3), the hybrid schemes GRASPb-GA and GRASPt-GA have the superiority. One can notice in problem 3 that GA performs better than GRASP, which gives the chance to HGA and TLPLC-GA to be comparable with GRASPb-GA. It is evident that GRASPb-GA have better performance than GRASPt-GA in P3 and P5. This is due to the better performance of *bottom level* heuristic function than *top-level* in GRASP phase. This can be noted from the results where GRASPb beats GRASPt in both P3 and P5. Generally, GRASP-GA outperforms the others in all test problems due to ability of discovering new solutions in the promising regions in the search space. GRASP-GA is managed to achieve results that outperform the stand-alone GRASP or GA, which reflects the efficiency of the hybridization.

In general, the results of GRASPb-GA in all problems are better than the results achieved by other algorithms in most test problems.The efficiency of processor is computed as in equation (8):

$$E = s \times \frac{100}{m} \tag{8}$$

where m is the number of processors and S is speed-up. Firstly, S is calculated by the formula $S = \frac{M}{M_m}$

where M is the average make-span time in one processor and M_m is the average make-span time in multi-processors. The efficiency is calculated for each test problem for each algorithm and shown in figures 10, 11, 12, 13 and 14. The efficiency assures all the results explained above. It is so clear that GRASPb-GA algorithm performs better than the other algorithms in most cases. The second better performing algorithm is GRASPt-GA.

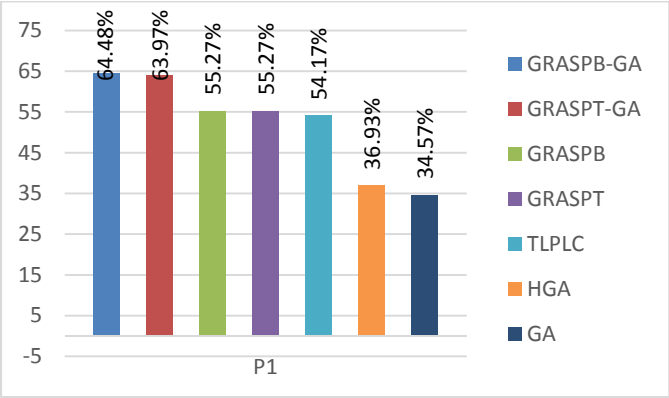

Fig. 10. Performance analysis for the compared Algorithms in test problem 1

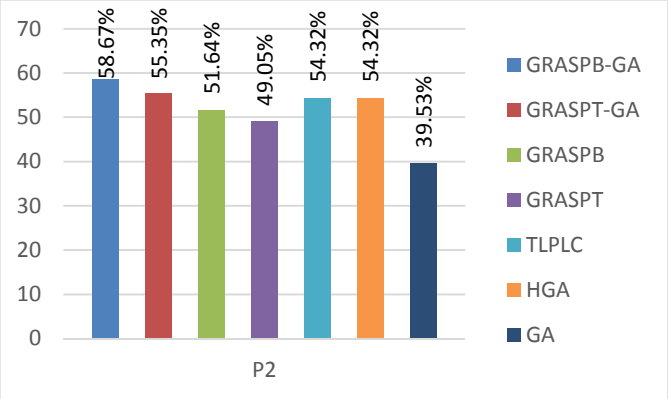

Fig. 11. : Performance analysis for the compared Algorithms in test problem 2

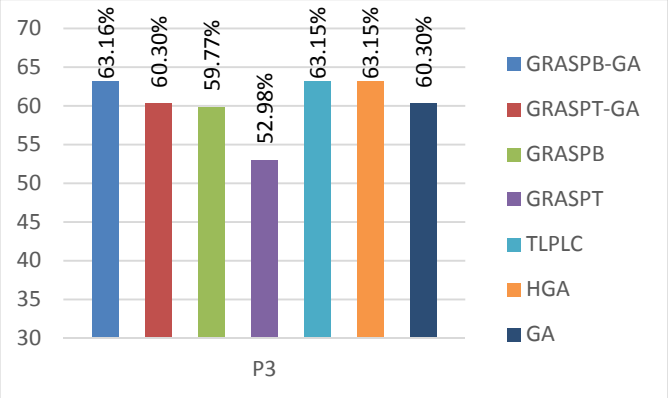

Fig. 12. Performance analysis for the compared Algorithms in test problem 3

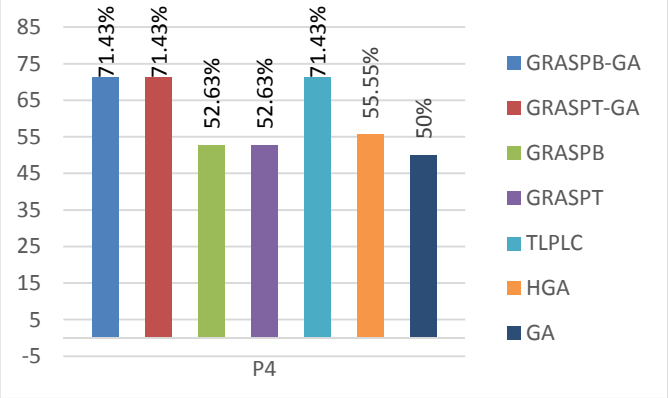

Fig. 13. Performance analysis for the compared Algorithms in test problem 4

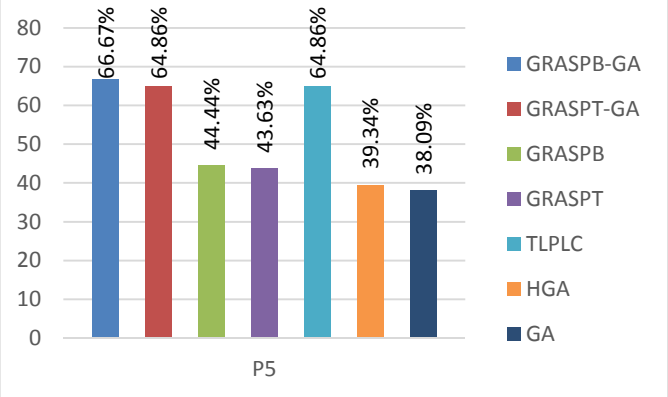

Fig. 14. Performance analysis for the compared Algorithms in test problem 5

VIII. CONCLUSION

In this paper, we proposed a new hybrid meta-heuristic for solving multiprocessor task scheduling named GRASP-GA. Two heuristic functions are adopted in GRASP, bottom-level and upper-level. Thus, two hybrid variants are proposed GRASPb-GA and GRASPt-GA. The hybrid variant GRASPb-GA have the ability to find the improved solutions for efficient assigning tasks in homogenous parallel multi-processor system. The performance of the proposed GRASP-GA is evaluated against two of the state-of-the-art algorithms from literature such as TLPLC and HGA as well as the different hybrid schemes. The proposed approach is also evaluated against its components GRASP and GA. The experiment is carried out using a set of benchmark problems from different applications. The results show that the proposed GRASPb-GA outperforms all other algorithms for all used benchmarks. This could be referred to the role played by GRASP in guiding the search to the promising regions in the search space besides the performance of *bottom-level* heuristic function in setting a trajectory to the GRASP to follow. The GA improves the best solutions obtained by GRASP in the second phase. According to these results, the proposed GRASPb-GA is highly competitive and can be considered as a viable alternative in tackling these problems. In our future work, we will improve the performance of the proposed GRASPb-GA by performing parameter tuning and using other heuristic functions for GRASP. This work can also be improved to tackle problems of task scheduling to multi-processor systems in heterogeneous environment.

REFERENCES

[1] I. Ahmad, Y. Kwok, “Benchmarking and comparison of the task graph scheduling algorithms”, J. Parallel Distributed Computing 95 (1999) 381_422.

[2] Y. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors”, ACM Computing Surveys, vol. , no. 4, (1999), pp. 406-471

[3] T. D. Braun, H. J. Siegel, and N. Beck, "A Comparison of Eleven Static Heuristic for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing, vol. 61, **(2001)**, pp. 810-837.

[4] R. Hwang, M. Gen, H. Katayama, “A comparison of multiprocessor task scheduling algorithms with communication costs”, Computers and Operations Research, Vol. 35, No. 3, pp. 976–993, 2008.

[5] K. Kaur, A. Chhabra, G. Singh, “Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System”, International Journal of Computer Science and Security, Vol. 4, No.2, 2010, pp. 149-264.

[6] J. Singh,G. Singh, “Improved Task Scheduling On Parallel Systems Using Genetic Algorithm”, International Journal of Computer Application,vol.39,(2012),pp.17-22.

[7] M. Awadall, A. Ahmad and S. Al-Busaidi, “Min–min GA Based Task Scheduling In Multiprocessor Systems”, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, vol. 3, no. 2, **(2013)** December.

[8] I. Boussaïd, J. Lepagnot, P. Siarry “A Survey on optimization metaheuristics”, Information Sciences 237 (2013) 82–117.

[9] A. Sharma, N. Singh, A. Hans, K. Kumar, “Review of task scheduling algorithms using genetic approach", International Conference on Innovative Applications of Computational Intelligence on Power, Energy, and Controls with their Impact on Humanity (HIPECH14) (2014)169_172.

[10] A. Mehrabi, S. Mehrabi, and D. Ali Mehrabi, “An Adaptive Genetic Algorithm for Multiprocessor Task Assignment Problem with Limited Memory”, International Journal in Foundations of Computer Science & Technology (IJFCST), vol. 4, no. 2, (2014) March.

[11] A. Sharma, M. Kaur, “An Efficient Task Scheduling of Multiprocessor using Genetic Algorithm based on Task Height”, International Journal of Hybrid Information Technology, vol. 8, no. 8, (2015), pp.83-90

[12] G.Padmavathi, S.R.Vijayalakshmi, “Multiprocessor scheduling for tasks with priority using GA”, International Journal of Computer Science Issues, IJCSI, Vol. 7, No. 1, 2010, pp. 37-42.

[13] S. Gulzar Ahmad, E. Ullah Munir, and W. Nisar, "PEGA: A Performance Effective Genetic Algorithm for Task Scheduling in Heterogeneous Systems", 2012 IEEE 14th International Conference on High-Performance Computing and Communications.

[14] A. Agarwal and S. Colak, “The Task Scheduling Problem: A NeuroGenetic Approach”, Journal of Business & Economics Research – Fourth Quarter, vol. 12, no. 4, 2014.

[15] M. S. Garshashbi, M. Effatparvar, “Task Scheduling On Parallel Heterogeneous Multi-Processor Systems Using Genetic Algorithm”, International Journal of Computer Applications, vol. 61, no. 9, (2013) ,pp.23-27.

[16] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109–133.

[17] M. Prais, C.C. Ribeiro, Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12 (2000) 164–176.

[18] Haupt, R.L., Haupt, S.E., Parallel genetic algorithms, John Wiley & Sons, 2004.

[19] D. E. Goldberg, “Genetic algorithms in search, optimization & machine learning”, Addison Wesley, 1990.

[20] M. R. Mohamed, M. Awadall, “Hybrid Algorithm for Multiprocessor Task Scheduling IJCSI International Journal of Computer Science Issues, ISSN: 1694 –0814, vol8, no. 2, (2011) May.

[21] Wu, A.S., Yu, H., Jin, S., Lin, K.-C., Schiavone, G., “An incremental genetic algorithm approach to multiprocessor scheduling”, IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 9, 2004, pp. 824–834.