Context-Aware Task Assignment in Ubiquitous Computing Environment - A Genetic Algorithm Based Approach

Pravin Pawar, Hailiang Mei, Ing Widya, Bert-Jan van Beijnum, Aart van Halteren

Abstract— **With the advent of ubiquitous computing, a user is surrounded by a variety of devices including tiny sensor nodes, handheld mobile devices and powerful computers as well as diverse communication networks. In this networked society, the role of a human being is evolving from the data consumer to the data producer. In these changing circumstances, pipelined processing finds applications where the data obtained from the human producer needs to be processed and interpreted in real-time. For example, in an M-Health system, the vital signs acquired from the patient are processed in the pipelined fashion.**

This paper proposes a genetic algorithm (GA) based approach for the optimal assignment of pipelined processing tasks onto a chain of networked devices that minimizes total end-to-end processing delay considering knowledge about the communication and computation resources as the context information. Although some existing graph-based algorithms can solve this problem in polynomial time, we expect that GA can take less computational time and requires less memory while providing a reasonably good assignment. We compare the performance of GA approach with the graph-based approaches. It is observed that when the number of devices and the number of processing tasks are large, the GA approach performs better in terms of the satisfactory quality of the obtained sub-optimal solution considering the advantage of reduced computational time.

I. Introduction

In a sequential data processing system, a *pipelined processing* job consists of a chain of processing tasks where every processing task except the first one; takes as input the output of the previous task. A number of applications which involve pipelined processing exist by applying a series of signal processing tasks e.g., to capture, storage, manipulate and transmit multi-media objects in the media processing system [1-3], and the flow-shop production plant among others. A characteristic of the pipelined processing system is that every task involved in the processing chain needs certain amount of time and additionally requires the resources which are capable of executing this task. Therefore, the research in this area has focused on performing the optimal assignment of the processing tasks onto the available resources such that the *data throughput* is maximized [1] or the *end-to-end processing time* is minimized [4, 5].

The more general problem of *multiprocessor scheduling* which involves obtaining a schedule for a general task graph to be executed on a multiprocessor system to minimize the schedule length is *NP-hard* [6]. Several subclasses of the problem are known to be in P, and several algorithms that solve these in polynomial time have been reported [1, 7, 8].

A. Task Assignment in the M-Health Domain

In today's world of ubiquitous computing, in terms of the data processing resources, a human user is surrounded by a variety of devices ranging from the tiny sensors, handheld mobile gadgets and powerful computers; each consisting of its own processing power, software and hardware platforms. Similarly, the communication technology includes short range, low bandwidth networks (e.g. Bluetooth), long range, low bandwidth networks (e.g. GPRS) as well as short range, high bandwidth networks (e.g. WLAN and UWB) coexisting with the conventional wire-line networks. Owing to these advances and the need to provide user-centric applications, the role of a human being is also evolving from the *data consumer* (e.g. downloading ring-tones and news) to the *data producer* (e.g. user context such as agenda, current activity). A representative example of these changing trends is the *remote patient tele-monitoring system* in the *Mobile healthcare (M-health)* domain. The applications in the M-Health domain are receiving more and more attentions due to its ability to reshape the healthcare delivery process to satisfy today's society demands, like patient self-management, continuous and anywhere tele-monitoring and tele-treatment [9-11].

The purpose of the remote patient tele-monitoring system is to gather the patient's bio-signals (e.g. ECG, oxygen saturation level, heart rate) collected from medical sensors attached to the patient's body, process and transfer this data in a near real-time fashion to the server in healthcare centers located in the fixed network via several computing devices in the signal delivery path. A *bio-signal processing job* often consists of cascaded processing tasks, which have to be configured onto the devices in the signal delivery path. This job falls in the category of the pipelined jobs mentioned earlier in this section.

Manuscript received March 15, 2007. This work is part of the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. (http://awareness.freeband.nl).

Pravin Pawar, Hailiang Mei, Ing Widya and Bert-Jan van Beijnum are with the Architectures and Services of the Networked Applications Group, Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500AE, Enschede, The Netherlands.

Ph: 0031 – 53 – 489 3283 Fax: 0031 – 53 – 489 5477

Email: {P.Pawar, H. Mei, I.Widya, B.J.vanBeijnum}@utwente.nl

Aart van Halteren is with Philips Research, Eindhoven, The Netherlands.

Email: Aart.van.Halteren@philips.com

Medical protocols and practices prescribe strict requirements on the processing quality of the bio-signal data, which necessitates the need of allocating the intermediate processing tasks on the devices providing suitable computation resources. Furthermore, during the emergency conditions, the caregiver or paramedics need to be informed about the critical condition of a patient within a specified time to react, which necessitates the need of minimizing the bio-signal processing time. Hence, a challenge in the remote patient tele-monitoring system is to optimally assign the bio-signal processing tasks to available devices such that the end-to-end processing delay is *minimized* and subject to the *constraints* on the communication and computation resources. We refer to the information providing the knowledge about the communication and computation resources as the *context information*. A context-aware system adapts according to the context information e.g. user context, service context and communication and computation environment context, as well as to changes to the context information over time [5].

B. Graph-based algorithm

The problem of the assignment of the bio-signal processing tasks to the available devices is found to be similar to an industrial case studied earlier by Bokhari in the area of *parallel and pipelined computing* [1]. To solve this problem, Bokhari presented a graph-based solution by constructing a layered assignment graph. In this assignment graph, any path connecting two distinguished nodes represents a possible assignment. Therefore, the assignment problem is translated into a path-search problem which can be solved in polynomial time. Some improved algorithms are further reported in [7, 12, 13]. Mei ([5]) studied a similar problem by relaxing the *contiguity constraint* [1] and proposed a new graph-based method to compute the optimal assignment. It has been found that this solution can also be extended to solve the problem with the contiguity constraint and it moreover reduces both the time complexity and space complexity compared to Bokhari's original algorithm.

C. Genetic Algorithm

Since the task assignment problem in the general multiprocessor scheduling is NP-hard, heuristic methods and in particular genetic algorithm are developed [14, 15]. However the solutions proposed in [14, 15] can not be applied to our task assignment problem in their original form because we consider a chain of processing devices, communication and computation constraints in addition to the ordering constraint (to be explained in the Section III). However, we consider some special techniques proposed in these approaches (e.g. knowledge augmented genetic algorithm in [15]). As compared to [14, 15], our approach benefits from the simpler chromosome structure and less processing intensive crossover and mutation operations consequently resulting in a lower time span to achieve the sub-optimal solution.

We perform a series of experiments using genetic algorithm to compare its performance with Bokhari and Mei's algorithms for the similar problems. Although both of them solve the pipelined task assignment problem in polynomial time, we expect that GA takes less computational time and requires less memory while providing a reasonably good solution. In this paper, we evaluate the performance of GA to verify our assumption with the following two aspects: 1) the *computation time* and 2) *quality of the obtained sub-optimal solution* as compared to the optimal solution. We also study the effect of the initial population generation strategy and the population size on the quality of the obtained sub-optimal solution.

D. Paper Outline

This paper is organized as follows: Section 2 describes a context-aware M-health application scenario for predicting epileptic seizures from sensed bio-signals. Section 3 formulates the minimization problem. Section 4 briefly describes Mei's algorithm. Section 5 presents the genetic algorithm framework. Section 6 discusses the results of applying the GA algorithm and compares these with Bokhari's and Mei's algorithm. Section 7 concludes this paper and discusses the future work.

II. Context Aware M-Health Application Scenario

An epilepsy prediction job developed in [16] consists of six bio-signal processing tasks (Figure 1). We name these tasks as *Bio-Signal Processing Units* (BSPUs). First, the patient's sensed Electro Cardio Gram (ECG) data is filtered to remove signal artifacts and environment noise. Thereafter, beat-to-beat heart rate (HR) is derived and heart rate variability (HRV) in the frequency domain is calculated. The frequency spectrum of the HRV is then used to calculate the probability of an upcoming or occurring seizure. To reduce the chance of false alarms, the patient's activity information is monitored as well and correlated with the analyzed spectrum in the final stage.

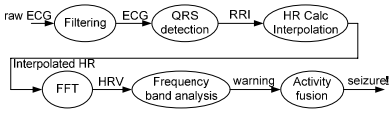

Figure 1. Epilepsy prediction task consisting of six BSPUs

An M-health platform [17], which facilitates the sensing, processing and transporting of the bio-signals, consists of the following devices: a set of body-worn sensor boxes, a handheld Mobile Base Unit (MBU), a back-end server and an end-terminal (Figure 2). The sensor boxes collect the bio-signals of the patient, sense other data like location or activity, and send the data to the MBU over wireless short-range connections, e.g. Bluetooth. As a gateway, the MBU sends the data to the back-end server over a WiFi, GPRS or UMTS connection. Thereafter, the data can be streamed to or accessed by a healthcare professional using his end-system, e.g. his laptop or desktop PC. Dynamically reassigning the processing tasks to the available devices in

run time has a potential to enhance the system adaptivity to the environmental context changes. To achieve this desired adaptation, the first topic to be addressed is how to find the best device for each BSPU. We formulate this assignment problem in the next section.

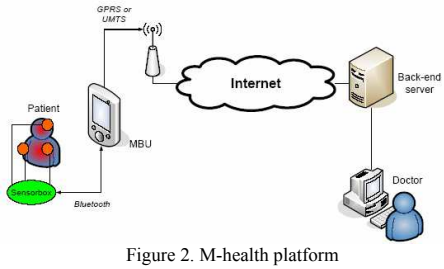

Figure 2. M-health platform

III. Bio Signal Processing Task Assignment Problem

We formulate the problem of optimal assignment of a BSPU chain onto a device chain as:

- Given: a directed BSPU chain containing m BSPUs increasingly indexed by $i \in \{1,2,\ldots m\}$, a directed device chain containing n devices increasingly indexed by $j \in \{1,2,\ldots n\}$ and an objective function τ, where τ denotes the end-to-end processing delay for a particular configuration of BSPUs across devices.
- Goal: To find an assignment function Ψ_{opt} among all possible Ψ: $\{i\} \rightarrow \{j\}$ such that $\tau(\Psi_{opt})$ is the minimal one among all $\tau(\Psi)$.
- Subject to: the following *ordering* and *local constraints*:

The *ordering constraint* is that for every pair of BSPU i and BSPU i', if $i < i'$, then $\Psi(i) \leq \Psi(i')$. This constraint therefore reflects the alignment of directions of the two chains. It is a more relaxed constraint compared to the *contiguity constraint* discussed in [7, 12, 13]. The *local constraint* is that for every BSPU i, its hosting device $\Psi(i)$ should satisfy the BSPU i's system requirement (e.g. CPU speed and library support) and user preference (e.g. this BSPU i has to be assigned on device $\Psi(i)$).

We define the computation time for BSPU i to process one frame of bio-signal data at device j as $p_{i,j}$ and the communication time for transferring one frame of BSPU i's output bio-signal data over the link between device j and $j+1$ as $c_{i,j}$. The variable $c_{0,j}$ denotes the communication time for transferring one frame of sensed data (the input to BSPU *1*) over the link between device j and $j+1$. We neglect intra-device communication time, e.g. if both BSPU i and $i+1$ are located on the same device. We further assume there is no additional computation time overhead for executing several BSPUs on the same device. Since the values of $p_{i,j}$ and $c_{i,j}$ could be known a prior, e.g. based on analytical benchmarking or task profiling [18], the end-to-end delay τ of a particular configuration of BSPUs across devices can be calculated. In Figure 3, we illustrate a *6-BSPU-4-device* assignment of the epilepsy prediction task on the M-health platform. In this example, BSPU 1 and 2 are assigned to device 1, BSPU 3, 4 and 5 are assigned to device 3 and BSPU 6 is assigned to device 4. Based on our definitions, its end-to-end processing delay can be calculated as:

$$\tau = p_{1,1} + p_{2,1} + p_{3,3} + p_{4,3} + p_{5,3} + p_{6,4} + c_{2,1} + c_{2,2} + c_{5,3}$$

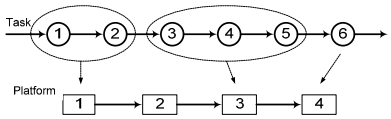

Figure 3. An example of assigning a directed BSPU chain (m=6) onto a directed device-chain (n=4)

For an m-BSPU-n-device model, the number of possible assignments without considering the local constraints is:

$$\binom{m+n-1}{n-1}$$

For a larger number of m and n, this problem is unsolvable if an exhaustive search is conducted: E.g. for $m=60$ and $n=30$, there are in total 2.24×10^{23} possible assignments to be checked.

IV. Graph-based algorithm for the Optimal Assignment of Pipelined Processing tasks

A graph-based method to compute the optimal assignment of a BSPU chain onto a device chain is proposed in [5]: it first constructs a layered assignment graph that contains all the possible sub-assignment of BSPU to device. After a source node and a sink node are added, all nodes in the assignment graph are connected according to the ordering constraint (Figure 4). When the construction of the graph is complete, every path connecting the source node and the sink node corresponds to one possible assignment. Therefore, the problem of finding the optimal assignment is translated into a path-search problem that can be solved with any shortest path algorithm. The space complexity is $O(m*n)$ and the time complexity for the search is $O(m^* n)$ if a specific labeling procedure is applied.

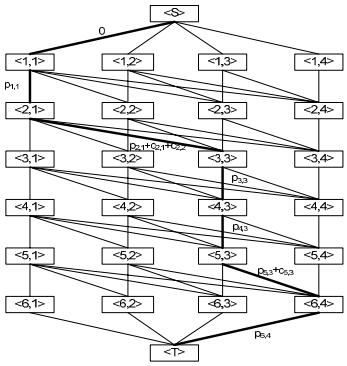

Figure 4. The 6-BSPU-4-device assignment graph without the contiguity constraint

The experiment study proves the efficiency of this graph-based method. Since this technique also finds its applications in the dynamic run-time (re)assignment of BSPUs to the processing devices, the processing time required to calculate the (re)assignment is also critical. A sub-optimal solution calculated using the heuristic algorithms such as GA may save on the processing time at

the expense of a little difference between the optimal and sub-optimal solutions. In this paper we want to evaluate the performance of GA from the following two aspects: 1) the computation time, and 2) quality of the obtained sub-optimal solution as compared to the optimal solution.

V. Genetic Algorithm for the Optimal Assignment of Pipelined Processing tasks

Genetic Algorithm (GA) is a proven heuristic technique based on the Darwin's theory of the '*survival of the fittest*' to find the sub-optimal solution in the large search spaces [19, 20]. Herewith, we provide a brief overview of the steps applied in our GA:

The first step is an encoding of the possible solution which is represented by a *chromosome*. There exist several encoding techniques such as binary encoding, number encoding, character encoding and tree encoding [21]. Each chromosome represents one solution to the problem. A set of chromosomes is referred to as a *population*. For our problem, we choose a *number encoding* of the chromosome.

The second step is the generation of an *initial population*. Usually, the chromosomes which constitute an initial population are generated purely randomly. However, to take care of the *ordering constraint*, we introduce certain ordering in the generation of the initial population. For the second step, the initial population represents the *current generation*. The number of chromosomes constituting the *current generation* is known as the *population* size. The population size affects the quality of the sub-optimal solution obtained.

The third step is to evaluate the *fitness* of every chromosome in the *current generation*. A specific *objective function* representing the *utility* of the chromosome determines the fitness of the chromosome. In our case, the objective function evaluates the *end-to-end delay* in the BSPU processing chain as assigned by the chromosome to the devices. Since our problem is the minimization problem, a *smaller delay* represents *higher fitness*.

The fourth step is applying the *selection operator* to form the *next generation*. The selection operator allows the GA to take biased decisions favoring the chromosomes having higher fitness value over the chromosomes having the lower fitness value. There exist several selection techniques such as *roulette wheel selection*. We choose the *roulette wheel selection* technique for our experimentation. In this technique, the probability of the selection of a chromosome in the next is directly proportional to its fitness value. The population size of the next generation typically remains the same as the initial generation. After selecting the next generation, the current generation is replaced by the initial generation.

Selection is followed by applying the *crossover operator* over the current generation. With some probability (referred to as *crossover probability*), some pairs of chromosomes (referred to as *parent chromosomes*) are selected from the current population and some of their corresponding components are exchanged at the randomly chosen *crossover point*(s) to form two new chromosomes (referred to as *children chromosomes*). There are also various types of the crossover involving *single point crossover* and *multi-point crossover*. We choose the single point crossover for our purpose. The crossover operator eventually leads to combining the best numeric sequence of one chromosome with that of the other chromosome to evolve a better new chromosome(s).

After applying the crossover operator, each element of the chromosome in the population is applied a *mutation operator* with some probability (referred to as *mutation probability*). The mutation operator transforms a chromosome into another chromosome which could eventually lead to a better chromosome which could not be obtained just by applying the crossover operator.

Later, the sequence of steps (referred to as *one iteration*) involving selection, crossover and mutation operators is repeated until a certain *stopping criterion* is met. A stopping criterion may involve various forms such as running the GA for a certain number of iterations, or when no improvement in the maximum fitness could be obtained after a predefined number of iterations. We adopt the later criterion in our experiments.

A. Encoding of a solution

We encode a chromosome as a sequence of numbers. The length of the chromosome is same as the number of BSPUs. The first number of the chromosome represents the index of device (i.e. $\Psi(1)$) on which the first BSPU will execute; the second number represents the index of device ((i.e. $\Psi(2)$)) on which the second BSPU will execute and so on. Thus, every chromosome represents a different assignment. An example of the chromosome corresponding to the assignment illustrated in Figure 3 is given in Figure 5.

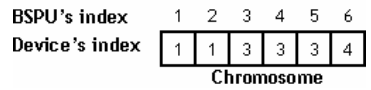

Figure 5: A possible chromosome representing the placement of BSPUs to the devices where $m = 6$ and $n = 4$.

B. Initial population generation

Since it is required that the assignment of BSPUs to devices should follow the *ordering constraint*, according to the encoding technique chosen in the Section V.A, the search space consists of all the BSPU to device mappings following the ordering constraint and closed by the chromosome assigning each BSPU to the 1st device and the chromosome assigning each BSPU to the nth device as shown in the Figure 6.

When creating the chromosomes in the initial population, initially we had chosen to use a random assignment of BSPUs to devices. However, during the experimentation, we found that unless the number of BSPUs is very small, the members of the initial population hardly follow our *ordering*

constraint. Hence we introduced certain ordering in the chromosomes using three approaches described as the following to possibly include the members covering a large area of the search space.

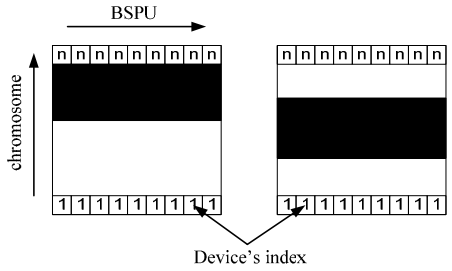

Figure 6: The characteristics of the initial population generated using the approach 1 and 2 respectively.

1. The first approach consists of assigning a random device with number $j < n$ to the 1st BSPU. The subsequent BSPUs are assigned the device whose index is equal to or greater than the device's index assigned to the previous BSPU. However, this approach leads to the generation of the chromosomes where most of the BSPUs are assigned to device n. This characteristic is shown in the first part of the figure 6.
2. The second approach chooses a BSPU $b < m$ randomly and assigns a random device $r < n$ to it. The BSPUs indexed by $i < b$ ($i > b$) are assigned to the device whose index is equal to or less (greater) than the index of the device assigned to the next (previous) BSPU. This approach generated initial population with more variety than the 1st approach. However, it leads to the generation of the chromosomes where most of the BSPUs are assigned device indexed by *1*, *n* or a combination of them. This characteristic is shown in the second part of the figure 6.
3. The third approach of the initial population generation is more controlled than the second approach. The initial step of choosing a random BSPU and assigning a random device to it remains the same. However, the BSPUs indexed by $i < b$ ($i > b$) are assigned to the device whose index is equal to or *JUST ONE* less (greater) than the index of the device assigned to the next (previous) BSPU. This approach generates the members of the initial population spreading most of the search space.

C. Objective Function and Assigning Fitness

Our GA takes as input two sets of values. The first set ($p_{i,j}$) containing $m * n$ values represents the processing time required for every BSPU to execute on every device. The second set ($c_{i,j}$) contains $(m+1) * (n-1)$ elements and represent the communication delay involved in the transfer of data between BSPUs. If there are n devices along the processing chain, then there are $n-1$ communication links between them. The number $m+1$ denotes that the first BSPU receives initial data for processing which propagates along the processing chain. Calculating the end-to-end delay in the BSPU assignment encoded by the chromosome is a two step process:

1. The first step is to calculate the *computation delay* D_p.
2. The second step involves calculating the *communication delay* D_c.

Refer to Section III for more information on the communication and computation delay.

The total end-to-end processing delay τ is a sum of D_p and D_c. The fitness F_i of a chromosome i is calculated as follows:

$$F_i = Max(\tau_1 \ldots \tau_{pop.\ size}) - \tau_i + 1$$

This ensures that the fitness of a chromosome is inversely proportional to its end-to-end delay and more than 0.

D. Crossover Operator

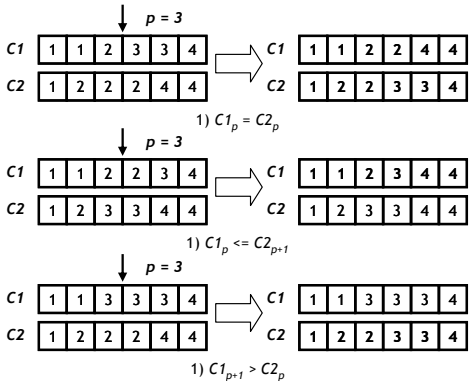

Figure 7: Effect of crossover operator

Let *C1* and *C2* be the two chromosomes selected for the crossover. To ensure that the crossover operator results in the chromosomes satisfying the ordering constraints, we consider the following three conditions in the given order at the randomly chosen crossover point p:

1. $C1_p=C2_p$: This case results in the normal crossover where *C1* and *C2's* parts after the crossover point are exchanged.
2. $C1_p<=C2_{p+1}$: In this case, *C1's* part after the crossover point is replaced with the *C2*'s part after the crossover point. There is no change in *C2*.
3. $C1_{p+1}>C2_p$: In this case, *C2's* part after the crossover point is replaced with the *C1*'s part after the crossover point. There is no change in *C1*.

Figure 7 shows the results of the crossover for the three conditions explained above. The probability of crossover in our experiments is *0.8*.

E. Mutation Operator

In a chromosome C, let m be the index of the element chosen for the mutation. To ensure the ordering constraint, the mutation operator changes C_m to a value which is in the range of $[C_{m-1} \ldots C_{m+1}]$ inclusive of both the values.

Figure 8 shows the result of the mutation. The value 1 could have mutated in the range [1…4]. The probability of mutation in our experiments is *0.01*.

Our approaches for the initial population generation,

Table 1: Some of the results comparing the performance of GA with the deterministic approaches (time and delay are in milliseconds)

Settings			Time			Sum Delay			
#BSPU	#device	Pop. Size	Bokhari	Mei	GA - 3	Mei	Avg. GA3	Best GA3	Std. Dev.
10	20	100	141	47	13	1167	1346	1241	77.18
10	30	150	422	32	36	1743	1885	1773	45.26
10	40	200	1359	31	39	2195	2572	2425	83.37
20	30	300	281	110	94	2029	2161	2090	42.93
20	40	400	1297	156	230	2457	2750	2627	87.62
20	50	500	5796	157	303.1	3157	3349.1	3255	78.01
30	40	600	437	391	335.9	2845	3244.9	3100	91.31
30	50	750	2484	500	528.1	3357	3720.1	3650	37.01
30	60	900	NA	875	665.6	3732	4379.7	4105	146.21
40	60	1200	NA	1344	1310.9	3825	4351.6	4072	249.99
40	70	1400	NA	2015	1779.8	4418	4939.4	4800	127.85
50	70	1750	NA	3109	1079.7	4792	6069.6	5555	235.69
50	80	2000	NA	4125	3689.1	5257	6081.9	5719	143.03
60	80	2400	NA	7125	5201.4	5463	6573.5	6454	85.78
60	90	2700	NA	7937	5782.8	6147	7148.3	6923	232.17
70	90	3150	NA	10985	8988.8	6111	6871.7	6643	117.42
70	100	3500	NA	13296	13034.1	6400	7271.4	6991	159.18
80	100	4000	NA	23437	13245.2	7261	8331.7	8098	148.82

crossover and mutation operators are inspired by the *knowledge-augmented genetic algorithm* presented in [15] where certain knowledge of the problem is taken into account to generate the initial population and perform crossover and mutation operations.

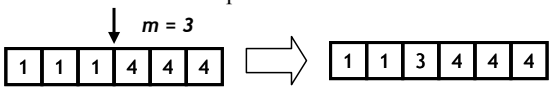

Figure 8: Effect of mutation operator

VI. Performance Results of the GA

The experimentation strategy consists of executing the GA algorithm along with Bokhari and Mei's algorithms for a combination of various number of BSPUs and devices. Table 1 shows the number of BSPUs and devices used. We execute three variants of GA algorithm based on the three initial population generation strategies described in the Section V.B. We refer to these algorithms as GA-1, GA-2 and GA-3. For each GA algorithm, we experiment on two population sizes of *0.5*(m*n)* and *(m*n)* respectively. *(m*n)* represents the number of nodes in the assignment graph in Mei's algorithm. The results reported for GA algorithm are averaged over 10 runs each.

Although Bokhari's algorithm uses the *contiguity constraints* and Mei's algorithm and GA use the ordering constraints, we present the results for all the three algorithms because they basically address the task assignment problems in the pipelined processing.

Table 1 shows the results obtained for the Bokhari, Mei and GA-3 algorithm for the population size *0.5*(m*n)*. For the GA-3 algorithm, we provide the standard deviation from the 10 runs. Other results are excluded from the table for brevity. Please note that Bokhari's algorithm could not be executed for all the combinations of BSPUs and devices because of the exceptional memory requirements in representing Bokhari's graph.

A. Time Comparison

Figure 9 (Figure 10) compare the required execution time of GA-1, GA-2 and GA-3 algorithms for the population size *0.5*(m*n)* and *(m*n)* against Bokhari and Mei's algorithms. As it can be observed from the Figure 9, for the population size *0.5*(m*n)*, as the number *m*n* increases, the GA algorithms require less time than Bokhari and Mei's algorithms. However, as Figure 10 shows, for the population size *(m*n)*, GA require higher execution time compared to Mei's algorithm.

B. GA Solution Quality

As Mei's algorithm provides an optimal solution to the assignment problem and GA provides sub-optimal solution, Figure 11 (figure 12) compares the average % difference in the end-to-end processing delay obtained using GA-1, GA-2 and GA-3 algorithms for the population size *0.5*(m*n)* (*(m*n))* to the end-to-end delay obtained using Mei's algorithm. These graphs show that GA-3 algorithm outperforms GA-1 and GA-2 algorithms by a substantial margin. The average percentage difference in the sub-optimal solution obtained using GA-3 algorithm is around 10% to the Mei's algorithm.

C. GA Population Size Comparison

Figure 13 shows the percentage difference between the best sub-optimal solution obtained using GA-3 algorithm for the population size *0.5*(m*n)* and *(m*n)*. As can be seen from the graph, it does not show any definite trend. Figure 14 shows the average % difference between the end-to-end delay obtained using GA-1, GA-2 and GA-3 algorithms for the population size of *0.5*(m*n)* and *(m*n)*. It can be observed that there is no specific behavior in the solution quality with respect to the population size.

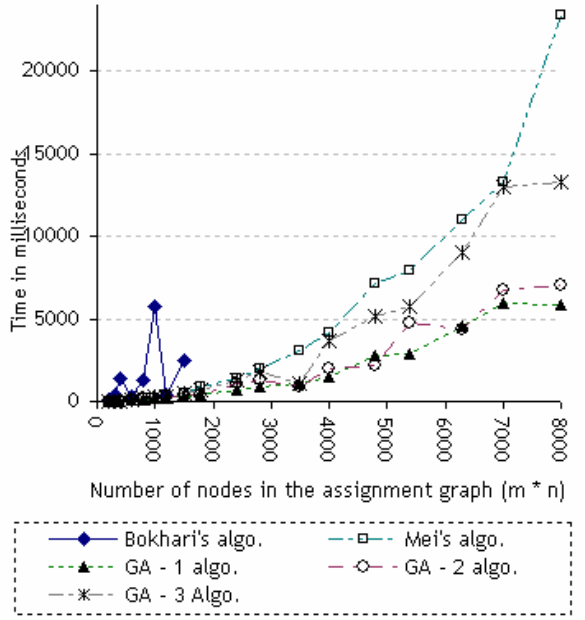

Figure 9: Time required to obtain the (sub) optimal solutions. For the GA algorithms, the population size is 50% of (*m* * *n*)

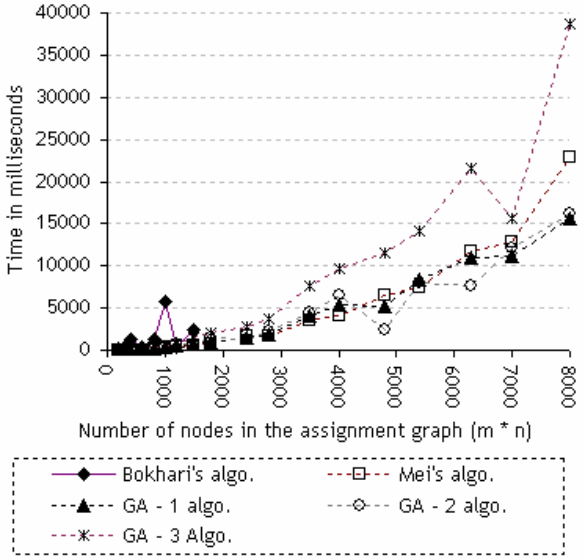

Figure 10: Time required to obtain the (sub) optimal solutions. For the GA algorithms, the population size is (*m* * *n*)

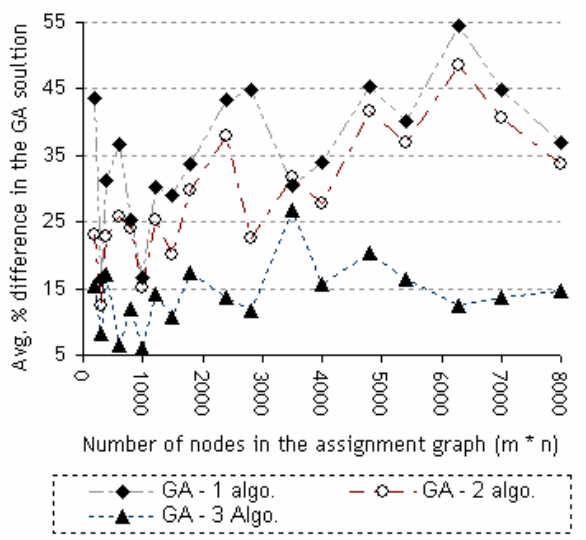

Figure 11: The average difference in percentage for the sub-optimal solution obtained using GA as compared to Mei's algorithm. For the GA algorithm, the population size is 50% of (*m* * *n*)

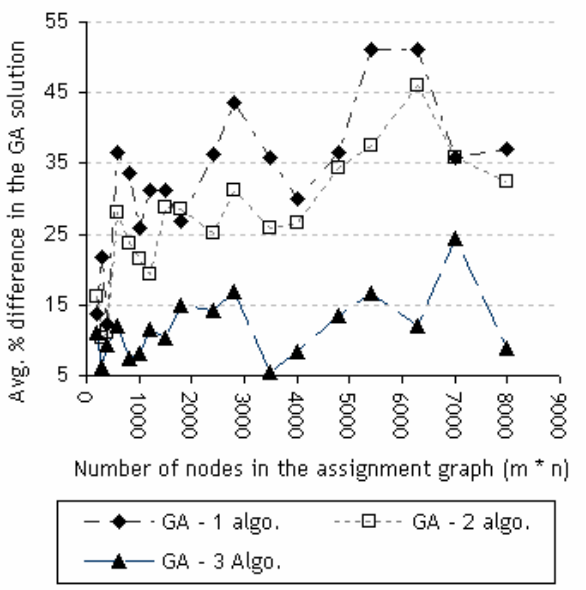

Figure 12: The average difference in percentage for the sub-optimal solution obtained using GA as compared to Mei's algorithm. For the GA algorithms, the population size is (*m* * *n*)

VII. Conclusion and Future Work

This paper presents a genetic algorithm based solution for the optimal assignment of the pipelined processing tasks onto a chain of networked devices to minimize the total processing time subject to the context information such as the processing power of these devices and the capacity of the communication the links connecting these devices. Comparing the performance of genetic algorithm based heuristic approach with the other two approaches which provide optimal solution to the assignment problem, it is

observed that when the number of devices and processing tasks is large, genetic algorithm performs better in terms of the required time. The quality of the sub-optimal solution obtained using GA is satisfactory considering the advantage of saving time in case of the real-time calculations. We used a variety of initial population generation approaches and observed that GA provides a better solution if the initial population has the samples uniformly distributed in the search space. For this problem, we did not find a certain relationship between the population size and the quality of the sub-optimal solution obtained using GA. However, it is to be noted that arbitrarily small population size may affect the quality of the solution.

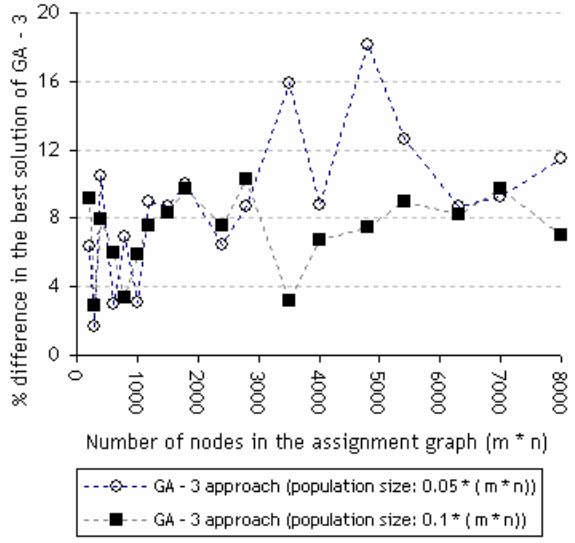

Figure 13: The % difference for the sub-optimal solution obtained using GA – 3 algorithm as compared to Mei's algorithm.

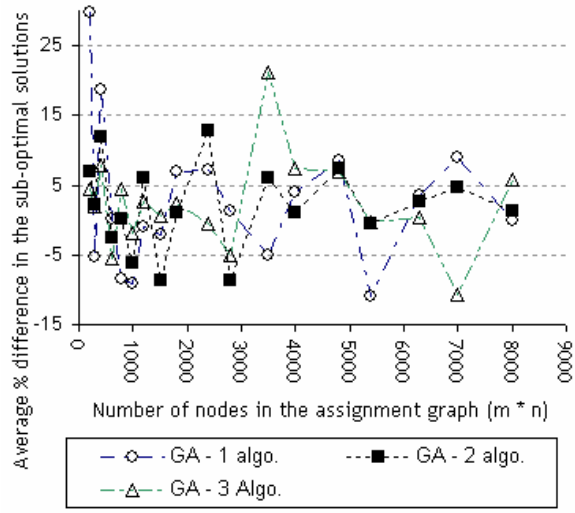

Figure 14: Average % difference for t solutions obtained for the population size *0.5 * (m * n)* and *(m * n)* using GA

Part of future work will focus on the integration of the assignment algorithm (both graph-based and GA) with the M-health system. Therefore, we are planning to develop a task reallocation infrastructure. This infrastructure will take the computed (sub-)optimal assignment as the input and reconfigure the M-health system accordingly. In particular, we are interested in the tradeoff between the performance gain and the cost of reconfiguration and try to improve the assignment algorithm to take this trade-off into account.

REFERENCES

1. Bokhari, S.H., *Partitioning problems in parallel, pipelined, and distributed computing.* IEEE Transactions on Computers, 1988. **37**(1): p. 48-57.
2. Gu, X. and K. Nahrstedt, *Distributed multimedia service composition with statistical QoS assurances*. Multimedia, IEEE Transactions on, 2006. **8**(1): p. 141.
3. Dasu, A. and S. Panchanathan, *A survey of media processing approaches.* Circuits and Systems for Video Technology, IEEE Transactions on, 2002. **12**(8): p. 633-645.
4. Stone, H.S., *Multiprocessor scheduling with the aid of network flow algorithms.* IEEE Transactions on Software Engineering, 1977. **3**: p. 85-93.
5. Mei, H. and I. Widya. *Context-Aware Optimal Assignment of a Chain-like Processing Task onto Chain-like Resources in M-Health*. in *7th International conference on computational science*. 2007. Beijing, China.
6. Fernandez-Baca, D., *Allocating modules to processors in a distributed system.* Software Engineering, IEEE Transactions on, 1989. **15**(11): p. 1427-1436.
7. Nicol, D.M. and D.R. O'Hallaron, *Improved Algorithms for Mapping Pipelined and Parallel Computations.* IEEE Transactions on Computers, 1991. **40**(3): p. 295-306.
8. Ashraf Iqbal, M. and S.H. Bokhari, *Efficient algorithms for a class of partitioning problems.* Parallel and Distributed Systems, IEEE Transactions on, 1995. **6**(2): p. 170.
9. Halteren, A.v., et al., *Mobile Patient Monitoring: The MobiHealth System.* The Journal of Information Technology in Healthcare, 2004. **2**(5).
10. Hung, K. and Z. Yuan-Ting, *Implementation of a WAP-based telemedicine system for patient monitoring.* Information Technology in Biomedicine, IEEE Transactions on, 2003. **7**(2): p. 101.
11. Yuan-Hsiang, L., et al., *A wireless PDA-based physiological monitoring system for patient transport.* Information Technology in Biomedicine, IEEE Transactions on, 2004. **8**(4): p. 439.
12. Sheu, J.-P. and Z.-F. Chiang, *Efficient allocation of chain-like task on chain-like network computers.* Information Processing Letters, 1990. **36**(5): p. 241 - 246.
13. Woeginger, G.J. *Assigning chain-like tasks to a chain-like network*. in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. 2001.
14. Wang, L., et al., *Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach.* Journal of Parallel and Distributed Computing, 1997. **47**: p. 8-22.
15. Correa, R.C., A. Ferreira, and P. Rebreyend, *Scheduling multiprocessor tasks with genetic algorithms.* Parallel and Distributed Systems, IEEE Transactions on, 1999. **10**(8): p. 825-837.
16. Tönis, T., H.J. Hermens, and M. Vollenbroek-Hutten, *Context aware algorithm for discriminating stress and physical activity versus epilepsy*. 2006, AWARENESS deliverables (D4.18).
17. AWARENESS, *Freeband AWARENESS project webpage*. 2005, http://www.freeband.nl/project.cfm?id=494&language=en.
18. Braun, T.D., et al. *A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems*. in *17th IEEE Symposium on Reliable Distributed Systems*. 1998.
19. Goldberg, D.E., *Genetic Algorithms in search, optimization and machine learning*. 1989: Addison Wesley publication.
20. Mitchell, M., *Introduction to genetic algorithms*. 1996: MIT press.
21. Pawar, P. and S.L. Mehndiratta. *Database Hierarchy Optimization in PCS networks: A Genetic Algorithm Approach*. in *International Conference on Knowledge Based Computer Systems (KBCS 2004)*. 2004. Hyderabad, India.