International Journal of Management Science
Vol 10, No 2, November 2004

A Processor Assignment Problem for ATM Switch Configuration

Junghee Han*

College of Business Administration, Kangwon National University,
Hyoja-2Dong, Chunchon-Shi, Kangwon-Do, 200-701, Korea

Youngho Lee**

Department of Industrial Systems and Information Engineering, Korea University,
Sungbuk-Gu Anam-Dong, Seoul, 136-701, Korea

(Received May 2004 ; Revised Oct. 2004 ; Accepted Oct. 2004)

ABSTRACT

In this paper, we deal with a processor assignment problem that minimizes the total traffic load of an ATM switch controller by optimally assigning processors to ATM interface units. We develop an integer programming (IP) model for the problem, and devise an effective tabu search heuristic. Computational results reveal the efficacy of the proposed tabu search procedure, finding a good quality solution within 5% of optimality gap.

Keywords: Graph partitioning, Tabu search, ATM switch

1. INTRODUCTION

The processor assignment problem (PAP) seeks to minimize the total traffic transaction load of an ATM switch controller by optimally assigning processors to interface units. Figure 1 illustrates the configuration of ATM switch, controller, interface units and processors. For example, let us suppose that there are eight processors with STM-1 (155Mbps) interface and an ATM switch supporting two STM-4 (620Mbps) interface units. Each interface unit multiplexes four STM-1s to a STM-4, or demultiplexes a STM-4 into four STM-1s. The switch controller, directly connected to the ATM switch, sends control signal to interface units via

* Corresponding Author, Email: jhhan@kangwon.ac.kr
** Email: yhlee@korea.ac.kr

signaling path such as HDLC (High Data Link Control) bus. When requested to set up SVC (Switched Virtual Circuit) between two processors, the switch controller sends a signal to an interface unit. However, if two processors are assigned to different interface units such as the case of processors 1 and 5 in Figure 1, the switch controller sends two separate signals to the interface units via HDLC bus. This increases the load of switch controller incurring the congestion in the signaling path. Accordingly, the processor assignment plays a key role in managing the traffic load of ATM switch controller.

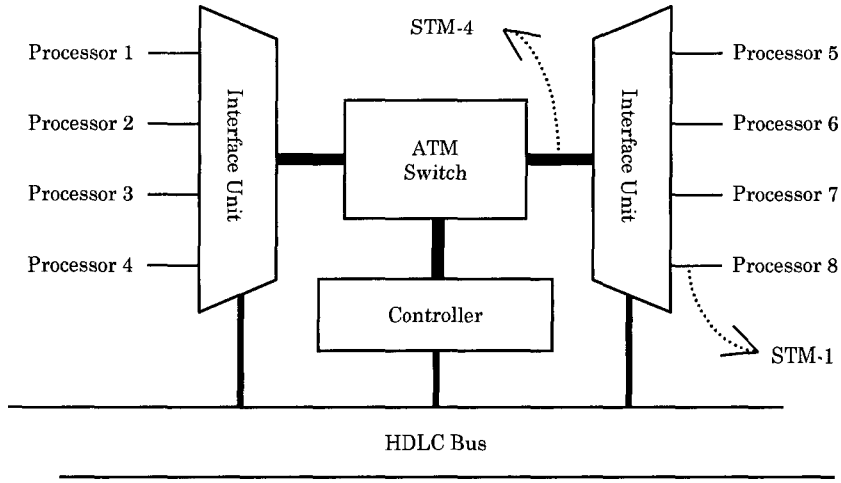


Figure 1. An example illustration of processor assignment

The PAP can be found from configuring the processors in the telecommunication systems such as RNC (Radio Network Controller), a component of WCDMA (Wideband Code Division Multiple Access) IMT-2000 network. In general, there are tens of processors connected by one ATM switch module in a RNC. The use of ATM technology is specified in the WCDMA standards by 3GPP (2004). Since a RNC covers tens of Node-Bs, equivalent to base stations, the traffic load is usually very high. Also, considering that at least several thousands of radio links terminate at a single RNC, the reliability and performance of RNC is important. And, both the reliability and performance of RNC can be improved by reducing the load of ATM switch controller while supporting the required system capacity. In this respect, we consider the PAP that minimizes the transaction load of ATM switch controller for improving the reliability of RNC.

The PAP can be conceptualized as a graph-partitioning problem. The simplest form of graph-partitioning problem is to divide a node set into two disjoint

subsets, so called the 2-way partitioning problem. The 2-way partitioning problem that maximizes the sum of edge weights inter-connecting the two subsets, referred to as the max-cut problem, is NP-complete (see Ahuja *et al.*, 1993). Perhaps, the most popular heuristics for the 2-way partitioning problem are the KL algorithm by Kernighan and Lin (1970) and the simulated annealing (SA) algorithm by Kirkpatrick *et al.* (1983). Recently, Kim and Moon (2004) developed a new heuristic, so called the "lock-gain algorithm", based on the KL algorithm for the 2-way partitioning problem. According to Kim and Moon (2004), the lock-gain algorithm combined with the genetic algorithm framework outperforms the previously known best solution for most of the benchmark test problem instances.

For the k-way partitioning problem that divides a node set into k (≥ 2) disjoint subsets, there are several studies on developing meta-heuristic such as genetic algorithm by Bui and Moon (1996), simulated annealing combined with tabu search by Gil *et al.* (2002) and so forth. For the k-way partitioning problem, some distinguished studies on deriving tight lower and upper bounds include the works by Minoux (1995), by Adil and Ghosh (1999), and by Tu *et al.* (2000) that are based on the LP-relaxation, on the Lagrangian–relaxation, and on the eigenspace-relaxation, respectively. Recently, Myung (2003) presented an intensive survey on the graph-partitioning problem, which describes various formulations and solution procedures.

Randall and Abramson (1999) developed a parallel tabu search algorithm. Meanwhile, Banos *et al.* (2004) presented a parallel simulated annealing algorithm combined with tabu search, and Vigo and Maniezzo (1997) presented a genetic algorithm combined with tabu search for the processor allocation problem. They employed genetic algorithm or simulated annealing as a main framework, whereas tabu list is used only for preventing cycles. Also, Wiangtong *et al.* (2002) compared the performance of genetic algorithm, simulated annealing and tabu search for solving the graph partitioning problem. However, in this paper, we develop an effective tabu search heuristic for solving the processor assignment problem. The main contribution of this paper is stated as follows. First, we show that we can obtain promising computational result by focusing on generating elite solutions, instead of focusing on the move evaluation. Second, we develop a new tabu search framework for finding a good quality solution in determining the optimal number of clusters.

The remainder of the paper is organized as follows. Section 2 develops an integer programming (IP) formulation, and investigates the inherent special structure of the problem. Section 3 prescribes a tabu search heuristic. Section 4 presents computational results and concludes the paper.

2. FORMULATION

To precisely describe the processor assignment problem (PAP), we define N as a set of processors, K as a set of interface units and E as a set of pairs of traffic demands d_{ij} (≥ 0) between processors i and j ($> i$) $\in N$, where d_{ij} represents the number of requests for setting up SVCs between processors i and j ($> i$) $\in N$. Define β as the maximum number of processors that are supported by an interface unit. Also, we define $N_m \subseteq N$ as a set of identical processors of type $m \in M$, where M is a set of processor types, and we define r_m as the maximum number of processors of type $m \in M$ that can be assigned to an interface unit. Now, we define decision variables. Let $x_{ik} = 1$ if processor $i \in N$ is assigned to interface unit $k \in K$, and 0 otherwise. Also, let $y_{ij} = 1$ if processors i and j ($> i$) $\in N$ are assigned to different interface units, and 0 otherwise. Then, we can formulate the PAP as follows.

PAP: Minimize $\sum_{(i,j)\in E} d_{ij} y_{ij}$

subject to

$$\sum_{k\in K} x_{ik} = 1, \qquad \forall i \in N, \tag{1}$$

$$y_{ij} \geq x_{ik} - x_{jk}, \qquad \forall (i,j) \in E,\ k \in K, \tag{2}$$

$$y_{ij} \geq x_{jk} - x_{ik}, \qquad \forall (i,j) \in E,\ k \in K, \tag{3}$$

$$\sum_{i\in N} x_{ik} \leq \beta, \qquad \forall k \in K, \tag{4}$$

$$\sum_{i\in Nm} x_{ik} \leq r_m, \qquad \forall m \in M,\ k \in K, \tag{5}$$

$$x_{ik} \in \{0,1\}, \qquad \forall i \in N,\ k \in K,$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in E.$$

The objective function is to minimize the sum of traffic demand switched among processors that are assigned to different interface units. Constraints (1) limit that each processor is assigned to an interface unit. Constraints (2) and (3) show the logical relationship between variables x_{ik} and y_{ij}. Constraints (4) ensure that the maximum number of processors assigned to each interface unit is β. Constraints (5) indicate that at most r_m identical processors of type $m \in M$ are assigned to a single interface unit. The requirement of constraints (5) is essential in the multi-processor architecture design for improving the reliability of a system. For example, suppose that we have three identical processors performing the same task for load sharing or for the survivability of a system. Also, suppose that

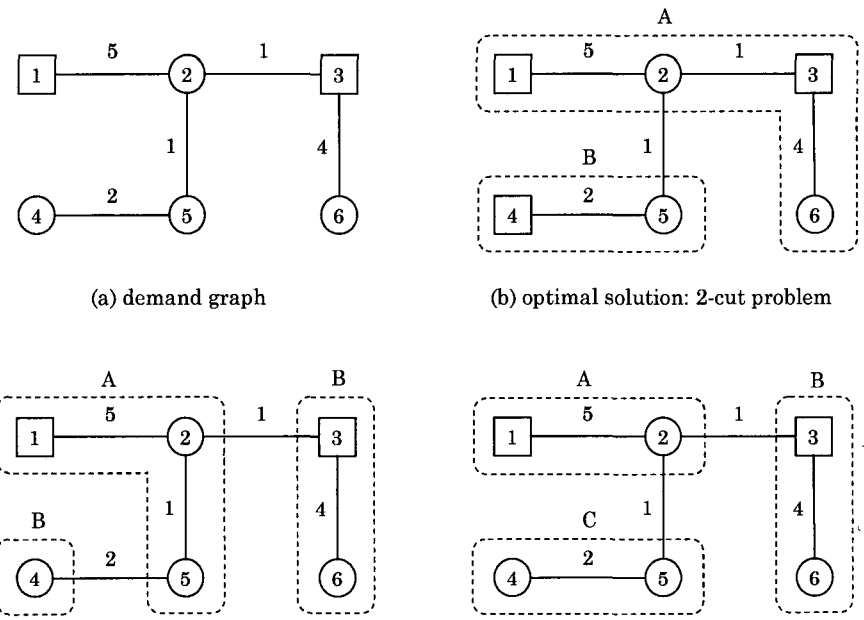
we have assigned all of them to a single interface unit. In this case, if the interface unit fails, there is no way to maintain the survivability of the system.

Remark 1. It is evident that the PAP is NP-hard since, by deleting the constraints (5) from the PAP, it reduces to the clustering problem having cardinality constraints for each cluster, which is NP-hard (Myung, 2003). Also, note that constraint (4) forces to use at least $\lceil |N|/\beta \rceil$ clusters for finding a feasible solution to the PAP. Thus, the PAP can be conceptualized as a $\lceil |N|/\beta \rceil$-cut problem with additional constraints (4) and (5), where k-cut problem is to divide a set of nodes into at least k disjoint subsets while minimizing the sum of edge weights interconnecting the subsets. Here, note that there exists an optimal solution to the k-cut problem dividing a set of nodes to only k disjoint subsets (Myung, 2003). However, unlike the k-cut problem, we may not find an optimal solution to the PAP by using only $\lceil |N|/\beta \rceil$ interface units due to the constraints (4) and (5), which is illustrated in the following example. ▲

Example 1. Suppose that $N = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1,2), (2,3), (2,5), (3,6), (4,5)\}$ with SVC demands between processors as follows: $d_{12} = 5$, $d_{36} = 4$, $d_{45} = 2$, $d_{23} = d_{25} = 1$. These input parameters are illustrated in Figure 2(a). In Figure 2(b), we present an optimal solution for the 2-cut problem of this example, where we can find an optimal solution with two interface units, i.e., $K = \{A, B\}$. That is, processors 1, 2, 3 and 6 are assigned to interface unit A, and processors 4 and 5 are assigned to interface unit B, which gives an objective value of one. Here, we see that we cannot find a better solution than the one that presented in Figure 2(b) even if we use more than 2 interface units.

Now, let us consider the foregoing example problem with the additional constraints (4) and (5). For this purpose, let $\beta = 3$. Also, we consider a single set of identical processors such as $N_1 = \{1, 3\}$ with $r_1 = 1$. That is, we cannot assign both processors 1 and 3 to a common interface unit. If $K = \{A, B\}$, we have an optimal solution with the objective value of three by assigning processors 1, 2 and 5 to interface unit A, and processors 3, 4 and 6 to interface unit B, which is illustrated in Figure 2(c). However, despite that $\lceil |N|/\beta \rceil = 2$, if we use three interface units, i.e, $K = \{A, B, C\}$, we obtain an optimal solution with the objective value of two by assigning processors 1 and 2 to interface unit A, processors 3 and 6 to interface unit B, and processors 4 and 5 to interface unit C. This is illustrated in Figure 2(d). As seen in this example, due to constraints (4) and (5), we may not find an optimal solution for the PAP with only $\lceil |N|/\beta \rceil$ interface units unlike the case of generic k-cut problem. Thus, in the PAP, we cannot reduce the problem size by

letting $|K| = \lceil |N|/\beta \rceil$ even if $|K|$ is far greater than $\lceil |N|/\beta \rceil$. ▲



(a) demand graph

(b) optimal solution: 2-cut problem

(c) optimal solution: w/ constraint (5), two interface units

(d) optimal solution: w/ constraint (5), three interface units

Figure 2. An example of the problem

3. SOLUTION PROCEDURE

For finding a feasible solution of good quality, we develop a tabu search heuristic. For detailed information of tabu search, refer to Glover and Laguna (1997). In developing the tabu search for the problem, we focus on generating and managing elite solutions and implement a direct method for changing the number of clusters instead of using step-wise moves.

For finding an initial feasible solution, we develop a simple heuristic procedure, stated as follows. Let us begin the processor assignment with $|K|$ interface units. First, we choose an arbitrary processor type $m \in M$ and assign $|N_m|$ identical processors to $min\{|N_m|, |K|\}$ interface units in a round-robin manner. Then, we choose another processor type l ($\neq m$) $\in M$ at random and assign all the processors in N_l to $min\{|N_l|, |K|\}$ interface units in a round-robin manner. We repeat this procedure until all the processors in $U_{m \in M} N_m$ are assigned. Then, we

assign all the processors in $N - \mathrm{U}_{m \in M} N_m$ to $min\{|N - \mathrm{U}_{m \in M} N_m|, |K|\}$ interface units in a round-robin manner. If we find a feasible assignment, we try to find another one after reducing the number of interface units $|K|$ by one. We repeat this procedure until we cannot find a feasible assignment. Then, we choose the best one among several feasible assignments as an initial solution. Our initial heuristic procedure is described as follows.

Initialization. Let $\eta = |K|$ and $F = \varnothing$, where F denotes the set of feasible assignments.

Step 1. Let $M\hat{} = M$ and $c = 0$, where c is an interface unit index. Here, we assume that interface unit begins with index 0.

Step 2. Pick an arbitrary processor type $m \in M\hat{}$ and assign $|N_m|$ identical processors to $min\{|N_m|, \eta\}$ interface units in a round-robin manner beginning with the interface unit of index c. Then, let $M\hat{} = M\hat{} - \{m\}$ and $c = \{[c + (|N_m| \ mod \ \eta)] \ mod \ \eta\}$. If we fail to assign all the processors in N_m to $min\{|N_m|, \eta\}$ interface units, go to Step 4.

Step 3. If $M\hat{} \neq \varnothing$, go to Step 2. Otherwise, assign all the processor in $N - \mathrm{U}_{m \in M} N_m$ to $min\{|N - \mathrm{U}_{m \in M} N_m|, \eta\}$ interface units in a round-robin manner beginning with the interface unit of index c, and add the current solution of processor assignment to F. Then, let $\eta = \eta - 1$ and go to Step 1. If we fail to assign all the processors in $N - \mathrm{U}_{m \in M} N_m$ to $min\{|N - \mathrm{U}_{m \in M} N_m|, \eta\}$ interface units, go to Step 4.

Step 4. Choose the best assignment in F and set it as an initial solution, and stop.

Next, we develop tabu search heuristic for improving the initial solution. For this purpose, we define moves that would work as a principal mechanism for improving the incumbent solution, and we develop three phases such as intensification phase, normal phase, and diversification phase. Then, we define tabu list for preventing recent moves from reverse-implementing.

Moves: We define add-move as relocating a processor to a different interface unit. Also, we define swap-move as exchanging interface unit assignment between a pair of processors. Our tabu search heuristic begins with the move at the intensification phase, which is described in the following.

Intensification Phase: We improve the incumbent solution by performing add-moves and swap-moves repeatedly. During the intensification phase, the move type, i.e., add-move or swap-move, is randomly chosen. Also, in order to reduce the computation time, we perform the first move that improves the incum-

bent solution. However, if we cannot find any move that improves the incumbent solution, we switch to the normal phase.

Normal Phase: We consider both add-move and swap-move to find a better feasible solution without increasing the number of interface units. That is, the add-move that reduces the total number of interface units is allowed, while the add-move that increases the total number of interface units is not allowed. The task of the normal phase is to proceed to a new solution space that would hopefully give a chance to improve the current best solution when returned to the intensification phase. Thus, in the normal phase we do not prevent any move that increases the objective value. However, this may result in a new feasible solution that is too far from the current best solution in terms of the objective value, in which case we may have to consume so much computation time to catch up with or overcome the current best solution. Thus, we perform the add-move or the swap-move that provides a new feasible solution without increasing the objective value if possible, and return to the intensification phase. In this process, swap-move is preferred to add-move. However, if such a move is not found, we perform Max_N random moves without evaluating the objective value. In this case, the selection of move type, i.e., add-move or swap-move, is at random. If we find a new solution, we return to the intensification phase after updating the set L of feasible solutions. The cardinality of L is limited to some predetermined value Max_L, and the set L is managed by the first-in first-out (FIFO) strategy in the normal phase. Here, note that we do not update the set L in the intensification phase. However, if we fail to find a new solution during the iterations of Max_N move, we switch to the diversification phase.

Diversification Phase: The task of the diversification phase is to generate a new feasible solution as the case of normal phase. However, there are some differences, which are listed as follows. In the diversification phase,

1) we generate a new feasible solution based on an arbitrary solution in L, not based on the incumbent solution,
2) we do not perform any add-move that decreases the total number of interface units,
3) we do not evaluate moves in terms of the objective value at all, and
4) we delete the worst solution, not the most aged solution, in L before adding a new solution to L when $|L| = Max\_L$.

The rationale that we seek to find a new feasible solution by operation 1) is that we fail to enter a new feasible region that is not investigated by sequential

moves from the incumbent solution of the normal phase. Thus, we need to change the combination of the incumbent solution and tabu lists. That is, by changing the incumbent solution to some different one in L a new combination of the incumbent solution and tabu lists may be constructed, and this may give us opportunity to overcome the current best solution.

The rationale behind the operation 2) is that we do not know the optimal number of interface units and that we fail to find a new feasible solution in the normal phase by moves, except the add-move that increases the total number of interface units.

The rationale behind the operation 3) is that we fail to find a new feasible solution in the normal phase, and thus it is better to focus only on finding a new feasible solution. However, this may aggravate the average quality of the solutions in L. In order to compensate the operation 3), we adopt the operational strategy of 4) in the diversification phase.

Tabu List and Aspiration: There are two types of recency based tabu lists: one for the add-move and the other for the swap-move, each being denoted by T_A and T_S, respectively. That is, both tabu lists T_A and T_S are managed by the FIFO strategy. Tabu list T_A defines a set of add-moves not to be implemented for some iterations. Also, tabu list T_S defines a set of swap-moves not being implemented for some duration. However, tabu lists T_A(or T_S) can be overridden by add-move(or swap-move) if the move improves the incumbent solution. Due to this aspiration criteria, we do not examine tabu lists in the intensification phase.

The purpose of incorporating tabu lists into the move evaluation is to prevent the incumbent solution from returning to the previously visited solution. However, if we permanently prevent all the moves that were implemented from reverse-implementing, we may be stalled at some stage. Thus, we need to carefully determine the size of tabu lists and the duration of each element in the tabu lists, referred to as tabu tenure. In order to find a good combination of the size of tabu lists and tabu tenure, we perform preliminary experiments, which is described in the next section.

4. COMPUTATIONAL RESULTS AND CONCLUSION

We have tested both the proposed tabu search heuristic for randomly generated problem instances. Also, in order to evaluate the performance, we have run the

MIP optimization procedure of CPLEX 9.0 (CPLEX, 2004). Both our tabu search heuristic and the CPLEX MIP optimization procedure were run on a Pentium IV PC (CPU clock speed = 2.8GHz, RAM = 512MB). For generating test problem instances, we choose the capacity of ATM switch first. In this paper, we consider 8 × 8 STM-4 ATM switch (4.8Gbps) since this is typically used for switching data streams between processors inside the telecommunication systems such as RNC in the WCDMA IMT-2000 system, where 8×8 STM-4 indicates that the ATM switch can support eight STM-4 data streams simultaneously (WCDMA system architecture group, 2002). This means that $|K| = 8$ and $\beta = 4$ if one interface unit supports 4 STM-1 data streams for processors. Thus, 8 × 8 STM-4 ATM switch supports up to 32 processors, each being assigned a single STM-1. Therefore, we generated $|N|$ (≤ 32) processors, where we assumed that there are three sets of identical processors such that ($|N_m|$, r_m) = (3, 2), (4, 2) and (5, 2) for m = 1, 2, and 3. Then, we generated $|E|$ undirected pairs of traffic demands between processors according to the uniform distribution in the range of [1, 5]. In general, estimating the traffic demand between processors may incur complicated traffic engineering analysis for real-world problem. However, we may quite accurately estimate the traffic demand between processors in some special cases. For example, in a mobile telecommunication system, we define several classes of service in terms of the connection type (circuit or packet) and the data rate, i.e., 12.2Kbps circuit based voice service and packet based data service supporting up to 64Kbps, 144Kbps, 384Kbps and 2Mbps. Moreover, we can further breakdown each class of service by call type, i.e., initial incoming call and handover call. For each combination of service class and call type, the call flow describing the set of tasks and the sequence of tasks may be different (WCDMA system architecture group, 2002). For example, the call flow of 12.2Kbps circuit based voice call for initial connection can be processor 1 → processor 2 → processor 4, while the call flow of 64Kbps packet based data call for handover can be processor 1 → processor 3 → processor 4. Now, the remaining issue is to estimate the frequency of realizations for each combination of service class and call type in the phase of commercial service roll-out. This problem is partially solved by analyzing the traffic log during the trial period.

To set the parameters of our tabu search heuristic, we conducted some preliminary experiments. From the preliminary experiments, we learned that $Max\_N = |N|$, $Max\_L = |N| \times |K|$ and $|T_A| = |T_S| = |N| \times |K|$ is a good combination in terms of both the computation time and solution quality. Also, we observed that the tabu tenure of each element in tabu lists T_A and T_S in conjunction with the size of tabu lists $|T_A|$ and $|T_S|$ significantly influences on the performance of our

tabu search heuristic. If tabu tenure is too long, solution quality tends to be low. On the other hand, we obtained high quality solutions with small variance when the tabu tenure is set to a small value as long as it is not too small, i.e., 1 or 2. Also, we observed the recurrence of consecutive moves that were considered as tabu when the tabu tenure for all tabu moves is set to a common value. Thus, we have set the tabu tenure for each element of tabu lists to some integer value being generated by the uniform distribution in the range of $[|N|/2, |N|]$.

In Tables 1, 2 and 3, we present some computational results, where "LP" indicates the LP-relaxation lower bound, "CPLEX" indicates the upper bound obtained by CPLEX MIP optimization procedure, "Tabu" indicates the upper bound obtained by the proposed tabu search heuristic, and "GAP" indicates the percentage gap of "Tabu" upper bound compared to "CPLEX" upper bound. Also, "CPLEX_T" indicates the CPLEX run time, measured in seconds. Asterisk mark(*) in "CPLEX_T" indicates that we interrupted the CPLEX MIP optimization procedure at that time. Thus, "CPLEX" upper bound is equal to an optimal objective value if the corresponding "CPLEX_T" is presented with no asterisk mark. CPU time for tabu search heuristic is limited to 60 seconds for all the test problem instances. As we see from the computational results of Tables 1, 2 and 3, the CPLEX found an optimal solution for 19 test problems out of 60 within 10,000 seconds. On the other hand, our tabu search heuristic found a feasible solution within 5% of optimality gap for 53 test problems. Also, note that the largest gap of our tabu search heuristic does not exceed 5.8%. For 22 test problems we observe that the proposed tabu search heuristic generates a good feasible solution compared with the CPLEX run up to 10,000 seconds.

Table 1. Computational results: $(|N|, |K|, \beta) = (15, 8, 4)$.

$\|E\| = 40$						$\|E\| = 50$					
No.	LP	CPLEX	Tabu	Gap	CPLEX_T	No.	LP	CPLEX	Tabu	Gap	CPLEX_T
1	1	59	61	3.4%	1,204	1	3	85	85	0%	10,000*
2	2	76	78	2.6%	5,433	2	4	100	101	1%	10,000*
3	1	69	73	5.8%	8,893	3	3	77	81	5.1%	3,263
4	2	72	72	0%	4,150	4	3	77	77	0%	9,487
5	2	61	61	0%	460	5	3	92	92	0%	10,000*
6	1	66	68	0%	6,917	6	2	93	97	4.3%	10,000*
7	3	65	67	3.1%	3,368	7	3	85	85	0%	10,000*
8	3	54	55	1.9%	1,523	8	1	91	96	5.5%	10,000*
9	2	74	75	1.4%	5,682	9	3	89	89	0%	10,000*

10	2	65	66	1.5%	1,480	10	2	83	84	1.2%	6,089

Table 2. Computational results ($|N|, |K|, \beta$) = (20, 8, 4).

$\|E\| = 40$						$\|E\| = 50$					
No.	LP	CPLEX	Tabu	Gap	CPLEX_T	No.	LP	CPLEX	Tabu	Gap	CPLEX_T
1	1	52	55	5.7%	10,000*	1	1	77	81	5.2%	10,000*
2	2	59	62	5.1%	8,083	2	3	80	80	0%	10,000*
3	0	65	67	3.1%	10,000*	3	1	79	81	2.5%	7,310
4	2	65	67	3.1%	10,000*	4	1	84	86	2.4%	10,000*
5	3	57	57	0%	499	5	1	79	79	0%	10,000*
6	3	56	56	0%	9,399	6	1	88	88	0%	10,000*
7	1	49	51	4.1%	10,000*	7	2	71	71	0%	10,000*
8	1	58	58	0%	10,000*	8	1	83	83	0%	10,000*
9	0	58	58	0%	10,000*	9	3	85	88	3.5%	10,000*
10	1	59	61	3.4%	1,720	10	0	68	70	2.9%	10,000*

Table 3. Computational results: ($|N|, |K|, \beta$) = (25, 8, 4).

$\|E\| = 40$						$\|E\| = 50$					
No.	LP	CPLEX	Tabu	Gap	CPLEX_T	No.	LP	CPLEX	Tabu	Gap	CPLEX_T
1	1	53	53	0%	10,000*	1	1	72	71	-1.4%	10,000*
2	0	45	46	2.2%	10,000*	2	0	69	71	2.9%	10,000*
3	0	47	49	4.3%	7,967	3	1	78	80	2.6%	10,000*
4	1	53	54	1.9%	10,000*	4	0	61	62	1.6%	10,000*
5	0	59	62	5.1%	10,000*	5	1	65	67	3.1%	10,000*
6	1	44	46	4.5%	10,000*	6	1	75	76	1.3%	10,000*
7	2	53	55	3.8%	10,000*	7	2	68	70	2.9%	10,000*
8	2	50	51	2%	10,000*	8	1	72	72	0%	10,000*
9	1	41	41	0%	10,000*	9	1	74	74	0%	10,000*
10	1	51	53	3.9%	10,000*	10	1	83	83	0%	10,000*

In this paper, we proposed a graph-partitioning problem with cardinality constraints. Also, we developed an effective tabu search heuristic to solve the processor assignment problem that minimizes the total traffic load of an ATM switch controller by optimally assigning processors to ATM interface units. Computational results show that our tabu search heuristic works quite well usually finding a good quality solution within 5% of optimality gap. Further work includes investigating the polyhedral structure of the proposed graph-partitioning

problem, and developing a branch-and-cut algorithm.

REFERENCES

[1] CPLEX, *Using the CPLEX Callable Library*, Version 9.0, ILOG, Inc., 2004.

[2] Adil, G. K. and J. B. Ghosh, "Analysis of Lagrangian lower bounds for a graph partitioning problem," *Operations Research* 47 (1999), 785-788.

[3] Ahuja, R. K., T. L. Magnanti, and J. B. Orlin, *Network Flows*, Prentice Hall, New Jersey, 1993.

[4] Banos, R., C. Gil, J. Ortega and F. G. Montoya, "A parallel multilevel metatheuristic for graph partitioning," *Journal of Heuristics* 10 (2004), 315-336.

[5] Bui, T. N. and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Transaction on Computers* 45(1996), 841-855.

[6] Gil, C., J. Ortega, M. G. Montoya, and R. Banos, "A mixed heuristic for circuit partitioning," *Computational Optimization and Applications* 23(2002), 321-340.

[7] Glover, F. and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[8] Kernighan, B. and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Systems Technical Journal* 49 (1970), 291-307.

[9] Kim, Y. H. and B. R. Moon, "Lock-gain based graph partitioning," *Journal of Heuristics* 10 (2004), 37-57.

[10] Kirkpatrick, S., C. D. Gelatt Jr. and M. P. Vecchi, "Optimization by simulated annealing," *Science* 220 (1983), 671-680.

[11] Minoux, M., "On some large-scale L P relaxations for the graph partitioning problem and their optimal solutions," *Annals of Operations Research* 58 (1995), 143-154.

[12] Myung, Y. S, "The graph partition problem," *Journal of the Korean Operations Research and Management Science Society* 28 (2003), 131-143.

[13] Randall, M. and A. Abramson, "A parallel tabu search algorithm for combinatorial optimization problems," In *Proceedings of 6th Australasian Conference on Parallel and Real Time Systems*, Springer-Verlag, (1999), 68-79.

[14] Tu, C. C., C. K. Shieh, and H. Cheng, "Algorithms for graph partitioning problems by means of eigenspace relaxations," *European Journal of Operational Research* 123 (2000), 86-104.

[15] Vigo, D. and V. Maniezzo, "A genetic/tabu thresholding hybrid algorithm for

the process allocation problem," *Journal of Heuristic* 3(1997), 91-110.

[16] WCDMA system architecture group, *Technical description of WCDMA RNC system*, LG Electronics, Inc., 2002.

[17] Wiangtong, T., P. Cheung and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware-software codesign," *Design Automation for Embedded Systems* 6 (2002), 425-449.

[18] 3GPP, http://www.3gpp.org/ftp/Specs/html-info/25-series.htm, 2004.