Title:

Mapping Flows onto Networks to Optimize Organizational Processes*

(*student paper*)

Suggested Track:

Modeling and Simulation

(Sessions on Adaptive Architectures for Command and Control (A2C2))

Authors:

Georgiy M. Levchuk
Graduate Student, ECE Dept., UCONN
Storrs, CT
Fax: 860-486-5585
Phone: 860-486-2210
e-mail: georgiy@engr.uconn.edu

Yuri Levchuk
Aptima, Inc.
Woburn, MA
Phone: 781-935-3966x236
e-mail: levchuk@aptima.com

Krishna R. Pattipati
Professor, ECE Dept., UCONN
Storrs, CT
Fax: 860-486-5585
Phone: 860-486-2890
e-mail: krishna@sol.uconn.edu, krishna@engr.uconn.edu

David L. Kleinman
Professor, Naval Postgraduate School
589 Dyer Road, Room 200A
Monterrey, CA
Fax: 831-656-3679
Phone: 831-656-4148
e-mail: kleinman@nps.navy.mil

Correspondence:

Prof. Krishna R. Pattipati

Dept. of ECE, The University of Connecticut

260 Glenbrook Road

Storrs, CT 06269-2157

Phone: (860)-486-2890

Fax: (860)-486-5585

e-mail: krishna@sol.uconn.edu

* This work was supported by the Office of Naval Research under contract # N00014-00-1-0101.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2005		**00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Mapping Flows onto Networks to Optimize Organizational Processes	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Office of Naval Research,One Liberty Center,875 North Randolph Street Suite 1425,Arlington,VA,22203-1995**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**24**	

MAPPING FLOWS ONTO NETWORKS TO OPTIMIZE ORGANIZATIONAL PROCESSES*

Georgiy M. Levchuk

Dept. of ECE, University of Connecticut

Storrs, CT 06269-2157

georgiy@engr.uconn.edu

Yuri N. Levchuk
Aptima, Inc.
Woburn, MA
levchuk@aptima.com

Krishna R. Pattipati

Dept. of ECE, University of Connecticut

Storrs, CT 06269-2157

krishna@engr.uconn.edu

David L. Kleinman
Naval Postgraduate School, 589 Dyer Road, Room 200A
Monterrey, CA
kleinman@nps.navy.mil

* This work was supported by the Office of Naval Research under contract # N00014-00-1-0101.

Abstract

Interdependence of tasks in a mission necessitates information flow among the organizational elements (agents) assigned to these tasks. This information flow introduces communication delays. An effective task schedule that minimizes the total execution time, including task processing and coordination delays, is an important issue in designing an organization and its task processing strategy. This paper defines the structure of information-dependent tasks, and describes an approach to map this structure to a network of organizational elements (agents).

Since the general problem of scheduling tasks with communication is NP-hard, only fast heuristic (list scheduling and linear clustering) algorithms are discussed. We modify the priority calculation for list scheduling methods, matching the critical path with a network of heterogeneous agents. We present our algorithm, termed Heterogeneous Dynamic Bottom Level (HDBL), and compare it with various list-scheduling heuristics. The results show that HDBL exhibits superior performance to all list scheduling algorithms, providing an improvement of over 25% in schedule length for communication-intensive task graphs.

1 Introduction

Contingency theorists argue, and the empirical research confirms, that a proper organizational design is critical to superior organizational performance ([Burton98], [Entin99], [Hocevar99]). Setting up efficient organizational processes is one of the keys to a successful organizational design. Formalizing the team processes provides a basis for identifying the design parameters that can be optimized to improve team performance.

From a systems theory viewpoint, an organization is an *open system*. It can be modeled by specifying several key entities including: (i) the environment; (ii) the organizational elements (sometimes termed *agents* or processors); (iii) the organizational structure; (iv) the organizational processes; and (v) the organizational outcomes. While mission decomposition into tasks provides a basis for balancing the effort among agents, the input-output transformations that link tasks define the "flows" within the organization and/or between the organization and its environment. Two important examples of such flows include the information flow (specifying communication among the agents) and commodity flow (e.g., the production cycle that transforms the raw materials into ready-to-sell products). The corresponding flows (from hereon termed process flows) characterize the organizational processes by specifying input-output relationships among the organizational elements.

The problem of optimizing team processes can be decomposed into three parts: (1) optimizing the functional allocation strategy to achieve desired goal states; (2) decomposing functions into sets of interdependent tasks; and (3) mapping the tasks and their process flows onto an organization to optimize the processing cost. The solution to the first of these three problems from two different perspectives, dynamic Bayesian networks and Markov Decision Processes, is presented in [Meirina2002] and [Tu2002]. An important feature of the process flows is that the *flow medium* can change its content and volume after passing through any of the processing nodes (e.g., the information can be filtered or fused).

A coherent, timely, and efficient team process greatly improves the chances for successful team performance. Therefore, mapping process flows onto an organization is an important issue affecting team performance [Levchuk2002], since it specifies two sets of variables: (i) an allocation of tasks to agents (or system elements), and (ii) requirements for the flow of information among agents. The task

processing schedule and flow routing can be optimized to minimize schedule inefficiencies (i.e., delays), utilized resources, expended energy, coordination overhead, and so on.

The problem of scheduling tasks on a processor architecture is of considerable importance in parallel processing. The general problem of scheduling a task graph to minimize the total parallel time is NP-hard (no polynomial-time algorithm exists to find an optimal solution). Even if communication among task nodes is zero, we obtain a simplified scheduling problem that is still NP-hard. For this problem, it was shown that any list scheduling heuristic is within 50% of the optimal solution (see [Graham66]). It was later empirically demonstrated ([Adam74]) that the critical path list scheduling method has even better performance: its solution is within 5% of the optimum in 90% of cases.

In the presence of inter-task communication, however, the problem becomes much harder. List scheduling no longer has the 50% performance guarantee. The problem of scheduling tasks with communication has received much attention [Wu88], [Kim88], [Sarkar89], [McCreary90], [El-Rewini90a&b], [Gerasoulis90a&b], [Sih93] since it can be used for scheduling tasks on message-passing architectures. The one-stage approach (assigning tasks to physical agents) of list scheduling method was contrasted with two-stage methods in which the first stage performs reduction clustering and preprocessing that explore the topology of communication graph regardless of agent constraints. However, the problem becomes significantly complex when various constraints are introduced, and the two-stage methods can no longer be applied.

This paper is organized as follows. The problem of scheduling information tasks onto an agent structure is formulated in section 2. Section 3 outlines scheduling and mapping variables used for our problem. The mapping and scheduling feasibility conditions are discussed in section 4. The solution approach, together with related research, is presented in sections 5-6. List scheduling method is discussed in section 6, with previous work outlined in subsection 6.1 and our algorithm presented in subsection 6.2. Section 7 presents simulation results. Conclusions and future extensions are given in section 8. The issue of mapping a critical path onto a heterogeneous system is presented in Appendix A, and an efficient algorithm for scheduling multiple information messages with release times onto agents' communication link is outlined in Appendix B.

This paper provides the following contributions to the methodology of scheduling task communication graphs onto a heterogeneous system of agents:

- The agent network structure is utilized to find the dynamic critical path, the earliest possible start time and the latest possible finish time of a task on an agent (Appendix A);
- The dynamic critical path mapped to agent network is utilized to compute agent-task priorities in HDBL algorithm that outperforms conventional list-scheduling methods; and
- Efficient algorithm is developed for scheduling multiple information messages with release times onto an agent's communication link (Appendix B).

2 Problem Statement

The goal of an organization is to complete assigned missions in the most efficient manner. Each mission can be decomposed into a set of tasks with specific constraints [Levchuk2000a], [Kapasouris91]. One of the constraints of the mission is a particular ordering in which the tasks must be completed (a mission plan). An organization consists of *agents* (processors, human decision-makers, etc.). Each agent has certain resources available to it, which, together with training (for human organizations), determine the

expertise to carry out assigned tasks/processes. Generally, agents have different capabilities to perform tasks. Processing elements with different capabilities are termed *heterogeneous*.

This paper addresses the problem of finding the optimal allocation (scheduling) or *mapping* of tasks with communication requirements (called *information* tasks) to heterogeneous *agents* (organizational elements) while satisfying various constraints. The objective function is to minimize the completion time of the mission, that is, the finish time of the terminal task (also termed *the makespan*).

TABLE I.
Task Attributes

$G_t = (V_t, E_t)$	directed acyclic graph of tasks with precedence constraints
$T_i \in V_t$	task node
$e^t_{i,j} =< T_i, T_j > \in E_t$	a precedence arc in the task graph
f_{ij}	amount of information transmitted between tasks T_i and T_j along the arc $e^t_{i,j}$
w_i	task processing load (*workload*)
m_i	task memory load

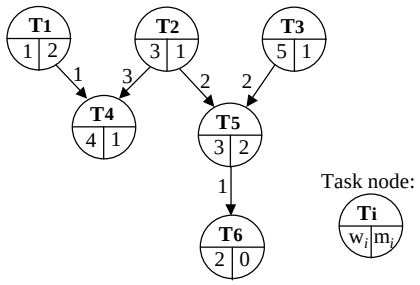

Figure 1: Information Graph

Information tasks are modeled via directed acyclic *information graph* $G_t = (V_t, E_t)$, where $V_t = \{T_i, i = 1,..., N\}$ is the set of task nodes, $N = |V_t|$ is the number of nodes, $E_t = \{e^t_{i,j} =< T_i, T_j >\}$ is the set of directed edges, and $e_t = |E_t|$ is the number of edges. Edges in the graph correspond to communication messages and precedence constraints among tasks. Amount of information (weight of communication) transmitted from task T_i to T_j (incurred along the edge $e^t_{i,j} =< T_i, T_j >$) is denoted by $f_{i,j}$, which becomes zero if both tasks are allocated to the same agent. For each task T_i, a processing load (or *workload*) w_i and memory load m_i are defined. Task attributes are outlined in Table I. Fig. 1 shows an example of a task graph, with weights on the arcs representing the information flow transferred between the concomitant tasks, and weights on the task nodes indicating task workload and memory load.

The agents are modeled via another dependency graph. The agent structure is defined by an undirected graph $G_a = (V_a, E_a)$, where $V_a = \{A_r, r = 1,..., K\}$ is the set of agent nodes and $K = |V_a|$ is the number of nodes, and $E_a = \{e^a_{r,u} =< A_r, A_u >\}$ is the set of undirected communication links among agents with transfer rate $c_{r,u}$, and $e_a = |E_a|$ is the number of links. For each agent A_r, a processing (or *workload*) capacity W_r and a memory capacity M_r are defined, and the time to process task T_i is $p_{r,i}$ ($p_{r,i} = \infty$ if the agent cannot process this task; we assume that $p_{r,i} = \infty$ if $W_r < w_i$).

Agent attributes are outlined in Table II. Fig. 2 shows an example of an agent network, with weights on the arcs representing the rate of information transfer between the corresponding agents, and weights on the agent nodes indicating agent workload and memory capacity. Table III shows the agent-task processing time matrix.

The execution model works as follows (for a similar macro-dataflow model, see [Sarkar89], [Wu88]). The data flow triggers the execution of tasks. A task receives all data from its predecessors in parallel. It then executes without interruption (non-preemptively) and immediately after completion it sends the data to all successors in parallel. In this model, task execution and agent communication are done in parallel subject to constraints on workload and memory capacities, and communication contention.

TABLE II.
Agent Attributes

$G_a = (V_a, E_a)$	non-directed graph of agents with communication links
$A_r \in V_a$	agent node
$e^a_{r,u} =< A_r, A_u >\in E_a$	a communication link
c_{ru}	rate of information transfer between agents A_r and A_u along the arc $e^a_{r,u}$
$p_{r,i}$	time required to process task T_i
W_r	agent processing (*workload*) capacity
M_r	agent memory capacity

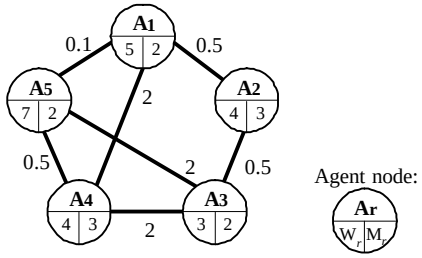

Figure 2: Agent Network

TABLE III.
Agent-Task Processing Time

	Task **T1**	Task **T2**	Task **T3**	Task **T4**	Task **T5**	Task **T6**
Agent **A1**	3	1	2	1	∞	3
Agent **A2**	∞	2	∞	1	∞	1
Agent **A3**	1.5	1	∞	∞	∞	0.5
Agent **A4**	∞	∞	∞	3	0.5	∞
Agent **A5**	1	∞	1	∞	2	2

The processes of an organization assigned to execute a mission consisting of information tasks can be conceptualized as follows:

- Task execution (processing) by organizational agents.
- Agent communication – routing task information flow among agents.
- Storing of tasks in the agent's memory.

2.1 Task Execution

Every task is allocated to a single agent capable of processing this task. When a task T_i is processed by an agent A_r, the latter's workload is increased by w_i units. Agents can generally process more than one task at a time, but the dynamic workload (total load of simultaneously processed tasks) of any agent A_r must not exceed agent's workload capacity W_r. A task can begin to be processed by an agent when all the predecessors of a task have been completed and all the information flow from them was communicated to this agent. An example of task processing under an agent's workload capacity constraint is shown in Fig. 3.

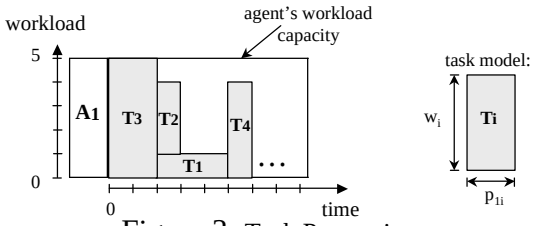

Figure 3: Task Processing

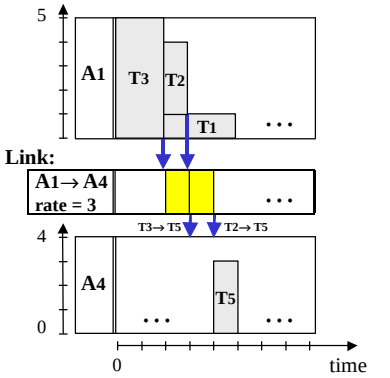

Figure 4: Agent Communication

2.2 Agent Communication

If tasks T_i and T_j are assigned to different agents, information $f_{i,j}$ must be communicated between these agents in the organization (communication is zero if these tasks are assigned to the same agent). The agents can communicate only one message at a time. The time required to communicate $f_{i,j}$ units of information from agent A_r to A_u along the link $e^a_{r,u}$ is equal to $\frac{f_{i,j}}{c_{r,u}}$ if $c_{r,u} \neq 0$. We could generalize the problem formulation by making this time dependent on tasks and on the link between communicating agents.

We assume that only connected agents communicate, and if $c_{r,u} = 0$, then communication between these agents cannot happen. Another approach is to allow such communication to occur through the shortest path between these agents in the network, assuming that the agent network is fully connected. In this case, the most efficient routing of information should be performed dynamically to account for communication link contention. An example of agent communication due to task information flow is shown in Fig. 4.

2.3 Task Storing

The storing of task T_i (in the agent's memory) is required if:

a) Task T_i and its successor task T_j (the task that requires information from T_i) are assigned to the same agent A_r; in this case, the dynamic memory load of agent A_r is increased by m_i units from the finish time of T_i until the start time of T_j;

b) Task T_i is assigned to agent A_r, but its successor task T_j is assigned to agent A_u ($u \neq r$); in this case, the dynamic memory load of agent A_r is increased by m_i units from the finish time of T_i until the time communication of information $f_{i,j}$ is initiated from agent A_r to A_u, and a dynamic memory load of agent A_u is increased by m_i units from the time information $f_{i,j}$ is received from agent A_r until the start time of task T_j.

The dynamic memory load of any agent A_r must not exceed its memory threshold M_r. An example of agent communication and task storage is shown in Fig. 5.

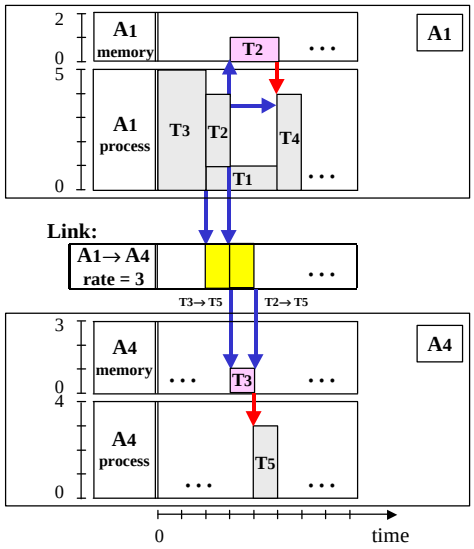

Figure 5: Task Storing

TABLE IV.
Mapping and Scheduling Variables

$x_{r,i}$	assignment variables (=1 iff task T_i is assigned to agent A_r)
$ST(i)$	start time of task T_i
$FT(i)$	finish time of task T_i
$SF(i, j)$	start time of transfer of information between tasks T_i and T_j
$FF(i, j)$	finish time of transfer of information between tasks T_i and T_j
$W(r,t)$	workload of agent A_r at time t
$M(r,t)$	memory load of agent A_r at time t
$E(r,u)$	set of messages passed from agent A_r to agent A_u

The objective is to find a mapping of task structure onto agents' network and the corresponding task schedule that minimize the mission completion time (*makespan*) – the completion time of the last task. This problem can be viewed as consisting of three parts:

1. Allocation of tasks to agents.
2. Sequencing of task execution for each agent.
3. Sequencing of communication (due to task information flow) in agents' network.

3 Mapping Variables

In Table IV, we list the variables used to define the solution to the mapping and scheduling problem.

We define other relevant variables to facilitate the explanations:

- $a(i)$ is an agent to which task T_i is assigned ($x_{a(i),i} = 1$);
- $ST(j,u)$ = (earliest) start time of task T_j on agent A_u;
- $FT(j,u)$ = (earliest) finish time of task T_j on agent A_u;
- $SF(i, j, r, u)$ = start time for transfer of information from task T_i to T_j on the link between agents A_r and A_u;
- $FF(i, j, r, u)$ = finish time for transfer of information from task T_i to T_j on the link between agents A_r and A_u;
- $OUT(i) = \{ j : \exists\ e^t_{i,j} \in E_t \}$ – a set of immediate successors of task T_i;

- $IN(j) = \{ i : \exists\ e_{i,j}^t \in E_t \}$ – a set of immediate predecessors of task T_j;
- $l_b(i,k)$ = length of the longest assignment among all minimum schedules of paths leading from task T_i assigned to agent A_k to the end of the graph; $l_b(i)$ is the smallest among them (sometimes called *bottom level* of a node): $l_b(i) = \min_m l_b(i,m)$;
- $CP(i,k)$ = a critical path starting with task T_i assigned to agent A_k (a sequence of tasks that corresponds to $l_t(i,k)$); $CP(i)$ is the shortest path among $CP(i,k)$:

$$CP(i) = CP(i, m^*),\ m^* = \arg\min_m l_b(i,m) \tag{1}$$

- $CA(i,k)$ = a sequence of agents that are scheduled a critical path $CP(i,k)$; $CA(i)$ is the shortest such schedule ($CA(i) = CA(i,m^*),\ m^* = \arg\min_m l_b(i,m)$);
- $l_t(j,m)$ = length of the longest assignment among all minimum schedules of paths leading from start of the task graph to task T_j assigned to agent A_m not counting the execution time of task T_j; $l_t(j)$ is the smallest among them (sometimes called *top level* of a node): $l_t(i) = \min_m l_t(i,m)$.

Note that $FT(j,u) = ST(j,u) + p_{u,j}$, and

$$FF(i,\ j,r,u) = \begin{cases} SF(i,\ j,r,u) + \dfrac{f_{i,j}}{c_{r,u}}, \text{if}\ \ r \neq u, c_{r,u} \neq 0 \\ \infty, \qquad\qquad \text{if}\ \ r \neq u, c_{r,u} = 0 \\ SF(i,\ j,r,u) = FT(i,r), \ \ \text{otherwise} \end{cases} \tag{2}$$

Static calculation and dynamic update of $l_b(i,k)$ and $l_t(j,m)$ are presented in Appendix A.

4 Mapping Feasibility

To find whether the problem of mapping a task graph onto a network of agents is feasible, we need to test the following conditions:

a) agent-to-task assignment feasibility – for each task there exist an agent that can execute this task:

$$\forall T_i\ \ \exists A_r\ :\ \ p_{ri} \neq \infty \text{ and } W_r \geq w_i \tag{3}$$

b) agent's workload capacity constraints – at any time *t* during mission processing the total workload of an agent must not exceed agent's workload capacity:

$$W_r \geq W(r,t) = \sum_{\substack{i: \\ x_{r,i}=1,\ ST(i) \leq t,\ FT(i) > t}} w_i \tag{4}$$

c) information transfer feasibility – each information message can be communicated in agent's network (only directly connected agents can communicate and the transfer of information between agents according to task information flow occurs without interruption):

$$\forall e^t_{i,j} = (T_i, T_j) \in E_t,\ f_{i,j} \neq 0 \quad \exists P_r, P_u \quad \Rightarrow p_{ri} \neq \infty,\ p_{uj} \neq \infty, \text{ and } \ c_{r,u} \neq 0 \tag{5}$$

d) information scheduling feasibility – information link can transmit only one message at a time:

$$\text{for } E(r,u) = \{f_{i_1,j_1}, ..., f_{i_m,j_m}\}, \forall k = 1,..., m-1 \Rightarrow FF(i_k, j_k) \leq SF(i_{k+1}, j_{k+1}) \tag{6}$$

e) memory allocation feasibility – at any time *t* during mission processing, the memory load of an agent (subsection 2.3) does not exceed agent's memory capacity:

$$M_r \geq M(r,t) = \sum_{\substack{e^t_{i,j} \in E_t: \\ x_{r,i}=1,\ FT(i) \leq t,\ SF(i,j) > t}} m^{out}_i + \sum_{\substack{e^t_{i,j} \in E_t: \\ x_{r,j}=1,\ FF(i,j) \leq t,\ ST(j) > t}} m^{in}_i \tag{7}$$

f) precedence constraints – a task can start execution only after all of its predecessors are finished and all the information is communicated to the corresponding agent:

$\forall e^t_{i,j} \in E_t,\ x_{r,i} = 1,\ x_{u,j} = 1$ we have:

$$FT(i) = ST(i) + p_{ri} \leq SF(i,j) \leq FF(i,j) \tag{8}$$

$$FF(i,j) = \begin{cases} SF(i,j) + \dfrac{f_{i,j}}{c_{r,u}} \leq ST(j), & \text{if } u \neq r \\ SF(i,j) = ST(i), & \text{otherwise} \end{cases} \tag{9}$$

To find the feasibility of scheduling a task graph on an agent architecture, we need to run the critical path algorithm for a heterogeneous network (Appendix A). If there exists *i* such that $l_b(i) = \infty$, then the task graph scheduling is infeasible. The local dynamic feasibility of a schedule is maintained by always allocating a task $_i$ to an agent A_r such that $l_b(\ ,r) \neq$. Coefficients $l_b(i,r)$ are computed off-line as in Appendix A.

5 Solution Approach

The problem defined in section 2 is NP hard in very simple cases ([Garey79], [El-Rewini90b]).

As mentioned in section 2, the scheduling problem can be thought of as consisting of three parts: 1) the task-to-agent assignment; 2) the task execution ordering within an agent; and 3) sequencing of information transfer (information routing) in agents' network. A list scheduling heuristic applied to task and information scheduling solves all three problems at once (*one-stage* method). While low complexity one-stage methods such as the *Critical Path* (CP) algorithm perform very well when communication delays are zero, this is not the case with non-zero communication delays. This is because the edge weights are no longer deterministic; they are functions of task-to-agent assignment (communication would be zero when two tasks are assigned to the same agent, and non-zero when they are assigned to different agents). Consequently, task priorities depend on task-to-agent assignment, and cannot be accurately estimated using a one-stage method. In two-stage methods, once the mapping of tasks to

agents is obtained, the communication pattern becomes deterministic, and a better priority ordering among tasks can be derived.

In general, the existing algorithms for static scheduling of parallel programs represented by a macro-dataflow graph to the set of processors can be classified into three categories:

- Bounded number of processors (BNP) scheduling
- Unbounded number of clusters (UNC) scheduling
- Arbitrary processor network (APN) scheduling

The first class of algorithms is limited to the fully connected processor/agent structures; they do not consider link contention. The second class of algorithms employs hierarchical clustering of tasks; these algorithms do not account for non-homogeneousness of the processing architecture. The last class performs scheduling of tasks onto processors and communication onto network channels; these algorithms consider link contention.

BNP algorithms are modified to account for network topology and can be employed to solve our problem. On the other hand, only linear clustering methods of UNC scheduling can be used for non-homogeneous networks of agents.

The algorithms considered here could be decomposed into two groups: (*i*) list scheduling algorithms, and (*ii*) clustering algorithms (with message routing). The list-scheduling algorithms assign tasks and the corresponding communication one-by-one in a topological order obtained from the task graph. Clustering algorithms assign sets of tasks onto clusters, dynamically changing the original communication graph. List scheduling can be improved by utilizing the insertion heuristic [Gan96].

The following definitions are used in the descriptions of the algorithms:

- $\tilde{p}_i$ = median task processing time (equal to the largest feasible processing time if $\exists\ k$ such that $p_{k,i} = \infty$);
- $\bar{p}_i$ = mean task processing time (equal to the largest feasible processing time if $\exists\ k$ such that $p_{k,i} = \infty$);
- $\bar{c}$ = mean communication link rate: $\bar{c} = \frac{1}{|E_a|} \sum_{e^a_{k,m} \in E_a} c_{k,m}$.

6 List Scheduling Heuristics

List scheduling (or priority scheduling) algorithms define priorities of tasks (either static or dynamic), and allocate "ready" tasks (that have all predecessors assigned to agents) in the decreasing order of priority $pr(i)$ accounting for agent idle times and network topology. The information message routing performance is included in computing a task's earliest start time for each agent.

Earlier research was concerned with specifics of implementation, and it was agreed that the following features improve the list scheduling algorithm's performance:

- List scheduling without a universal time clock is used (tasks are considered according to their assigned predecessors – not just completed ones);

- Insertion is used (search for idle time slots is performed);
- All agents are considered (not only "free" agents at the current time).

TABLE V.

Initialize: $READY = \{i : IN(i) = \varnothing\}$

Step 1: Select task $i = \arg\max_{j \in READY} pr(j) \Rightarrow READY \leftarrow READY \setminus \{i\}$

Step 2: Select agent $r = \arg\min_{u} FT(i,u)$

Step 3: Assign task T_i to agent A_r

Step 4: Update successors' data (including priority info):
$\forall j \in OUT(i)$: $IN(j) \leftarrow IN(j) \setminus \{i\}$ and if $IN(j) = \varnothing \Rightarrow READY \leftarrow READY \cup \{j\}$

Step 5: Repeat steps 1-4 until $READY = \varnothing$.

Priority list scheduling algorithm is outlined in Table V. List scheduling algorithms differ by the method used to calculate the priorities of tasks (Step 1), agent selection (Step 2), and information routing strategy used in Step 3. We outline various existing list-scheduling methods in subsection 6.1, and present our algorithm in subsection 6.2.

6.1 Related Work

6.1.1 Task Priority Selection (Step 1)

Mapping Heuristic (MH). In a slightly different formulation of the problem, the following algorithms describe the basic notion of assigning tasks according to the longest length of the critical path from a task node to the end of the graph: Modified Critical Path (MCP) [Wu88], Mapping Heuristic (MH) [El-Rewini90a,b], [Gan96], and Heterogeneous Earliest Finish Time (HEFT) [Topcuoglu99]. The idea is to select a task with the highest priority defined as a static upward rank $blevel(i)$ that is equal to the length of the longest exit path from task T_i (the computation is based on the mean computation and communication costs). For a heterogeneous system, it can be iteratively calculated as in [Topcuoglu99]:

$$blevel(i) = \overline{p}_i + \max_{j \in OUT(i)} [blevel(j) + \frac{f_{i,j}}{\overline{c}}] \qquad (10)$$

The task is assigned to an agent that minimizes the finish time of the task.

Dominant Sequence (DS). Algorithms such as Mobility Directed (MD) [Wu88], Dominant sequence clustering (DSC) [Gerasoulis90a,b&95] and Critical-path-on-a-processor (CPOP) [Topcuoglu99] assign tasks in decreasing priority defined as the sum of upward and downward rank:

$pr(i) = tlevel(i) + blevel(i)$, where $tlevel(i)$ is the length of the longest path from the entry (top) node to the task node T_i not including the computation cost of T_i. The $tlevel(i)$ is calculated for heterogeneous systems in a similar fashion as $blevel(i)$:

$$tlevel(j) = \max_{i \in IN(j)} [tlevel(i) + \overline{p}_i + \frac{f_{i,j}}{\overline{c}}] \qquad (11)$$

Note that tasks in the same critical path would have the same priorities. CPOP algorithm differs from others by fixing the assignment of tasks on the critical path to the same agent (that completes this path the fastest).

Dynamic Critical Path (DCP). DCP [Kwok94] differs from DSC and MD by restricting certain assignments obtained from a "look-ahead" strategy and load balancing. DCP computes priority characteristics *tlevel* and *blevel* dynamically by "matching" the task graph to the agent system. These values are computed as follows:

$$tlevel(j,m) = \max_{i \in IN(j)} \left[p_{a(i),i} + tlevel(i,a(i)) + I_{m,a(i)} \cdot \frac{f_{i,j}}{c_{a(i),m}} \right] \tag{12}$$

$$blevel(i,m) = p_{a(i),i} + \max_{j \in OUT(i)} \left[blevel(j,a(j)) + I_{m,a(j)} \cdot \frac{f_{i,j}}{c_{a(j),m}} \right] \tag{13}$$

Then task priority is selected as $pr(i) = \min_m \left[tlevel(i,m) + blevel(i,m) \right]$. The calculations and priority updates are straightforward when the agent allocation is known, but this is not the case at the beginning of the algorithm. In [Kwok94], the selection of $a(i)$ is not specified. Therefore, we utilize the following recursive calculations: $tlevel(j,m) = l_t(j,m), blevel(i,m) = l_b(i,m)$ (see Appendix A for details).

Level scheduling (LS). The list scheduling based on levels (or *layers*) of the task graph was first considered in [Adam74] in the case of identical agents. [Shirazi90] used this approach for *heavy node first* (HNF) algorithm. [Iverson95] applied this method for heterogeneous agents. His *Levelized-Min Time* (LMT) is a two-phase procedure. The first phase orders tasks according to their precedence constraints layer-by- layer. The tasks in the same layer are grouped and can be executed in parallel. The second phase is a greedy method that schedules tasks in the same layer in the decreasing order of average computation cost. The agent that provides the fastest finish time of a task is selected.

Dynamic level scheduling (DLS). The Dynamic Level Scheduling algorithm (DLS) proposed in [Sih93] assigns node priorities by using a *dynamic level* (*DL*) of a task that is equal to

$$DL(i,k) = blevel(i) - \max\left[DA(i,k), TF(k) \right] + d(i,k) \tag{14}$$

The variables in equation (14) are defined as follows:

a) $blevel(i)$ is equal to the length of the longest exit path from task T_i; only *median execution times* $\tilde{p}_i$ at each task node among the processing times of this task on all agents are used in computing $blevel(i)$:

$$blevel(i) = \tilde{p}_i + \max_{j \in OUT(i)} blevel(j) \tag{15}$$

(if the median time is infinity, the largest feasible processing time is used);

b) $d(i,k) = \tilde{p}_i - p_{k,i}$ (large positive $d(i,k)$ indicates that agent A_k executes task T_i faster than most agents);

c) $DA(i,k)$ is equal to the time that data from all predecessors of task T_i arrives at agent A_k; and

d) $TF(k)$ is equal to the time that the last task is executed by A_k.

DLS algorithm does not assign priorities to tasks using the critical path. Instead, it performs an exhaustive matching of task nodes to agents. At each scheduling step, the algorithm selects a pair (i,k) of a ready task and an available agent that maximizes $DL(i,k)$. The algorithm uses non-insertion based scheduling, and, therefore, can be modified to better utilize idle times in agent processing and communication network schedules as described in subsection 6.1.3.

6.1.2 Information Routing

For the algorithms described in subsection 6.1.1, the information is routed on a "first come – first serve" basis. The information $f_{i,j}$ can be scheduled to a link between agents A_r and A_u on which *m* messages $f_{i_1,j_1}, f_{i_2,j_2},..., f_{i_m,j_m}$ have been scheduled if there exists some *k* such that

$$SF(i_{k+1}, j_{k+1}, r, u) - \max\{FF(i_k, j_k, r, u), FT(i)\} \geq \frac{f_{i,j}}{c_{r,u}},\ k = 1,...,m \tag{16}$$

where $SF(i_{m+1}, j_{m+1}, r, u) = \infty$. The start time of information $f_{i,j}$ on a link between agents A_r and A_u is given by

$$SF(i, j, r, u) = \max\{FF(i_s, j_s, r, u), FT(i)\} \tag{17}$$

where *s* is the smallest *k* satisfying the inequality (16). Hence, $FF(i, j, r, u) = SF(i, j, r, u) + \frac{f_{i,j}}{c_{r,u}}$.

6.1.3 Agent Selection (Step 2)

For algorithms described in subsection 6.1.1 (except for DLS), an agent is selected to process a task to minimize the finish time of this task. The start time $ST(j,u)$ of scheduling task T_j to agent A_u to minimize the finish time of T_j can be found via

$$ST(j,u) = \max_{t \geq \Delta}\left\{t \mid \forall t \in [t, t + p_{u,j}): W_u \geq W(u,t) + w_j\right\} \tag{18}$$

where $\Delta = \max_{i \in IN(j)} FF(i, j, a(i), u)$. The task is then assigned to an agent A_k so that its finish time is minimized: $k = \arg\min_u FT(j,u)$. Hence, $ST(j) = ST(j,k)$.

6.1.4 Updates

After the assignment has been determined, scheduling and mapping variables are updated. First, information transfer statistics and the corresponding memory loads (commensurate with the new assignment) are updated:

For $\forall r$ and $\forall i$ such that $x_{i,r} = 1$, $e_{i,j}^t \in E_t$, $f_{i,j} \neq 0$ we have:

$$SF(i, j) = SF(i, j, r, k), FF(i, j) = FF(i, j, r, k) \tag{19}$$

$$M(r,t) \leftarrow M(r,t) + m_i \text{ for } \forall t \in [FT(i), SF(i,j)) \tag{20}$$

$$M(k,t) \leftarrow M(k,t) + m_i \text{ and } E(r,k) \leftarrow E(r,k) \cup \{f_{i,j}\} \text{ for } \forall t \in [FF(i,j), ST(j)) \tag{21}$$

Second, assignment variables and workload data corresponding to new assignment are updated:

$$x_{j,k} = 1,\ FT(j) = ST(j) + p_{k,j} \tag{22}$$

$$W(k,t) \leftarrow W(k,t) + w_j \text{ for } \forall t \in [ST(j), FT(j)) \tag{23}$$

6.2 Heterogeneous Dynamic Bottom Level (HDBL) Algorithm

6.2.1 Agent-Task Selection

Algorithms that assign tasks using a static upward rank either completely neglect the mapping of task flow graph onto a heterogeneous agent network (MH, DS) by using the average of processing time and link rate, or use the "best" mapping for priority calculation while disregarding the load balancing issue (DCP). Calculation of priorities in DLS fails to capture the real relation of upward rank to the start time of the task; for instance, calculation of the static upward rank using median values without considering communication and network topology degrades the performance.

In this paper, we propose HDBL algorithm, in which task mapping to every feasible agent is considered. As a result, steps 1 and 2 of list scheduling heuristic are combined to find a task-agent pair. For each *ready* task T_i, an agent $A_{r(i)}$ is selected to minimize a value identifying the schedule delay introduced by assigning it to this agent. A straightforward approach is to select $r(i) = \arg\min_k [ST(i,k) + l_b(i,k)]$. Then a task is selected for scheduling that maximizes the mapped critical path value: $i^* = \arg\max_{\substack{i \in READY \\ l_b(i,r(i)) \neq \infty}} l_b(i, r(i))$, and T_{i^*} is allocated to agent $A_{r(i^*)}$ (hence, $a(i^*) = r(i^*)$). In our algorithm, we use the following rules that utilize the network topology in a more efficient manner:

- Agent-task selection: $$r(i) = \arg\min_k [ST(i,k) + blevel(i,k)] \tag{24}$$
- Task selection: $$i^* = \arg\min_{\substack{i \in READY \\ blevel(i,r(i)) \neq \infty}} [ST(i,r(i)) - blevel(i,r(i))] \tag{25}$$

Here,
$$blevel(i,k) = p_{k,i} + \max_{j \in OUT(i)} \left[\underset{m}{mean} \left(blevel(j,m) + I_{m,k} \cdot \frac{f_{i,j}}{c_{k,m}} \right) \right], \tag{26}$$

where $\underset{m}{mean}(F[m])$ is equal to the mean of $\{F[m] : F[m] \neq \infty\}$, and $I_{m,k} = \begin{cases} 0, \text{ if } k = m \\ 1, \text{ otherwise} \end{cases}$.

The earliest start time $ST(i,k)$ of task T_i on agent A_k is computed dynamically using message routing strategy for single task allocation (as described in subsection 6.2.2) to obtain the time of arrival of all information to agent A_k, equal to $\max_{j \in IN(i)} FF(j,i)$, and the available time slot in agent's processing (equation (18)).

6.2.2 Information Routing

When a task T_j is to be scheduled on agent A_u, all information from its predecessors (that is, tasks from the set $IN(j)$) should be communicated to agent A_u. The set of communication messages from predecessors can be decomposed according to the agents to which the corresponding tasks are allocated. Each such set, denoted as $F_r = \{f_{i,j} \mid i \in IN(j), a(i) = r, f_{i,j} \neq 0\}$, must be mapped to a single communication link $e^a_{r,u} =< A_r, A_u >$ (mapping is feasible iff $c_{r,u} \neq 0$). Instead of scheduling messages from F_r one-by-one (as is done in existing algorithms, subsection 6.1), we utilize a better message routing strategy (see Appendix B for algorithm details). The idea is to use the subset-sum problem and the structure of the link idle times to improve the link "packing", and, as a result, minimize the arrival of all information to agent A_u.

6.3 Example: Agent Selection in HDBL

A selection of the agent that can complete a chosen task at the earliest time (subsection 6.2) disregards the network contention issues and can result in unnecessarily long schedules. The example of Figs. 1,2 and Table III shows the improvement that can be obtained by considering the effect of agent allocation on the critical path length (and, as a result, the total execution time of the task graph).

Table VI shows the values of $blevel(i,k)$. Consider two iterations of the algorithm shown in Fig. 6.

TABLE VI: Bottom Level Values *blevel*(*i*,*k*)

k \ i	T_1	T_2	T_3	T_4	T_5	T_6
A_1	5.5	16.83	17.83	1	–	3
A_2	–	–	–	1	–	1
A_3	4.75	7.33	–	–	–	.5
A_4	–	–	–	3	3.33	–
A_5	9	–	8.33	–	7.33	2

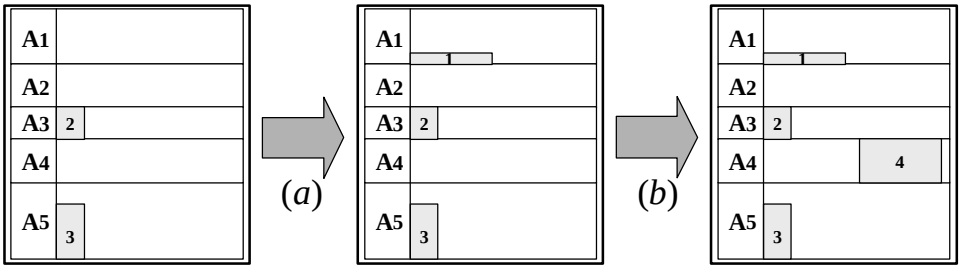

Figure 6: Scheduling Iterations in HDBL

Stage (*a*): ready tasks $READY = [1,5]$; the earliest start times $ST(i,k)$ and finish times $FT(i,k)$ on agents, resulting "agent efficiency" $ST(i,k) + blevel(i,k)$ and "delay" coefficients $ST(i,r(i)) - blevel(i,r(i))$ are listed in Table VII. As a result, task T_1 and agent A_1 are selected. Note that the choice of agent A_5 would minimize the finish time of task T_1, but this would increase the total schedule length of the task graph: the earliest finish times for successor task T_4 would be $[\infty,\infty,-,7,-,-]$.

Stage (*b*): ready tasks $READY = [4,5]$; the earliest start and finish times, resulting "agent efficiency" and "delay" coefficients are shown in Table VIII. As a result, task T_4 and agent A_4 are selected. This agent mapping also minimizes the finish time of selected task. The length of the schedule obtained by HDBL is 6.5. All other algorithms result in a schedule with total length equal to 10 (the outputs of HDBL and MH are shown in Fig. 7).

TABLE VII: Mapping Coefficients (*a*)

Earliest Start Times:

	T_1	T_5
A_1	0	–
A_2	–	–
A_3	1	–
A_4	–	5
A_5	0	2

Earliest Finish Times:

	T_1	T_5
A_1	3	–
A_2	–	–
A_3	2.5	–
A_4	–	5.5
A_5	1	4

Agent Selection: *Agent Efficiency*

	T_1	T_5
A_1		–
A_2	–	–
A_3	5.75	–
A_4	–	
A_5	9	9.33

Task Selection: *Delay Coefficients*

	T_1	T_5
A_1		·
A_2	·	·
A_3	−3.75	·
A_4	·	
A_5	−9	−6.66

TABLE VIII: Mapping Coefficients (*b*)

Earliest Start Times:

	T_4	T_5
A_1	∞	–
A_2	7	–
A_3	–	–
A_4	3.5	5
A_5	–	2

Earliest Finish Times:

	T_4	T_5
A_1	∞	–
A_2	8	–
A_3	–	–
A_4	6.5	5.5
A_5	–	4

Agent Selection: *Agent Efficiency*

	T_4	T_5
A_1	∞	–
A_2	8	–
A_3	–	–
A_4		
A_5	–	9.33

Task Selection: *Delay Coefficients*

	T_4	T_5
A_1	·	·
A_2	6	·
A_3	·	·
A_4		
A_5	·	−6.66

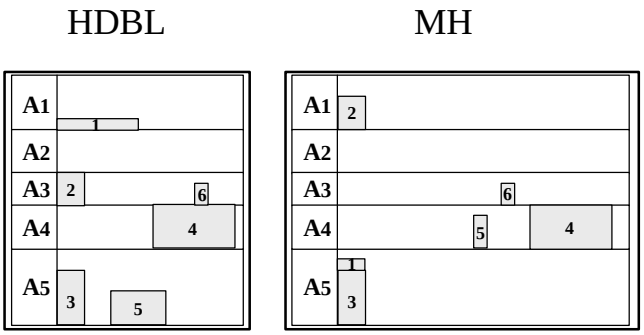

Figure 7: Task Schedule Gantt-Chart

7 Simulation Results

To compare the performance of scheduling algorithms, we utilize the Schedule Length Ratio (SLR) measure for heterogeneous agent systems:

$$SLR = \frac{\max_i FT(i, a(i))}{\sum_{i \in CP} \min_k p_{k,i}}, \tag{27}$$

where *CP* is the critical path in the modified graph without communication constraints with task node computation cost equal to its minimum processing time among agents. SLR is equal to the schedule length (main performance measure of a scheduling algorithm) normalized by its lower bound (equal to the summation of computation costs of nodes on the critical path).

7.1 Communication Versus Computation Cost

We have explored the effect of ratio between communication cost (and corresponding delays) and task computation cost. We consider the simple example of a task graph with 50 task and 6 agent nodes:

- Task processing times are identical for all agents (equal to 1 unit of time).
- Task workload (=1) and task memory load (=0) are identical.
- Agent workload capacity is the same (=1, implying that each agent can do only a single task at a time), task knowledge rate is uniform in [0.3,1].
- Agent architecture is fully connected with link rates uniformly distributed in [1,10] interval.
- Task communication cost is varied from 0 units (no communication between tasks) to 300 units (average communication delay = 60; communication-intensive graph).
- The density of the graph is varied by the ratio of tasks per layer, which is uniform in [0.0,0.2], and the number of predecessors, which is uniform in [0,10]. As a result, the number of layers is between 10 and 50.

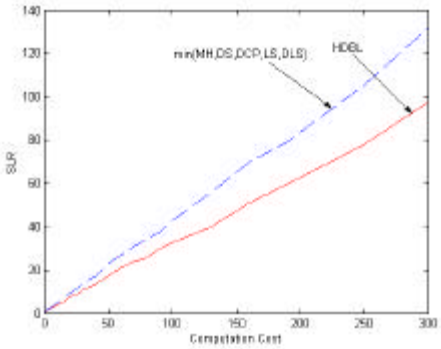

Figure 8: Communication Increase

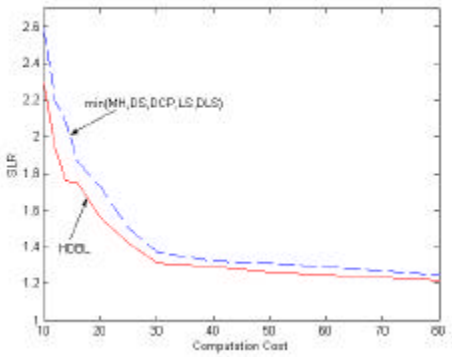

Figure 9: Computation Increase

SLR increases with communication cost (denominator does not include communication in its calculation). We found that the performance of other algorithms (MH, DS, LS, DLS, DCP) is approximately the same (with DLS consistently providing shorter schedules), while HDBL algorithm provides an improvement over these algorithms of up to 28% for communication-intensive graphs. Fig. 8 shows the results of average SLR ratios for 200 Monte-Carlo runs for HDBL algorithm and the best solution of other methods (MH, DS, DCP, LS, DLS). Fig. 9 provides the results for alternative comparison: fixing the communication cost at 50 units and varying the task processing time from 10 to 80 units. HDBL still exhibits superior performance to all other algorithms, although its improvement decreases as the obtained schedule approaches a lower bound (hence, the optimal solution).

7.2 Varied Communication

In subsection 7.2, we showed that our algorithm achieves more than 25% improvement for communication-intensive graphs with equal communication cost for all messages. In Fig. 10, we show results for graphs with varied communication cost. The minimum communication cost is set at 50 units (average delay = 10), and the maximum varies from 50 to 300 units (average delay = 60). We can see that HDBL provides improvement of 20% to 25% over other algorithms.

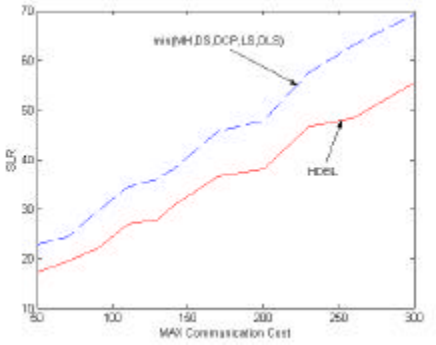

Figure 10: Varied Communication

8 Conclusions and Future Research

In this paper, we have considered the problem of mapping information task graphs onto an organization of heterogeneous agents under agent processing, network topology, workload and memory capacity

constraints. We proposed the HDBL algorithm, which outperforms existing list scheduling heuristics by providing an improvement of over 25% for communication-intensive task graphs. HDBL algorithm is an off-line procedure that is based on matching the critical path in the task graph to the network of heterogeneous agents, providing a better priority evaluation for agent-to-task mapping and scheduling. A novel approach for information routing further improves the performance of the algorithm by efficiently distributing information transfers on communication links in an agent network.

Our current research is focused on exploring task graph structure as means to improve its mapping onto agent's architecture. We consider linear clustering and other graph decomposition methods as a means to extract elementary subgraphs for temporal scheduling onto an agent network. Note that heterogeneous nature of agents in general prevents mapping subgraphs, such as linear clusters (chains of communicating tasks), to a single agent. Even when this is possible, in many situations mapping tasks to multiple agents can reduce the schedule length. In such cases, a trade-off between the optimality of local mapping and communication link contention should be considered.

The information routing in an agent network is one of the most important factors in minimizing communication delays. In our current model, only directly connected agents are allowed to communicate. This can be generalized to allow communication between any indirectly connected nodes by introducing the problem of finding paths for the transfer of information between these agents so as to minimize the aggregated communication cost (or delay) that accounts for link contention. Note that a direct link between two nodes is not necessarily a shortest path between them. We plan to address these issues in our future research efforts.

APPENDIX A: *Critical Path in Heterogeneous Networks*

In the following, we describe a procedure for finding the critical path in a task graph considering the topology of the agent network. Due to the heterogeneous nature of agent's characteristics and network, the generic critical path does not describe the real processes underlying the notion of a critical path as the longest processing sequence in the graph.

To find the critical paths (longest chains) in a task graph, we need to traverse the graph backwards storing the labels at the task nodes associated with the longest assignment among all exit paths leading from this node allocated to any feasible agent. The length of an assignment of a particular exit path is equal to the shortest feasible schedule of the corresponding chain graph on the agent network.

We define S = set of tasks that have all successors labeled. The critical path algorithm is shown in Table IX. We can see that the length of a critical path can be recursively computed:

$$l_b(i,k) = p_{k,i} + \max_{j \in OUT(i)} \left[\min_m \left(l_b(j,m) + I_{m,k} \cdot \frac{f_{i,j}}{c_{k,m}} \right) \right] \tag{28}$$

Note that if $f_{i,j} = 0 \Rightarrow \forall k,m: \frac{f_{i,j}}{c_{k,m}} = 0$.

If task-to-agent assignment is known, then

$$l_b(i,k) = p_{k,i} + \max_{j \in OUT(i)} \left[l_b(j,a(j)) + I_{a(j),k} \cdot \frac{f_{i,j}}{c_{k,a(j)}} \right] \tag{29}$$

Analogously, $l_t(j,m)$ can be computed recursively by traversing the task graph forward:

$$l_t(j,m) = \begin{cases} 0, \text{if } IN(j) = \varnothing \text{ and } p_{m,j} < \infty \\ \infty, \text{otherwise} \end{cases} \tag{30}$$

$$l_t(j,m) = \max_{i \in IN(j)} \left[\min_k \left(p_{k,i} + l_t(i,k) + I_{m,k} \cdot \frac{f_{i,j}}{c_{k,m}} \right) \right] \tag{31}$$

If the assignment of tasks is known, then

$$l_t(j,m) = \max_{i \in IN(j)} \left[p_{a(i),i} + l_t(i,a(i)) + I_{m,a(i)} \cdot \frac{f_{i,j}}{c_{a(i),m}} \right]. \tag{32}$$

More precisely, we should write: $l_t(j,m) = \max_{i \in IN(j)} FF(i, j, a(i), m)$.

$l_t(j,m)$ refers to the earliest possible time that a task T_j can start execution on agent A_m. The latest time that a task T_i can be started on agent A_k so that the start times of tasks in the critical path are not delayed is equal to $\max_j l_t(j) - l_b(i,k)$.

TABLE IX.

Initialize: $CP(i,k) = \begin{cases} [T_i], \text{ if } OUT(i) = \varnothing \\ \varnothing, \text{ otherwise} \end{cases}$ $S = \{i : OUT(i) = \varnothing\}$

$CA(i,k) = [k]$ $\forall j \in S : l_b(j,m) = p_{m,j}$

while $S \neq \varnothing$ **do**
  $S^* = S$, $S = \varnothing$
  for each $j \in S^*$
  for each $i \in IN(j)$
    for each $k = 1,\ldots,K$
      if $f_{i,j} = 0$, **then** $m^* = \arg\min_m [l_b(j,m)] \Rightarrow \Delta = l_b(j,m^*)$
      else

$$m^* = \begin{cases} k, \text{ if } l_b(j,k) \le \min_{m: m \neq k} \left[l_b(j,m) + \frac{f_{i,j}}{c_{k,m}} \right] \Rightarrow \Delta = l_b(j,k) \\ \arg\min_{m: m \neq k} \left[l_b(j,m) + \frac{f_{i,j}}{c_{k,m}} \right], \text{ otherwise } \Rightarrow \Delta = l_b(j,m^*) + \frac{f_{i,j}}{c_{k,m^*}} \end{cases}$$

      end if
      if $l_b(i,k) > p_{k,i} + \Delta$
      then $l_b(i,k) = p_{k,i} + \Delta$ $CP(i,k) = [i, CP(j,m^*)]$
      $CA(i,k) = [k, CA(j,m^*)]$
      end if
      $OUT(i) = OUT(i) \backslash \{j\}$
      if $OUT(i) = \varnothing$, **then** $S = S \cup \{i\}$ **end if**
  end for
  end for
end while

APPENDIX B: ***Scheduling Multiple Information Messages on a Single Communication Link***

For each agent A_r ($r \neq u$), we identify the set S_r which consists of predecessors of task T_j that are assigned to A_r: $S_r = \{i \mid i \in IN(j), a(i) = r\}$. Then, a set of information messages from these tasks $F_r = \{f_{i,j} \mid i \in S_r, f_{i,j} \neq 0\}$ must be communicated to agent A_u. If $F_r \neq \varnothing$, messages from set F_r must be scheduled to the link $e^a_{r,u} =< A_r, A_u >$. Note that this scheduling is feasible iff $c_{r,u} \neq 0$.

Let us assume that the message set $E(r,u) = \{f_{i_1,j_1}, f_{i_2,j_2}, ..., f_{i_m,j_m}\}$ have been scheduled to communication link $e^a_{r,u} =< A_r, A_u >$. We define

$$t_k = FF(i_k, j_k), \Delta_k = SF(i_{k+1}, j_{k+1}) - FF(i_k, j_k), \text{ for } k = 1,...,m-1, \tag{33}$$

where $t_0 = 0, \Delta_0 = SF(i_1, j_1)$. If $\Delta_k \neq 0$, then the interval $[t_k, t_k + \Delta_k]$ (called *idle interval*) can be used to allocate a set $F_{r,k}$ of messages that can be executed during $[t_k, t_k + \Delta_k]$. That is, $F_{r,k} = \{f_{i,j} : i \in S_{r,k}, f_{i,j} \neq 0\}$, where

$$S_{r,k} = \left\{ i : i \in S_r, \max\{t_k, FT(i)\} + \frac{f_{i,j}}{c_{r,u}} \leq t_k + \Delta_k \right\} \tag{34}$$

Suppose we wish to schedule a set *F* of messages with lengths w_n and release times r_n during the time interval $[a,b]$. Since the objective of message scheduling is to reduce the time of arrival of the last message to agent A_u, the problem of scheduling to a single idle time interval $[a,b]$ is equivalent to finding an allocation with maximum total scheduling time, that is $\max \sum_{n \in F} x_n w_n$, where $x_n = 1$ iff a message *n* is assigned to interval $[a,b]$ (=0 otherwise). The scheduling must take into account the time constraints of the problem (message release times r_n for each message, non-overlapping schedule of tasks, and interval length). When $\forall n \in F : r_n \leq a$, the problem becomes a *subset-sum* problem for a knapsack with capacity $C = b - a$:

$$\max \sum_{n \in F} x_n w_n$$
$$s.t. \begin{cases} \sum_{n \in F} x_n w_n \leq C \\ x_n \in \{0,1\} \end{cases} \tag{35}$$

Although this problem is NP-hard, it can be solved efficiently. For example, fully polynomial approximation schemes have been developed [Martello90].

When $\exists n \in F : r_n > a$, we can solve the problem by applying a sequence of subset-sum problems working "backwards" in the idle interval $[a,b]$. The corresponding algorithm called Bounded Interval Message Mapping (BIMM) is described in Table X. BIMM algorithm uses the consecutive application of Subset-Sum problem for time intervals determined from the release times of messages.

For our problem, when scheduling set $F = F_{r,k}$ in an interval $[a,b] = [t_k, t_k + \Delta_k]$, we have:

$$r_n = ST(i_n), w_n = \frac{f_{i_n,j}}{c_{r,u}}, \text{ and } C = \Delta_k . \quad (36)$$

We propose the algorithm (Table XI) that schedules information messages to a communication link utilizing the available idle time intervals in link assignment. The algorithm searches for time intervals and assigns messages to them according to BIMM.

TABLE X.

Input: Message set F, message release times r_i, message lengths w_i, time interval $[a,b]$
Output: Assigned messages F^*

Initialize: $F'=\varnothing$, $F^*=\varnothing$, $t_2=b$
Step 1: Find t_1 = maximal start time of messages from F.
Step 2: Select tasks from F with start time equal to t_1, remove them from F and add to F'.
Step 3: Find set F'' of messages in F' that can be executed during time interval $[t_1, t_2]$. That is:

$$F'' = \{ i : i \in F', \max[t_1, r_i] + w_i \le t_2 \}$$

Step 4: Solve Subset-Sum problem for items from F'' and knapsack with capacity $C = t_2 - t_1$. Obtain set of assigned items $\hat{F}$ and total assigned time

$$W = \sum_{i \in \hat{S}} w_i$$

Step 5: Remove items of $\hat{S}$ from F' and add them to F^*. Update $t_2 = t_1 + C - W$.
Step 6: Repeat steps 1-5 until $F = \varnothing$

TABLE XI.

Input: Set of messages $F_{r,u}$ to be assigned to a link with messages $E(r,u) = \{ f_{i_1,j_1}, f_{i_2,j_2}, ..., f_{i_m,j_m} \}$

Initialize: $FT(0) = 0, F = \varnothing, r_i = ST(i), w_i = \frac{f_{i,j}}{c_{r,u}}$

for k=0,…,m-1 **do**
  $a = FT(i_k) + w_{i_k}, b = FT(i_{k+1})$ and define a set of messages $F_{r,k}$
  if $a<b$, then $F = F \cup F_{r,k}$ and solve *BIMM*, obtaining set S^* of assigned tasks.
  Update: $F = F \setminus S^*$
end for

If $F \neq \varnothing$, then assign remaining items in the increasing order of message release times:
$a = FT(i_m) + w_{i_m}$
while $F \neq \varnothing$ **do**
  Select a message i from F with smallest release time $ST(i)$.
  Update $SF(i,j) = \max(a, FT(i)), FF(i,j) = SF(i,j) + w_i$
  $a = FF(i,j)$
end while

References

[Adam74] T. Adam, K.M. Candy, and J.R. Dickson. "A Comparison of list schedules for parallel processing Systems." *CACM*, 17:12 (1974), pp. 685-690.

[Burton98] R.M. Burton, and B. Obel, *Strategic Organizational Diagnosis and Design: Developing Theory for Application* (2nd Ed.). Boston, MA: Kluwer Academic Publishers, 1998.

[El-Rewini90a] H. El-Rewini, T. Lewis. "Parallex: A Tool for Parallel Program Scheduling." *IEEE Journal of Parallel and Distributed Computing*, 1990.

[El-Rewini90b] H. El-Rewini, T. Lewis. "Scheduling Parallel Programs onto Arbitrary Target Machines", *IEEE Journal of Parallel and Distributed Computing*, Vol. 9, no. 2, June 1990, pp. 138-153.

[Entin99] E.E. Entin, "Optimized Command and Control Architectures for Improved Process and Performance", *Proceedings of the 1999 Command & Control Research & Technology Symposium*, NWC, Newport, RI, June 1999, pp. 116-122.

[Gan96] Boon Ping Gan; Shell Ying Huang. "The Modified Mapping Heuristic Algorithm", *IEEE Second International Conference on Algorithms & Architectures for Parallel Processing*, 1996, pp. 456 –463.

[Garey79] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[Gerasoulis90a] A.Gerasoulis, S.Venugopal. "Linear Clustering of Linear Algebra Task Graphs for Local Memory Systems." *Report*, 1990.

[Gerasoulis90b] A. Gerasoulis, Tao Yang. "Dominant Sequence Clustering Heuristic Algorithm for Multiprocessors." *Report*, 1990.

[Gerasoulis94] A. Gerasoulis, Tao Yang. “DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors.” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 9, September 1994, pp. 951-967.

[Graham66] R.L. Graham. “Bounds for Certain Multiprocessing Anomalies.” *Bell System Tech. J.*, 45 (1966), pp. 1563-1581.

[Hocevar99] S.P. Hocevar, W.G. Kemple, D. Kleinman. And G. Porter, “Assessments of Simulated Performance of Alternative Architectures for Command and Control: The Role of Coordination”, *Proceedings of the 1999 Command & Control Research & Technology Symposium*, NWC, Newport, RI, June 1999, pp. 123-143.

[Iverson95] M. Iverson, F. Ozguner, G. Follen. “Parallelizing Existing Applications in a Distributed Heterogeneous Environments”, *Proc. Of Heterogeneous Computing Workshop*, pp. 93-100, 1995.

[Kapasouris91] P. Kapasouris, D. Serfaty, J.C. Deckert, J.G. Wohl, K.R. Pattipati. “Resource Allocation and Performance Evaluation in Large Human-Machine Organizations”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 3, May/June 1991, pp. 521-531.

[Kim88] S.J.Kim, J.C.Browne. “A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures.” *International Conference on Parallel Processing*, 3(1988), pp. 1-8.

[Kwok94] Yu-Kwong Kwok, Ishfaq Ahmad. “A Static Scheduling Algorithm Using Dynamic Critical Path for Assigning Parallel Algorithms onto Multiprocessors.” *Proceedings of the 1994 International Conference on Parallel Processing*, 1994.

[Levchuk2000a] Levchuk, G.M., Y. N. Levchuk, Jie Luo, Krishna R. Pattipati, and David L. Kleinman, “Normative design of organizations - part I: mission planning”, To be published in *IEEE Transactions on Systems, Man, and Cybernetics*, 2002.

[Levchuk2000b] Levchuk, G.M., Y. N. Levchuk, Jie Luo, Krishna R. Pattipati, and David L. Kleinman, “Normative design of organizations - part II: organizational structure”, To be published in *IEEE Transactions on Systems, Man, and Cybernetics*, 2002.

[Levchuk2002] Y.N. Levchuk G.M. Levchuk, K.R. Pattipati. “A Systematic Approach to Optimize Organizations Operating in Uncertain Environments: Design Methodology and Applications”, To appear in *Proceedings of the 7-th International Command & Control Research & Technology Symposium, 2002*, Québec City, QC, Canada, Sept, 2002.

[Martello90] S. Martello, and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY: John Wiley & Sons, 1990.

[McCreary90] C. McCreary, H. Gill. “Efficient Exploration of Concurrency Using Graph Decomposition.” *Proceedings 1990 IEEE International Conference on Parallel Processing*, pp. II-199, Institute of Electrical and Electronic Engineers, 1990.

[Meirina2002] Candra Meirina, Yuri N. Levchuk, Krishna R. Pattipati. “Goal Management in Organizations: A Markov Decision Process (MPD) Approach”, To appear in *Proceedings of the 2002 Command & Control Research & Technology Symposium*, NPS, Monterrey, CA, June, 2002.

[Sarkar89] V. Sarkar. “Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors.” *The MIT Press*, 1989.

[Shirazi90] B. Shirazi *et al*., “Analysis and evaluation of Heuristic methods for static task scheduling”, *J. of parallel and distributed computing*, 10, pp. 222-232, 1990.

[Sih93] G.C. Sih, and E.A. Lee. “A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 2, Feb. 1993, pp.175-187.

[Topcuoglu99] H. Topcuoglu, S. Hariri, Min-You Wu. “Task scheduling algorithms for heterogeneous processors”, HCW '99 Proceedings. Eighth , pp. 3-14, 1999.

[Tu2002] Haying Tu, Yuri N. Levchuk, Krishna R. Pattipati. “Robust Strategy to Induce Desired Effects”, To appear in *Proceedings of the 2002 Command & Control Research & Technology Symposium*, NPS, Monterrey, CA, June, 2002.

[Wu88] Min-You Wu, D. Gajski. “A Programming Aid for Hypercube Architectures.” *The Journal of Supercomputing*, 2(1988), pp. 349-372.