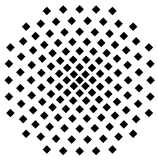
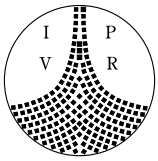


Universität Stuttgart
Fakultät Informatik



Problem Formulations, Models and Algorithms for Mapping Distributed Multimedia Applications to Distributed Computer Systems

Alexander Hagin, Gabriel Dermler, Kurt Rothermel

CR-Klassifikation: C.2.4, C.4, G.1.6, G.2.2, I.6

Problem Formulations, Models and Algorithms for Mapping Distributed Multimedia Applications to Distributed Computer Systems

Fakultätsbericht 3/1996
Technical Report
Februar 1996

Fakultät Informatik
Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20 - 22
D-70565 Stuttgart

G.Dermler, K.Rothermel

University of Stuttgart/IPVR
Breitwiesenstr. 20 - 22
D-70565 Stuttgart
Germany

A.A.Hagin

Technical University of St.-Petersburg
Polytechnicheskaja str. 29
195251 Saint-Petersburg
Russia

Abstract

In the report, the problem of optimal allocation of Distributed Multimedia Applications (DMA) into Distributed Computer Systems (DCS) is examined. We are given precedence graphs representing topologies of the DMA and data streams between components of DMA. Nodes and arcs of the graphs are weighted by the computational and communication resources needed to meet quality of service requirements of DMA. A special-purpose precedence graph model is proposed to present the structure of DCS including computers, virtual channel connections and communication resources of DCS over which the channels are routed. Nodes and links of the graph are weighted by the computational resources and capacities of communication resources available to mapped DMA.

An approach, based on the solving two kinds of the mapping problem, is proposed. The first one is formulated as a nonlinear integer programming problem with cost function under constraints on DCS resources available to mapped DMA. If the first one has not an acceptable solution, then other problem, formulated as minimax nonlinear integer one to find the DMAlocation into the DCS with minimum DCS resource gap, is solved. To solve both problems, effective algorithms based on the branch-bound method are proposed. Computational efficiency of the algorithms is examined and illustrated by numerical examples.

Contents

1 Introduction

Recent technological developments in high speed networks and multimedia workstations have given rise to entirely new classes of distributed applications such as distance learning, desktop videoconferencing, remote multimedia database access and so on. Multimedia systems combine a variety of information sources, such as voice, graphics, animation, images, audio, and full-motion video, into a wide range of applications. Research and development efforts in multimedia computing can be divided into two groups. One group centers on the stand-alone multimedia workstation and associated software systems and tools. The other combines multimedia computing with distributed systems. Potential new applications are based on distributed multimedia systems [1].

1.1 Framework for DMA implementation

A distributed multimedia application (DMA) needs a convenient framework for its implementation in a distributed computer system (DCS). One successful approach is the CINEMA (Configurable INtEgrated Multimedia Architecture) development platform providing abstractions for dynamic configuration of DMA and for arbitrary complex stream synchronization requirements [2].

Distributed multimedia applications (DMA) are employed to generate, process and consume (e.g. present) continuous (e.g. audio, video) data streams across distributed locations into computer network.

CINEMA allows a client to compose an application out of components and links as one or some data flow graphs. A component is an individually schedulable unit (e.g. by mapping to a thread). In the graph the nodes represent components, while links (arcs) represent the data streams between components. The streams may have different rates. A component accesses the data units of streams via ports.

Further, the initial configuration may be dynamically changed during run time as need. Dynamic configuration of DMA allows to take into account changes of available resources in distributed computer system (DCS) and changes of the required quality of service the user asks for at run time. Moreover DMA are often highly dynamic in the sense that users may join and leave the application during run time that causes necessity of the reconfiguration of DMA without session interruption. On the other hand, DCS is a dynamic system with set of different changeable DMA executed, changeable available computational and communication resources for allocation and execution of a new DMA.

Components encapsulate processing of multimedia data, e.g. for generating (source components), presenting (sink components) or manipulating (filters and mixers) data. To provide a uniform data access point for the components, ports are used that deliver data units to the component (input port) or take the data units from the component (output port). A component designer

has to associate with each component port the streamtype to be used making all related information available at the port. A client constructs an application by specifying a topology of components interconnected via links. A link provides an abstraction from underlying communication mechanisms which may be used to perform the transport of data units.

A data flow graph may be arbitrary distributed over several nodes of a distributed computer system. Components are configuration independent, which means that their internal logic is independent of the configuration they are used in.Thus, from the client's point of view, there is no conceptual difference whether two adjacent components run either on the same computer-node or on different computer-nodes connected by a remote link.

Before using an application, a client has to specify desired QoS (Quality of Service) to CINEMA with respect to output data generated by sink components (e.g. presented video frame size and rate). Client requests are related to sink component ports, assuming that the client has knowledge about sensed output QoS and corresponding media specific QoS required as input to sink components.

CINEMA offers the concept of a session allowing the client to specify the application to be instantiated and the QoS expected with respect to output generated by sink components. A session is the unit of resource reservation and QoS negotiation. By creating a session, a client causes the CINEMA system to reserve the resources that are needed to guarantee the specified quality of service requirements. After a session has been established, the transmission and processing of multimedia data may be started.

A session encompasses parts of the data flow graph which is defined by a client. Its actual extension is defined by specifying a set of source and sink components. Intermediate components and interconnecting data paths are determined from the data flow graph by the CINEMA system.

In CINEMA, a new protocol type termed negotiation and resource reservation protocol NRP is proposed. NRP is carried out by a corresponding protocol engine (PE) in three phases during which (in phase 1) the so-called application flowspec (AFS) available at the sink port is propagated towards the source components for resource reservation, (in phase 2) AFS from source ports propagated back to the sink to prepare resource relaxation, and (in phase 3) resource reservation are relaxed while propagating the AFS towards source again.

Thus, general framework supported by CINEMA session is following.

1. Constructing and specifying by client an DMA (data flow graphs) and QoS with respect to consumed data.
2. Checking the logical correctness (agreement) between application specification and QoS requirements taking into account limited functional capabilities of components. (For instance, all components connected to an output port of the same mixer have to get a picture with same characteristics of size, rate and so on. Two interconnected component ports have to be associated with the same streamtype. A limited resource availability of any component or intermediate link affects QoS to be provided by all other components or links and so on).

3. Mapping media specific parameter values (e.g. video frame size and rate and so on) to computational and communication resource requirements that agree QoS requests of client.
4. Mapping weighted data flow (application) graphs of DMA to DCS taking into account required computational and communication resources of DMA and, on the other hand, available computational and communication resources of DCS, and cost expenses of DCS resources that will be reserved.
5. Resource reservation in DCS by NRP protocol.

Return(s) from each of these 5 stage to previous one may be needed depending on result obtained on the stage.

The paper deals with the problem 4 of mapping a DMA to a DCS.

1.2 Mapping problem

A DMA can be represented by one or some application precedence graphs [2]. In an application graph (or simply DMA graph), nodes represent components that are interconnected by arcs representing data streams between components. Each component is associated with at least one device that produces (a source component) or processes (an intermediate component - filter or mixer) or consumes (a sink component) data streams. Each node of the application graph is weighted by computational requirements of the corresponding component and each arc is weighted by the channel capacity needed for remote communication between adjacent components. Media streams can originate at multiple sources, traverse a number of intermediate components and end at multiple sinks.

The DCS considered is heterogeneous and consists of computers (workstations) communicating with each other through point-to-point physical link, local area network(s) (LAN) and/or wide area network(s) (WAN). For every DMA, the part of the DCS that will always be considered consists of computers each of which can be used for executing the functions of one or more components. The communication structure of a DCS is described by a system oriented graph (or simply DCS graph). In the system graph, nodes represent computers that are interconnected by arcs representing virtual channels of the DCS. The DCS graph includes weights per nodes (values of available computational resources and cost functions for such resources) and per arcs (values of available channel capacity and capacity cost functions).

The mapping problem has been thoroughly investigated for parallel systems [3]; if usually arises when the number of computational modules required by the application exceeds that of processors available, or when the interconnection structure of the application computational modules differs from that of the parallel machine [4,5]. In [6], the mapping problem is defined as the assignment of modules to processors and maximizing the number of pairs of communicating modules that are allocated to pairs of directly connected processors. It is shown that the mapping problem is equivalent to the graph isomorphism problem, or to the bandwidth reduc-

tion problem. In either case, an exact and simple algorithm for the mapping problem is extremely difficult to develop. Research in this area has concentrated on efficient heuristics which give good solutions in most cases [3 -6].

The problem of mapping a DMA to a DCS differs from the one mentioned above in the following: a) a DCS is heterogeneous, b) the relevant part of the DCS network with computational and communication resources available may not be fully interconnected, in particular because of the channel utilization by other currently performed applications, c) communication is performed via data passing between the computers through a point-to point channel, local and/or global networks by using access to channel resources belonging to an adjacent computer pair or shared by some computer pairs connected to the same transmission media, d) the purpose of the DMA allocation scheduling in a DCS is to minimize the cost of DCS resources used for the DMA allocation, provided the DCS provides resources for the DMA needed to satisfy the quality of user service requirements.

Mapping a DMA to a DCS can be used to provide two kinds of service: an advance reservation service and an immediate one [10]. In the first case the provider has to do some planning for future allocations using two time parameters of the client request - the starting time and the duration of DMA. The mapping server takes into account all accepted requests of advance service, time parameters of which are overlapped by the ones of the arrived request. The mechanism of interval tables [10] can be used to determine resources of DCS available for the DMA request. Such advance reservation service mode allows to assume that the utilization state of the DCS will be not changed in the future interval of the DMA execution and therefore mapping server deals with the static state of DCS resource utilization.

The immediate reservation service has to do in the real-time mode. The mapping is made during the session establishment phase and is adjustable depending on the current utilization states of the DCS computers and channels.

In the paper a mathematical formulation of the mapping problem is considered. Algorithms for its solution is proposed. The algorithms can be used for an advance reservation service and, after some modifications, an immediate ones.

In section 2, the graph models for DMA and DCS resource specification as input data for mapping problem are presented. In section 3, an approach to the mapping problem solution is proposed. In section 4, problem formulation of the mapping a DMA to a DCS is presented. In section 5, pre-starting procedure for locating attached (source and sink) components of DMA onto DCS are examined. In section 6, some algorithms, based on the branch-bound method, for obtaining the optimal location of DMA onto DCS are considered. In section 7, a problem formulation and an algorithm of reduction procedure, based on the branch-bound method, for obtaining a feasible location of DMA onto DCS with minimum resource requirement relaxation are presented. Reduction procedure is performed if available resources of DCS can not meet resource requirements of DMA. In section 8, numerical examples are used to demonstrate the algorithms and to evaluate its computing efficiency. In section 9, conclusions about this approach and its use for DCSs are discussed.

2 Graph models for DMA and DCS specification

2.1 Parametrization of DMA graph

Let us consider a DMA graph and determine the values of node and arc weights.

Every node is weighted by computational requirements of the corresponding component of the DMA. Let λ_i denote the arrival rate (messages per second) of input data streams to component i in the DMA graph. To process every message, component i needs V_i processor operations. Let C_i denote the component computational requirement (operations per second). To exclude computer saturation, it is necessary that $C_i > \lambda_i V_i$. In this case we get utilization factor $\rho = \lambda_i V_i / C_i$ for computer with virtual processor speed C_i. To calculate the component computational requirement, we must construct a model for computer performance evaluation and take into account the quality of service parameters of the DMA. For instance, if we use the simple classic M/M/1 queueing system model and acceptable mean delay T_i of a message in the component i, then the computational requirement of component i is $C_i = V_i(1 + \lambda_i V_i)/T_i$. On the other hand virtual processor speed C_i is a function of real processor speed and values of main (high-speed) and disk memories used for storage and distribution of program modules and data.

We could use more detailed and exact models for computer performance evaluation, for example [7]. For the present, it is important that we can weight every node i of the DMA graph by computational requirement C_i of the corresponding component.

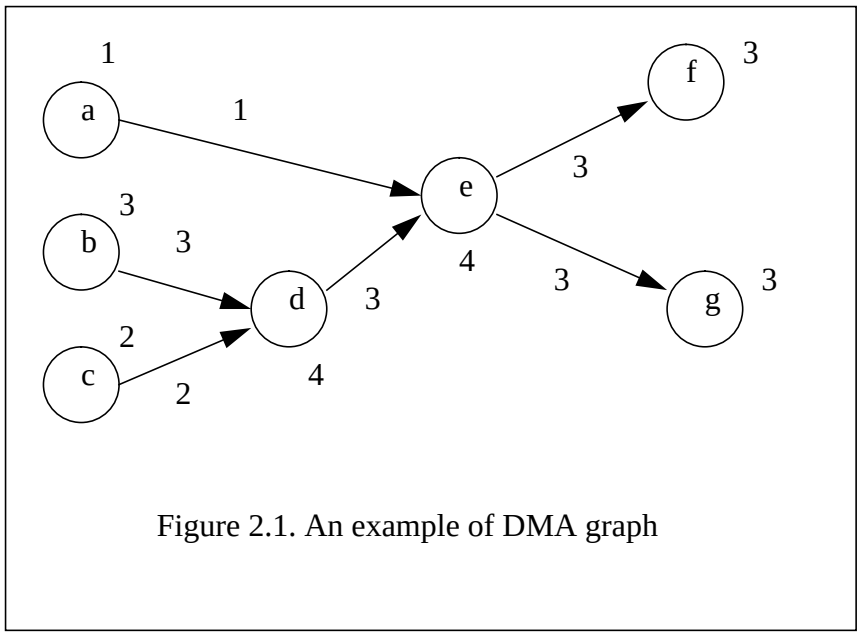


Figure 2.1. An example of DMA graph

Every arc (i, j) in the DMA graph is weighted by capacity requirement C_{ij} (bits per second) that can be calculated similarly to component computational requirement C_i.

An example of a DMA graph is presented in Figure 2.1. The topology of DMA is composed of three source-components a, b, c connected to two mixing components d and e last of which provides data streams to two sink components f and g. Numbers at nodes and arcs denote computational and communication resource requirements respectively.

2.2 Parametrization of DCS graph

Let us now consider a system graph of the DCS. Every node n is weighted by computational resource R_n available (operations per second) and cost functions $f_n(x)$ of used computational resource x.

If B_n is the total computational resource of computer n in the DCS and b_n is its computational resource already used by all applications processed in the DCS, then the computational resource available of computer n is $R_n = B_n - b_n$

The system graph representation of the DCS shows possible *virtual channel connections* (VC) between the computers of the DCS. A VC[1] is a direct oriented logical connection between two computers (endsystems) with some assigned capacity. A VC is routed over one or more communication resources of DCS (physical links, networks) to achieve sender-computer to receiver-computer connectivity. By *a communication resource of DCS* we shall basically mean maximum aggregated physical communication unit of DCS that is characterized by bandwidth (capacity), transmission direction and is used for connecting at least one computer pair of DCS.

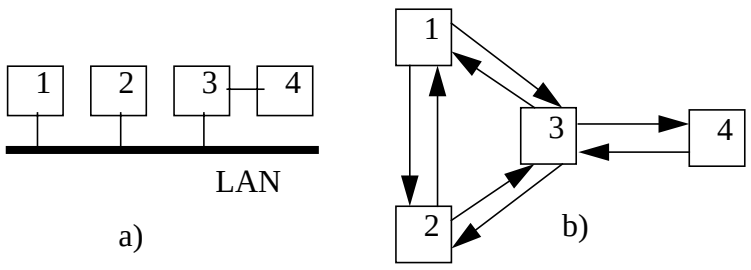


Figure 2.2,a) DCS example,
b) Representation of computer connections by VCs in the system graph

[1] A VC concept is similarly to virtual path concept in an ATM network [8] excluding that here it is used to the application level.

For DMA mapping problem we consider only such VCs each of that includes only two computers of DCS, first one as a sender and second one as a receiver, and is not routed over other computers, as intermediate ones. For example Figure 2.2 presents the DCS structure and system graph using acceptable VCs. A VC begins from the output interface of a computer-sender and ends at the input interface of a computer-receiver, i.e. a VC includes the interfaces of the end-systems.

The available capacity of a VC is equal to minimum available capacities of all DCS communication resources over which the VC is routed. Let A_s be the capacity of DCS communication resource s; ρ_{nm} be the set of DCS communication resources used by VC (n, m) from computer n to computer m of DCS. Then the available capacity of the VC (n, m) is

$$R_{nm} = min\{A_s, s \in \rho_{nm}\} \tag{2.1}$$

Let us consider following basic kinds of computer connections:

by an individual point-to-point (physical) channel,

through a wide area network (WAN),

through a local area network (LAN).

In the first case (see Fig. 2.3) the capacity of the point-to-point physical channel belongs to the pair of computers connected by this channel. However the capacity of output and input interfaces of each computer are shared by all VCs routed over the output and input interfaces of the computer.

Therefore, if

A_{nm} is the available capacity of the channel in direction from computer n to computer m,

I_n^{out}, I_n^{in} are available capacities of the output and input interfaces of computer n,

then the capacity of VC (n, m) available for the mapped DMA is

$$R_{nm} = min\{I_n^{out}, I_m^{in}, A_{nm}\} \tag{2.2}$$

In the second case (computer connection through a WAN, see Figurte 2.4) the system graph representation is similar to one shown in Figure 2.3,b. Formula (2.2) can be used also but here A_{nm} denotes the throughput of the WAN from network access point of computer n to one of computer m. To calculate the throughput of gigabit networks, the relation between latency and bandwidth must be taken into account [9].

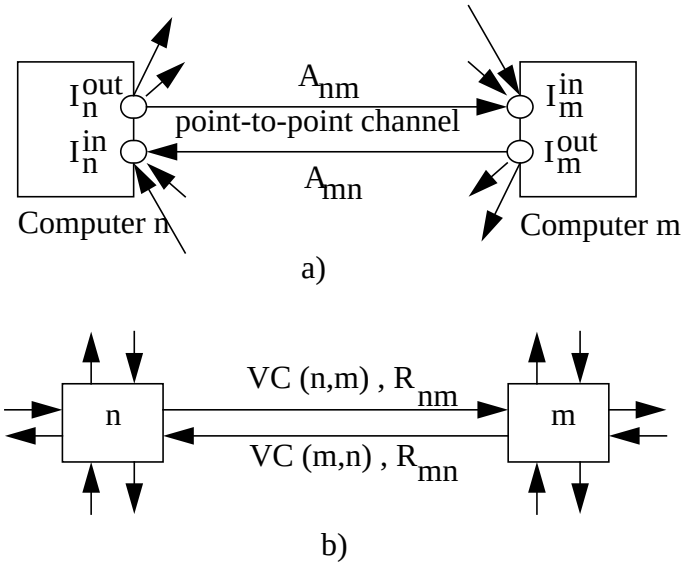


Figure 2.3, a) Computer connection by point-to point physical channel, b) Representation of computer connection by VCs in the system graph

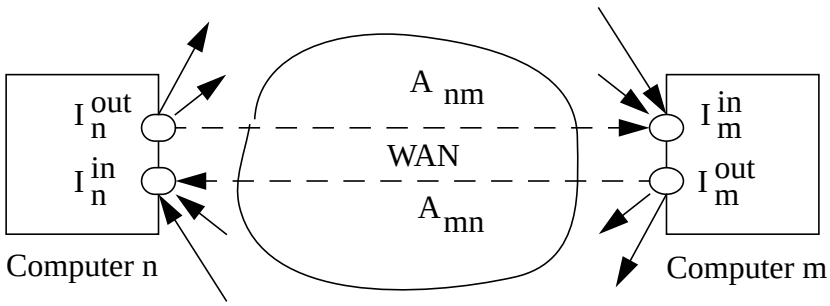


Figure 2.4. Connections in the DCS through wide area network

In the third case the capacity of the LAN transmission (see Figure 2.5) line does not belong to any pair of computers but is distributed among all computers of the LAN. The LAN provides a virtual channel between any pair of computers. Therefore, a system graph for the LAN is a logical graph representing all possible VCs between computers connected to the LAN (see Figure

2.6). Available capacity A of the LAN transmission line is distributed among all data-exchanging computer pairs. Therefore

$$0 \le \sum_{(n,m)} R_{nm} \le A,$$

$$R_{nm} = min\{I_n^{out}, I_m^{in}, A\} \qquad (2.3)$$

Capacity A is available for every possible VC in the system graph. It means that if the capacity available of the LAN, for example, is decreased by a, then the capacity available of every possible virtual channel with $R_{nm} = A$ decreases by the same value a, too.

Note that interfaces of computers are distributed among all channels routed over the interfaces. Therefore for every interface of a computer n routed by some VCs the following capacity constraints have to be satisfied $\sum_m R_{nm} \le I_n^{out}$, $\sum_m R_{mn} \le I_n^{in}$.

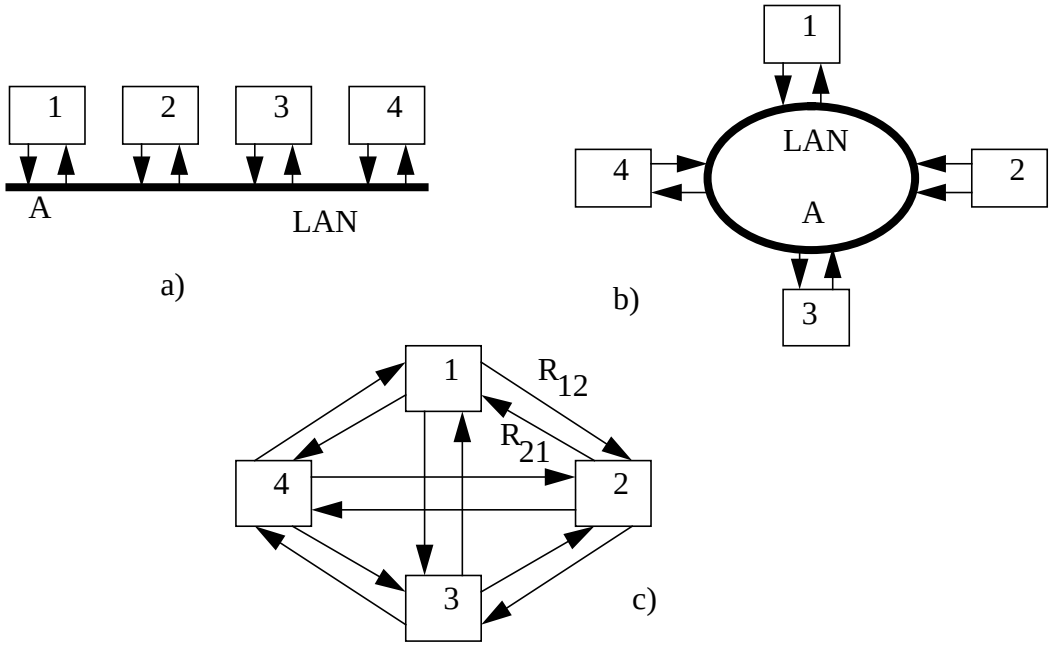


Figure 2.5,a,b) Connections in the DCS through local area network,
c) Representation of computer connections by VCs in the system graph

Figure 2.6,a illustrates a more complicated case of the DCS structure that can be represented by the logical system graph in Figure 2.6,b. Here the capacities available for every computer pair connection are the following: $R_{12} = min\{I_1^{out}, I_2^{in}, A^{LAN1}\}$, $R_{13} = min\{I_1^{out}, I_3^{in}, A^{LAN1}, A^{WAN}, A^{LAN2}\}$, etc.

Further every arc (n, m) of the system graph represents corresponding VC (n, m) of DCS and is weighted by available capacity R_{nm} (bits per second) and communication cost function $g_{nm}(x)$ of the VC (n, m).

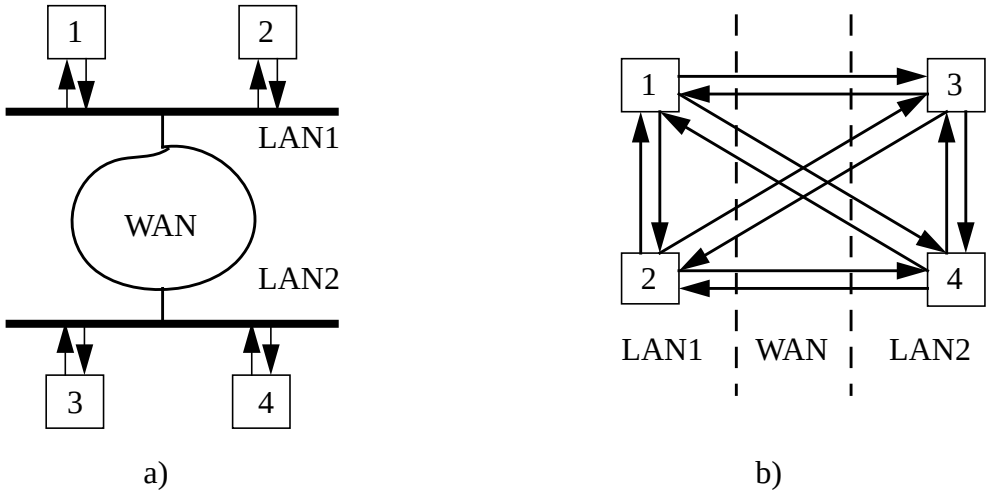


Figure 2.6, a) Communications in the DCS through some differnet networcs.
b) Representation of computer connections by VCs in the system graph

Let us consider an example of system graph construction for DCS depicted in Figure 2.7. DCS is composed of two LANs Ethernet, WAN and endsystems.

Points indexed by 1 and 2 denote points through which LAN1 and LAN2 are connected to WAN. Points 3, 4, and 5 denote WAN access points for LANs and computer C.

Suppose, that each of LANs has bus with transmission rate *C = 10 Mb/s* and physical length *L = 0.8 Km*, length of network packet *l = 1 Kb*. Assuming the signal propagation delay equal to *5 mcs/Km* we get one for the bus $p = 5 \cdot L$ *= 4 mcs/Km*. Then the packet delay *t = l/C = 100 mcs* and maximum throughput of the LAN Ethernet is determined by formula [13] *A = C / [1 + (2e + 1)p/t] = 8 Mb/s*. Suppose that LAN1 has $A_1 = 5Mb/s$ and LAN2 has $A_2 = 6Mb/s$ of available capacities for a DMA location.

Let us assume that WAN access points 3, 4, 5 have available capacities $A_3 = 3Mb/s$, $A_4 = 10Mb/s$ and $A_5 = 20Mb/s$ respectively.

In Figure 2.8 the DCS is represented by the system graph. The points on links indexed by numbers correspond to access points to communication resources of DCS. (We shall name the points shortly commpoints). The commpoint representation in the system graph is useful to compute available capacity of every virtual channel connection using formula (2.1). For example, capacity of channel (A,C) is $R_{AC} = min\{A_1, A_3, A_4\} = 3Mb/s$.

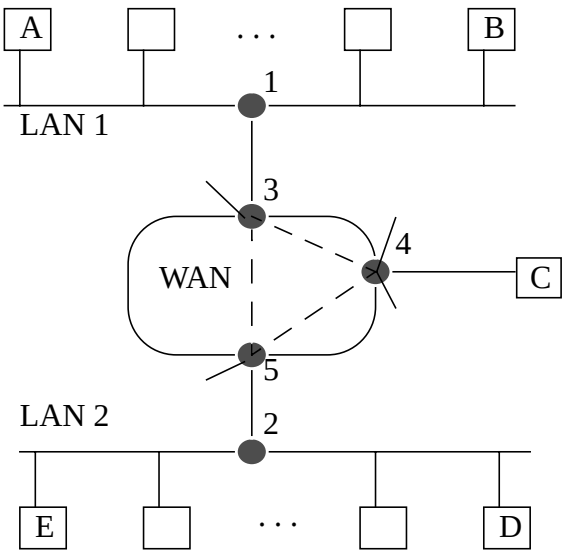


Figure 2.7. An example of DCS topology

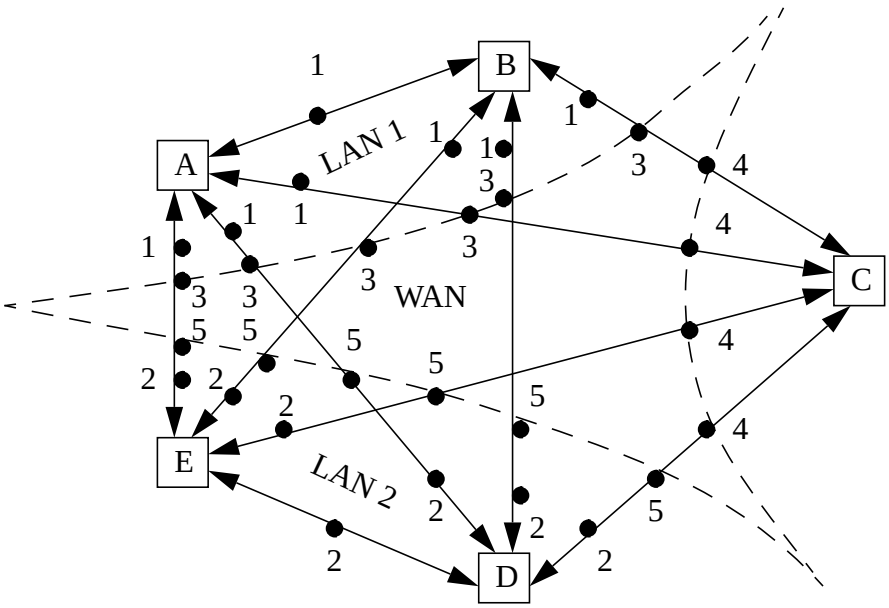


Figure 2.8. DCS graph representation with commpoints

Preliminary analysis of commpoint capacities allows to simplify DCS graph and further capacity computations. For example, summary capacity of all virtual channels routed over access point 4 (line of all commpoints 4) is less than capacity of the access point, i.e.

$R_{AC} + R_{CA} + R_{BC} + R_{CB} = min\{A_1, A_3, A_4\} = A_3 = 3,$

$R_{DC} + R_{CD} + R_{EC} + R_{CE} = min\{A_2, A_4, A_5\} = A_2 = 6,$

and consequently

$R_{AC} + R_{CA} + R_{BC} + R_{CB} + R_{EC} + R_{CE} + R_{DC} + R_{CD} = A_3 + A_2 = 9 < A_4 = 10.$

Therefore we can discard commpoint 4. In the same way commpoint 5, for which $min\{A_1, A_3, A_5\} = A_3$, $min\{A_2, A_4, A_5\} = A_2$, $A_3 + A_2 = 3 + 6 = 9 < A_5 = 20$, can be discarded too. In Figure 2.9, the final system graph is presented. Commpoints 1, 2, 3 corresponds to DCS communication resources of LAN1, LAN2 and WAN respectively.

Assuming that available resources of computers are $R_A = 5, R_B = 4, R_C = 6, R_D = 8, R_E = 7$, one can represent initial values of DCS resources available to mapped DMA by following matrix *R*

	A	B	C	D	E
A	5	5	3	3	3
B	5	4	3	3	3
C	3	3	6	6	6
D	3	3	6	8	6
E	3	3	6	6	7

(2.3) .

Note that capacities of virtual channels are interdependent through common communication resources used (see Figure 2.9).

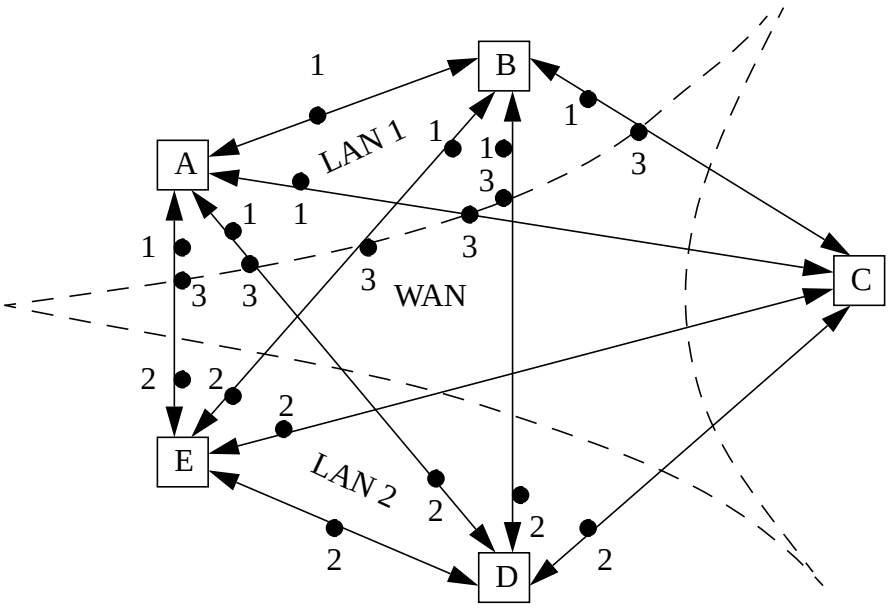


Figure 2.9. Final DCS graph representation with commpoints

3 Approach to the Mapping Problem Solution

An approach proposed consists of following procedures (see Fig. 3.1):

1. Attached Component Location Procedure for locating attached components[1] of DMA onto DCS.

   If not all attached components can be assigned because of a gap of DCS resources then Reduction Procedure is performed as next one to obtain a relaxed solution for DMA allocation.

[1] For such component there is only one particular computer that the component is assigned to. Such components are called attached ones. Generally, set of computers on which a component can be assigned depends on whether the computer configuration has devices and resources required to perform multimedia functions of a certain component type. On the other hand, source-components and/or sink-components can be assigned to certain computers in advance. Then the components are attached ones too.

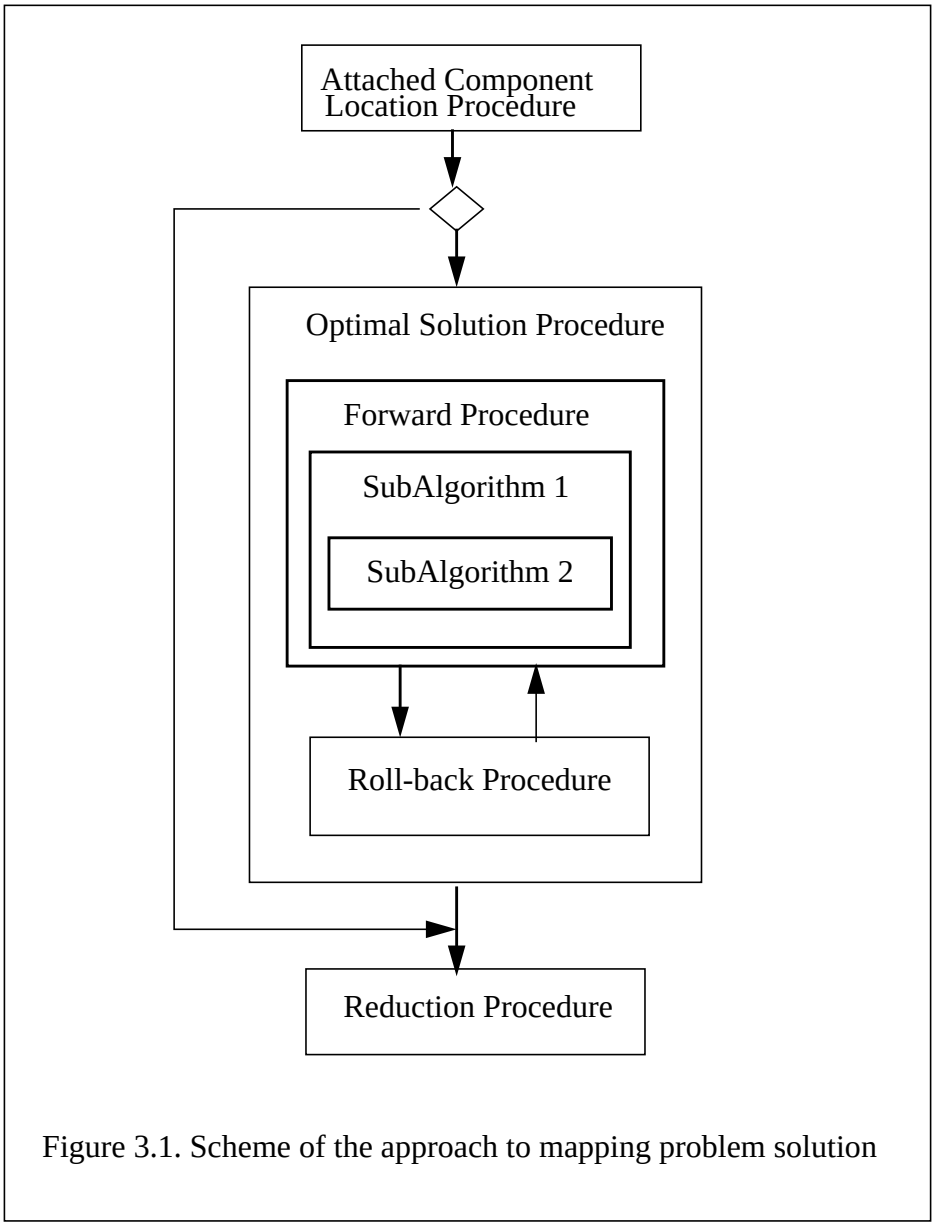


Figure 3.1. Scheme of the approach to mapping problem solution

2. Optimal Solution Procedure.

   The procedure obtains the solution of the mapping problem with cost objective function and unrelaxed resource requirements. This procedure uses Forward procedure, that executes component assignments, and Roll-back Procedure, that allows to look possible component assignments over. SubAlgorithm 1 and subAlgorithm 2 solve relaxed subProblems of computing bounding functions used in the branch-bound method.

3. Reduction Procedure

   The procedure is performed only if DCS resources available to mapped DMA are not enough to meet resource requirements of DMA. The procedure obtains the location of

DMA that guarantees minimum resource requirement relaxation for components and links of DMA.

4 Problem Formulation

We are given

- application graph of DMA with

  set of nodes (components) η,

  set of directed arcs connecting components with each other $\lambda = \{(i,j), i,j \in \eta\}$,

  set of required computational resources for components $\{d_i, i \in \eta\}$ and required communication resources $\{d_{ij}, (i,j) \in \lambda\}$,
- DCS graph with

  set of nodes (computers) ζ,

  set of VCs (or simply channels) connecting computers with each other $\pi = \{(n,m), n,m \in \zeta\}$,

  set of available (vacant) computational resources of computers $\{R_n, n \in \zeta\}$,

  set of communication resources ρ in the DCS[1],

  set of communication resources of DCS ρ_{nm} used by channel (n,m), $\rho_{nm} \in \rho$, $\bigcup_{(n,m) \in \mu} \rho_{nm}$,

  set of channels π_s routed over shared communication resource $s \in \rho$, $\bigcup_{s \in \rho} \pi_s = \pi$,

  set of communication resource capacities available to mapped DMA A_s, $s \in \rho$,

  set of resource cost functions for computers $\{f_n(D), n \in \zeta\}$ and for channels $\{g_{nm}(D), (n,m) \in \mu\}$,

  set of acceptable locations of every component in the DCS $\{\zeta_i, i \in \eta\}$,

The solution variables are x_{in} such that $x_{in} = 1$, if component i is assigned to computer n, and $x_{in} = 0$ otherwise.

Then the Original Mapping Problem is

$$F(x_{in}) = min_{x_{in}} \{ \sum_{i \in \eta} \sum_{n \in \zeta_i} x_{in} f_n(d_i) + \sum_{(i,j) \in \lambda} \sum_{(n,m) \in \pi} x_{in} x_{jm} g_{nm}(d_{ij}) \}, \quad (4.1)$$

$$g_{nm}(d_{ij}) = \sum_{s \in \pi_{nm}} g_s(d_{ij})$$

[1] Interfaces of DCS computers can be also included into set ρ.

subject to

$$\sum_{n \in \zeta_i} x_{in} = 1, \forall i \in \eta \tag{4.2}$$

$$\sum_{i \in \eta} x_{in} d_i \le R_n, \forall n \in \zeta \tag{4.3}$$

$$\sum_{(n,m) \in \pi_s} \sum_{(i,j) \in \lambda} x_{in} x_{jm} d_{ij} \le A_s, \forall s \in \rho \tag{4.4}$$

where $g_{nm}(d)$ is the cost function for channel (n, m); $g_{nm} = 0$ if $n = m$; $g_{nm} \ge 0$ if $n \ne m$ and $(n, m) \in \pi$; $g_{nm} = \infty$ if $(n, m) \notin \pi$ or $R_{nm} = \varepsilon_{nm} \ge 0$. Here ε_{nm} is the available capacity of channel $(n, m) \in \pi$ that is not enough to satisfy a resource requirement of any arc of component graph, $\varepsilon_{nm} < min\{D_{ij}\}$;

In this formulation, objective function F minimizes the summary cost of computational and communication resources used for the DMA allocation in the DCS. The first term in the objective function identifies the cost of resources of computers that components are assigned to. The second term represents the cost of communication resources of channels on which DMA arcs are placed.

Constraint set (4.2) guarantees that every component $i \in \eta$ will be placed into the DCS and only onto one computer.

Constraint set (4.3) guarantees that resources used by components assigned to a computer do not exceed the available resource of the computer.

Constraint set (4.4) guarantees that capacity of communication resource s in DCS used by all DMA arcs placed on resource s do not exceed the available capacity of the resource.

Analysis of the objective function and constraints of the mapping problem (4.1) - (4.4) shows that it is, in general, a nonlinear integer programming problem with Boolean variables.

5 Procedure for Location of Attached Components

Let us present an additional notation of variable sets used further:

η^a be the set of components assigned to computers, $\eta^a \subseteq \eta$;

η^u be the set of components that are not yet assigned to computers, $\eta^u = \eta \backslash \eta^a$;

η_n be the set of components assigned to computer $n \in \zeta$, $\bigcup_{n \in \zeta} \eta_n = \eta^a$;

$c(i)$ be the function returning the index of DCS computer $n \in \zeta$ to which component $i \in \eta^a$ is assigned;

$\zeta_i\left(\eta^u\right)$ be the set of computers on which component $i \in \eta^u$ may be assigned provided that components of the $\eta^a = \eta \backslash \eta^u$ are already allocated in the DCS, $\zeta_i\left(\eta^u\right) \subseteq \zeta_i(\eta), \eta^u \subseteq \eta$;

$\zeta\left(\eta^u\right)$ be the set of computers that are acceptable for component locations provided that components of the $\eta^a = \eta \backslash \eta^u$ are already allocated in the DCS, $\zeta\left(\eta^u\right) = \bigcup_{i \in \eta^u} \zeta_i\left(\eta^u\right)$, $\zeta\left(\eta^u\right) \subseteq \zeta(\eta), \eta^u \subseteq \eta$.

λ^a be the set of DMA arcs already assigned, $\lambda^a \subseteq \lambda$ [1];

λ^u be the set DMA arcs that are not yet assigned, $\lambda^u = \lambda \backslash \lambda^a$;

λ_{nm} be the set of DMA arcs assigned to DCS channel $(n, m) \in \pi$;

λ_s be the set of DMA arcs assigned to communication resource $s \in \rho$, $\lambda_s = \bigcup_{(n, m) \in \pi_s} \lambda_{nm}$;

ΔR_{nm} be the available capacity of channel (n, m);

The algorithm of the procedure is as follows:

1. Initializing work variables:

$\eta^u = \eta$; $\eta^a = \varnothing$,
$\zeta_i\left(\eta^u\right) = \zeta_i, \forall i \in \eta^u$; $\zeta\left(\eta^u\right) = \zeta$;
$\Delta R_n = R_n$; $\eta_n = \varnothing, \forall n \in \zeta$
$\lambda^u = \lambda$; $\lambda^a = \varnothing$;
$\lambda_{nm} = \varnothing, \forall (n, m) \in \pi$
$\Delta A_s = A_s$; $\lambda_s = \varnothing, s \in \rho$

2. Assigning attached components and their adjacent arcs:

for every attached component i

$\Delta R_{c(i)} \Leftarrow \Delta R_{c(i)} - d_i$; $\eta_{c(i)} \Leftarrow \eta_{c(i)} \cup i$; $\eta^a \Leftarrow \eta^a \cup i$; $\eta^u \Leftarrow \eta^u \backslash i$;

for every output arc $(i, j) \in \lambda^u$ such that $\left(j \in \eta^a\right) \wedge (c((i) \neq c(j)))$

$\lambda^a \Leftarrow \lambda^a \cup (i, j)$; $\lambda^u \Leftarrow \lambda^u \backslash (i, j)$; $\lambda_{c(i)c(j)} \Leftarrow \lambda_{c(i)c(j)} \cup (i, j)$;

for every communication resource $s \in \rho_{c(i)c(j)}$

$\lambda_s \Leftarrow \lambda_s \cup (i, j)$; $\Delta A_s \Leftarrow \Delta A_s - d_{ij}$;

for every input arc $(j, i) \in \lambda^u$ such that $\left(j \in \eta^a\right) \wedge (c((i) \neq c(j)))$

---

[1] We treat that a DMA arc is assigned if it is located into a DCS channel or components, connected by the arc, are placed into a same DCS computer. The last case we call as arc absorption.

$\lambda^a \Leftarrow \lambda^a \cup (j, i)$; $\lambda^u \Leftarrow \lambda^u \setminus (j, i)$; $\lambda_{c(j)\,c(i)} \Leftarrow \lambda_{c(j)\,c(i)} \cup (i, j)$

for every communication resource $s \in \rho_{c(j)\,c(i)}$

$\lambda_s \Leftarrow \lambda_s \cup (i, j)$; $\Delta A_s \Leftarrow \Delta A_s - d_{ji}$;

3. Computing capacities of DCS channels:

   for every channel $(n, m) \in \pi$

   $\Delta R_{nm} = min\{\Delta A_s, \forall s \in \rho_{nm}\}$

4. Checking a resource gap of the DCS and computing cost F of such assignments:

   If resources of DCS was enough for allocation of attached components, i.e. $\Delta R_n \geq 0, \forall n \in \zeta$ and $\Delta A_s \geq 0, \forall s \in \rho$,

   then computing initial value of the cost function $F_0 = \sum_{i \in \eta^a} f_{c(i)}(d_i) + \sum_{(i,j) \in \lambda^a} g_{c(i)\,c(j)}(d_{ij})$ and go to Optimization Procedure

   else go to Reduction Procedure.

6 Optimal Solution Procedure

The procedure obtains the optimal solution of the Original Mapping Problem (4.1) - (4.6) with cost objective function and unrelaxed resource requirements.

6.1 Problem Solution Technique

A solution technique proposed is based on a branch-bound method [11]. Performance of the method can be represented by a directed search-tree (see Figure 6.1). The vertices of the tree correspond to the subsets of complete set $S = \{x_{in} = 0 \text{ or } 1, \forall i \in \eta, \forall n \in \zeta\}$ of possible DMA locations into DCS. The vertices are partitioned into levels in the following way:

There is a single vertex of level 0 (called the root of the tree) which corresponds to the complete set S consisting of $2^{I \cdot N}$ elements[1], where $I = |\eta|$ is number of components in DMA and $N = |\zeta|$ is number of computers in DCS.

To construct the level 1 of the tree, we must first choose a component, for example indexed by 1. The level 1 contains N vertices denoted by (1,1), (1,2),..., (1,N). The first one corresponds to the subset $S_{(1,1)}$ of S for which $x_{11} = 1, x_{12} = 0, x_{13} = 0, \ldots, x_{1N} = 0$, i.e. component 1 is assigned to compute 1. The last vertex $(1, N)$ of level 1 corresponds to the subset $S_{(1,N)}$ of S for which $x_{11} = 0, x_{12} = 0, x_{13} = 0, \ldots, x_{1N} = 1$, i.e. component 1 is assigned to computer

[1] In accordance with constraint (4.2), every component can be placed into only one computer. Therefore set S consists of N^I feasible elements.

N. We have, evidently, $S_{(1,\,n)} \subset S$, $n = \overline{1, N}$. Subsets $S_{(1,\,n)}$, $n = \overline{1, N}$ form a partition of set S. We say that we have 'branched' S with respect to the component 1.

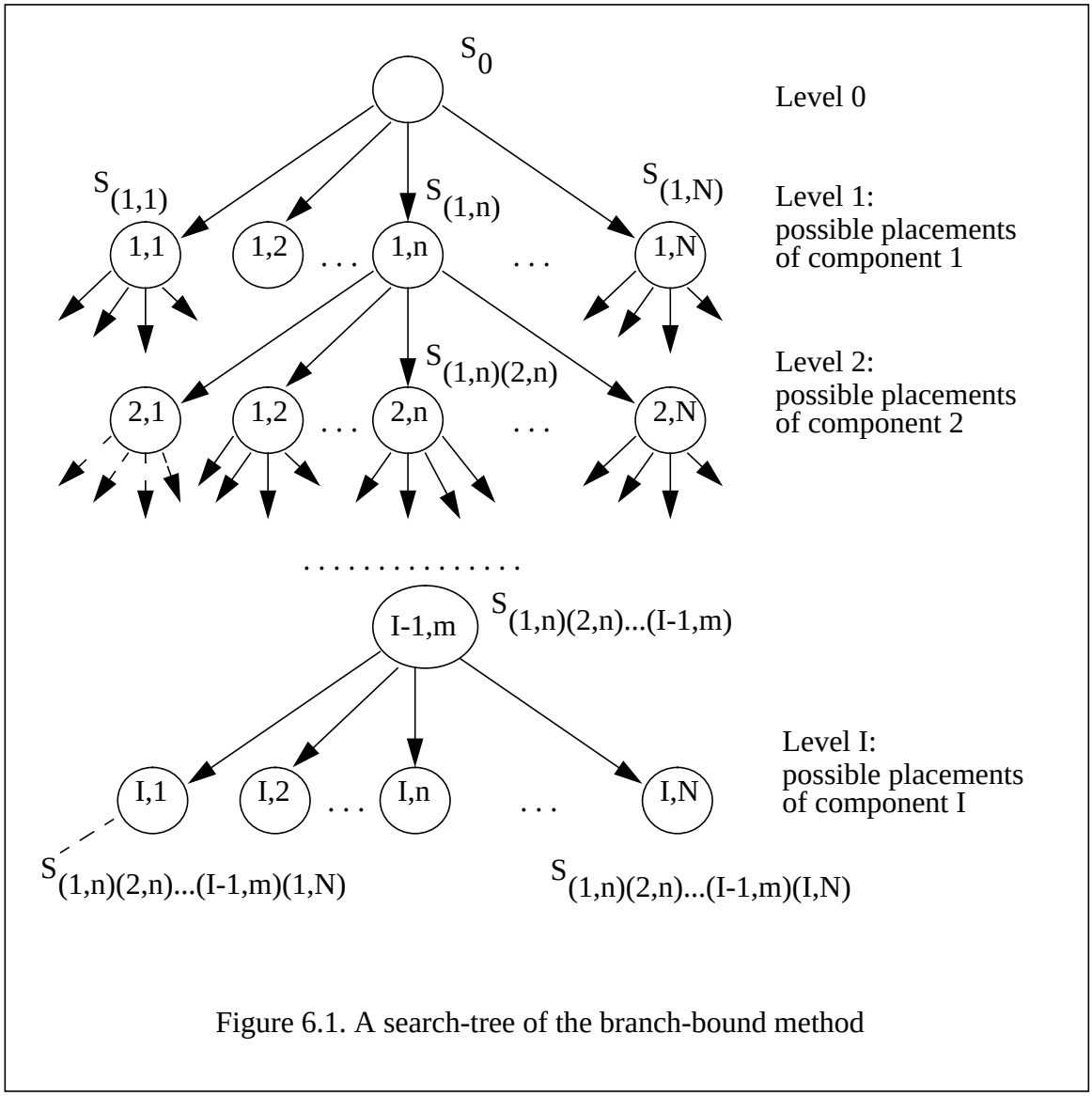


Figure 6.1. A search-tree of the branch-bound method

In the same way for every vertex of level 1, we construct level 2. In Figure 6.1 set $S_{(1,\,n)}$ is branched with respect to the component 2 and obtained subsets $S_{(1,\,n)\,(2,\,m)}$, $m = \overline{1, N}$, form a partition of set $S_{(1,\,n)}$.

The lowest level I of the tree corresponds to placements of the last component I of DMA. Every subset of terminal vertex of level I corresponds to a single element of S and determines one of DMA placements. The placement is determined by all vertices intersected by the branch joining the root to the terminal vertex. For example vertex (I, N) determines DMA placement $x_{1n} = 1, x_{2n} = 1, \ldots, x_{I-1,\,m} = 1, \ldots, x_{1N} = 1$, other $x_{in} = 0$.

To find the optimal solution of the mapping problem of important size without having to construct complete search-tree, the concept of bounding function [1] is used.

We present a branch-bound method, that take into account peculiarities of the mapping problem (4.1)-(4.4) by choosing

(a) the bounding function,

(b) the branching vertex at each stage,

(c) the component (variable $i \in \eta$) relative to which the branching of the chosen vertex has to be done.

Let us examine these three points.

At each stage of an algorithm for every terminal vertex t of the search-tree, we construct a lower bound B_t of objective function (1). For the choice of (b) we use the so-called 'depth first search' method [1], which chooses a vertex of maximum depth among those vertices not yet branched. If there is more than one, then one could choose that which corresponds to the lowest bounding value. This method aims at exhibiting a (good) solution of the problem as soon as possible. Then value F_s of the got solution is used for narrowing the domain of optimal solution search by rejection of those vertices t for which $B_t > F_s$. By limiting, at each stage, the computation of the bounds to the successors of only those vertices t for which $B_t \le F_s$, a considerable reduction of the number of vertices actually examined can be obtained, a reduction sufficient to deal with problems of a fairly large size.

When a terminal vertex with objective function value F_s of a DMA location into DCS is reached, one attempts to improve obtained solution by choosing and branching terminal vertices with bounds $B_t \le F_s$. The consideration of such terminals is started from the terminal vertex of the obtained solution towards the root of the search tree. This approach aims to improve the solution as soon as possible and, using the improved solution, allows to decrease the set of terminal vertices examined at upper levels of the search tree.

Every obtained solution is checked whether it is optimal. An obtained one is optimal if

- the bounding value for a terminal vertex of the lowest level I is satisfied to equality $F_s = B_0$, where B_0 is an initial value of lower bound for the root of the search-tree,
- or all terminal vertices of all levels have bounding values $B_t < F_s$.

For the choice of (c), two approaches are presented:

- one is based on a fixed ordering components is presented;
- other is based on the so-called 'penalty' method that associates with every pair (i, n) - acceptable location of not yet assigned component i to computer n - a number p_{in} equal to the penalty if this assignment will be not done. If the branching is then carried out relative to the pair (i, n) associated with the largest penalty p_{in}, then one obtains two new subsets, one of which has a much better chance of containing an optimal solution than the other.

In some way it is a question of the ‘most informative and quick’ choice. It leads to minimizing the risk of having to explore in vain a branch of the tree while the optimal solution is contained in the other branch.

Further, we refer to the first algorithm based on the fixed ordering components as Ord-Algorithm and to the second one as Penalty-Algorithm.

Computational efficiency of algorithms, based on the branch-bound method, depends on an accuracy and computational complexity of a method for construction of an objective function bound for vertices of the search-tree. The accurate is the method the more unpromising branches are rejected during an optimization procedure. However increasing an accuracy of bounds causes increasing the computational complexity.

Let us consider the construction of an objective function bound for every vertex of the search-tree.

Assume that some components η^a and arcs connecting their with each other are already assigned. Using formula (4.1), one can computes the placement cost $F_0\left(\eta^a\right)$ for these components.

The cost is an exact partial of the complete cost $F = F_0\left(\eta^a\right) + F_1\left(\eta^u\right)$, where $F_1\left(\eta^u\right)$ is the cost of placement of other components η^u and their arcs that are not yet assigned. To construct a lower bound B for F_1, we propose an approach based on a search of independent best placement (with minimum cost) of every unassigned component into DCS taking into account only available resources of DCS not used. Then for every vertex one has to examine not more than N acceptable placements for every yet unassigned component $i \in \eta^u$. Therefore for every vertex one has to examine not more than $N\left|\eta^u\right| \le N \cdot I$ placements of individual components. Summary number of component placements examined is not more than $N \cdot I \cdot V$, where V is the number of vertices examined in the search tree.

Thus solving the mapping problem is advantageous if number of examined vertices V is considerable less than summary number of vertices in the search-tree, i.e. $V \ll N^I$. Value of V depends on an accuracy of lower bound construction.

Two methods of independent component placements are proposed:

- the first one (see Figure 6.2,a) bases on a placement of a component with his adjacent arcs and components that are not yet assigned,
- the second one (see Figure 6.2,b) bases on a placement of an unassigned component only.

Evidently, the first method is characterized by more exact bound B construction but more tedious complexity of bound computation too. However it allows to take into account resource requirements of adjacent arcs and components. Therefore it rejects more unpromising vertices in the search-tree and can obtain a considerable reduction of the number of vertices actually examined.

On the other hand, the second method can be useful for the case when channels of DCS have the same cost and DCS has sufficiently much communication resources such that constraints (4.4) have a faint effect on the problem solution.

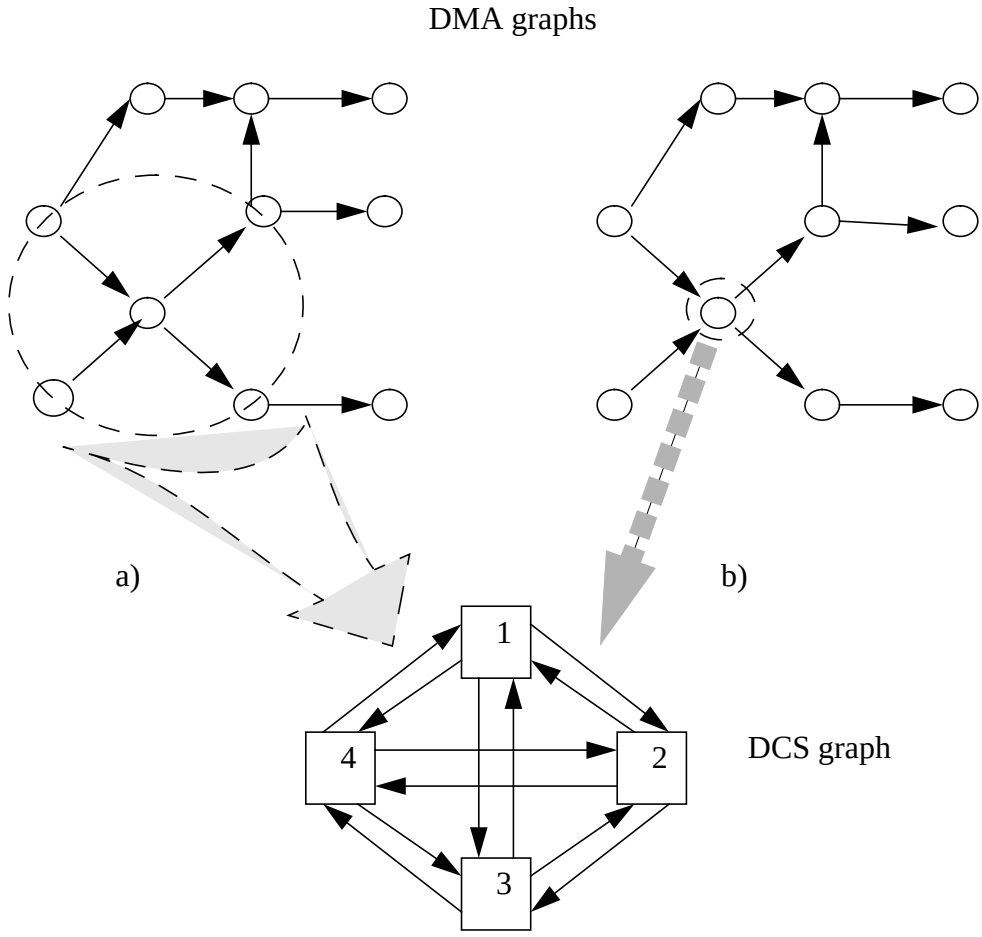


Figure 6.2. Mapping a DMA component to DCS
a) with his adjacent arcs and components or
b) without ones

Further we consider in detail the first method of bound construction which is more general.

Now let us examine constructing the search tree.

Suppose that, for any terminal vertex of the search-tree, $|\eta^a| < |\eta|$ component assignments $(\eta_n, n \in \zeta)$ have already been made, $\bigcup_{n \in \zeta} \eta_n = \eta^a$. The DMA arc assignments to DCS channels $(\lambda_{nm}, (n, m) \in \pi)$, $\bigcup_{(n, m) \in \pi} \lambda_{(n, m)} = \lambda^a$ depends in a unique fashion on the component assignments. On the other hand, having the arc assignments to channels, one can get corresponding arc assignments to DCS communication resources $\left(\lambda_s = \bigcup_{(n, m) \in \pi_s} \lambda_{nm}, s \in \rho\right)$. Thus, components of set η^a have already been located and computers of set $\zeta\left(\eta^a\right)$ are acceptable for location of not yet assigned components $i \in \eta^u$. This means that every computer

$n \in \zeta\left(\eta^u\right)$ has enough resources to locate at least one of components $i \in \eta^u$. For terminal vertex $(\eta_n, n \in \zeta)$, the Original Problem (4.1)-(4.4) can be rewritten as follows:

$$F = F_0\left(\eta^a\right) + F_1\left(\eta \backslash \eta^a\right) = F_0\left(\eta^a\right) + F_1\left(\eta^u\right) \tag{6.1}$$

$$F_0\left(\eta^a\right) = \sum_{i \in \eta^a}\left(f_{c(i)}(d_i) + \sum_{\left(\substack{j \in Out(i) \\ j \in \eta^a}\right)} g_{c(i),j}(d_{ij})\right) \tag{6.2}$$

$$F_1\left(\eta^u\right) = min_{x_{in}}\left\{\sum_{i \in \eta^u} \sum_{n \in \zeta_i(\eta^u)} x_{in}\left(M_{in}\left(\eta^a\right) + L_{in}\left(\eta^u\right)\right)\right\} \tag{6.3}$$

$$M_{in}\left(\eta^a\right) = f_n(d_i) + \sum_{\left(\substack{j \in Out(i) \\ j \in \eta^a}\right)} g_{n,c(j)}(d_{ij}) + \sum_{\left(\substack{j \in In(i) \\ j \in \eta^a}\right)} g_{c(j),n}(d_{ji}) \tag{6.4}$$

$$L_{in}\left(\eta^u\right) = \frac{1}{2}\left(\sum_{\left(\substack{j \in Out(i) \\ j \in \eta^u}\right)} \sum_{m \in \zeta_j(\eta^u)} x_{jm} g_{nm}(d_{ij}) + \sum_{\left(\substack{j \in In(i) \\ j \in \eta^u}\right)} \sum_{m \in \zeta_j(\eta^u)} x_{jm} g_{mn}(d_{ji})\right) \tag{6.5}$$

subject to

$$\sum_{n \in \zeta_i(\eta^u)} x_{in} = 1, \forall i \in \eta^u \tag{6.6}$$

$$\sum_{i \in \eta^u} x_{in} d_i \le \left(R_n - \sum_{i \in \eta_n} d_i\right) = \Delta R_n(\eta_n), \forall n \in \zeta\left(\eta^u\right) \tag{6.7}$$

$$\sum_{(n,m) \in \pi_s} \sum_{(i,j) \in \lambda^u} x_{in} x_{jm} d_{ij} \le \left(A_s - \sum_{(i,j) \in \lambda_s} d_{ij}\right) = \Delta A_s(\lambda_s), s \in \rho \tag{6.8}$$

where $In(i)$ is the set of components that are adjacent to component i and are direct senders to component i; $Out(i)$ is the set of components that are adjacent to component i and are direct recipients from the component in the DMA graph.

In formula (6.1) $F_0\left(\eta^a\right)$ characterizes the cost of component and their arcs assignments that have been already done. $F_1\left(\eta^u\right)$ is the objective function of minimization problem for further placement of not yet assigned components $i \in \eta^u$ and their arcs.

Term $M_{in}\left(\eta^a\right)$ of $F_1\left(\eta^u\right)$ takes into account computational resource cost of component i placed onto computer n and the cost of communications of component i with already assigned components $j \in \eta^a$. Term $L_{in}\left(\eta^u\right)$ determines the cost of communications of component i with not yet assigned components.

Constraints (6.6) - (6.8) take into account resources of computers $\Delta R_n(\eta_n)$, $\forall n \in \zeta(\eta^u)$ and capacities of communication resources $\Delta A_s(\lambda_s)$, $\forall s \in \rho$ remaining after previous $|\eta^a|$ assignments.

The factor of 1/2 is used in (6.5) in order that the responsibility for the communication costs common to the two assignments, component i and adjacent component j, will be divided equally between these assignments[1].

6.2 Problem Relaxation

Let us derive a lower bound of F in (6.1) for a terminal vertex that is characterized by vector of already executed component assignments $(\eta_n, n \in \zeta)$. The derivation is based on the assumption that every unassigned component $i \in \eta^u$ and unassigned arcs adjacent to him can be placed independent of other unassigned ones. Cost term $F_0(\eta^a)$ in formula (6.1) is a constant. Therefore let us derive a lower bound of $F_1(\eta^u)$.

$$F_1(\eta^u) = min_{x_{in}}\{\sum_{i \in \eta^u} \ldots\} >=$$

$$\sum_{i \in \eta^u}\left(min_{n \in \zeta_i(\eta^u)} S_{in}(\eta^a, \eta^u)\right) = b_1(\eta^u),$$

$$S_{in}(\eta^a, \eta^u) = M_{in}(\eta^a) + min_{x_{jm}}\{L_{in}(\eta^u)\} \quad , (6.9)$$

where $M_{in}(\eta^a)$ and $L_{in}(\eta^u)$ are determined by formula (6.4) and (6.5) respectively.
Thus the lower bound is

$$B(\eta^a, \eta^u) = F_0(\eta^a) + b_1(\eta^u) = F_0(\eta^a) + \sum_{i \in \eta^u}\left(min_{n \in \zeta_i(\eta^u)} S_{in}(\eta^a, \eta^u)\right) \le F \quad (6.10)$$

Using lower bound $b_1(\eta^u) \le F_1(\eta^u)$ we can relax the problem (6.3) - (6.8) to two subproblems:

subProblem 1. For every component $i \in \eta^u$, the potential best assignment is deduced by solving the problem

$$min_{n \in \zeta_i(\eta^u)} S_{in}(\eta^a, \eta^u) = min_{n \in \zeta_i(\eta^u)}\left(M_{in}(\eta^a) + min_{x_{jm}}\{L_{in}(\eta^u)\}\right) \quad (6.11)$$

[1]This approach allows to take into account communication resource expenses, caused by all output and input arcs of a component, each time when the component is assigned to a computer, and therefore to get more exact value of bounfing function further.

subject to [1]

$$d_i \le \Delta R_n(\eta_n),\ \forall n \in \zeta\left(\eta^u\right)$$

subProblem 2. For every acceptable assignments of component $i \in \eta^u$ to computer $n \in \zeta_i\left(\eta^u\right)$, the potential best placement of unassigned arcs adjacent to the component is deduced by solving the problem

$$^{min}L_{in}\left(\eta^u\right) = min_{x_{jm}}\{L_{in}\left(\eta^u\right)\} \quad =$$

$$= \frac{1}{2} min_{x_{jm}} \left\{ \sum_{\left(\substack{j \in Out(i) \\ j \in \eta^u}\right)} \sum_{m \in \zeta_j(\eta^u)} x_{jm} g_{nm}(d_{ij}) + \sum_{\left(\substack{j \in In(i) \\ j \in \eta^u}\right)} \sum_{m \in \zeta_j(\eta^u)} x_{jm} g_{mn}(d_{ji}) \right\} \quad (6.12)$$

subject to

for every computer m adjacent to computer n, i.e. $\forall m \in \bigcup_{j \in Out(i) \cap In(i)} \zeta_j\left(\eta^u\right)$,

$$\sum_{j \in \eta^u} x_{jm} d_j \le \Delta R_m(\eta_m) \quad (6.13)$$

for every communication resource over which output channel (n, m) and input ones (m, n) of computer n are routed, i.e. $\forall s \in \rho_{nm} \cup \rho_{mn}$, $\forall m \in \bigcup_{j \in Out(i) \cup In(i)} \zeta_j\left(\eta^u\right)$,

$$\sum_{(n,m) \in \pi_s} \sum_{j \in \eta^u} x_{jm} d_{ij} + \sum_{(m,n) \in \pi_s} \sum_{j \in \eta^u} x_{jm} d_{ji} \le \Delta A_s(\lambda_s) \quad (6.14)$$

Thus the bounding function $B\left(\eta^a, \eta^u\right)$ is computed by formula (6.10) that needs, for every unassigned component, to solve the subProblem 1 of the potential best assignment of the component. At that the solution of subProblem 1 needs to solve, for every acceptable assignment of the component, subProblem 2 of the potential best placement of his unassigned arcs.

6.3 Forward Procedure

The Forward Procedure executes following key operations:

---

[1] The form of objective function (6.11) allows us to argue: if subProblem 1 has solution then constraints (6.6) will be necessary satisfied else component i can not be allocated and the corresponding vertex $(\eta_n, n \in \zeta)$ does not contain an acceptable solution. If the vertex is one of level 0, i.e. $(\varnothing, \eta)$ (called the root of the directed search-tree) and component i can not be allocated, then the original problem (4.1) - (4.4) has not a solution.

- Choosing next component for location.
- Examining all acceptable locations of the component.
- Computing by formula (6.10) the bounding value for every acceptable location of the component using subAlgorithm 1 (see below).
- Choosing the best location of the component using the bounding values.
- Checking an obtained acceptable component location whether it is optimal.

The Forward Procedure is based on the fixed ordering of component assignments and it's presentation is as follows:

1. **If** there are 'non-alternative' assignments[1] for components yet unassigned

   then

   executing the assignments;

   increasing cost F_0 in the value of expenses for such assignments;

   If at least one 'non-alternative' component can not be assigned

   then go to Roll-back Procedure;

   If all components are assigned **then** the optimal solution is obtained; stop.

2. Ordering and renumbering components $i \in \eta^u$ in decreasing with respect to d_i (or $f(d_i) + \sum_j [g(d_{ij}) + g(d_{ji})]$, where f and g are cost functions of an arbitrary computer and channel of DCS respectively)[2].
3. Computing initial bounding value for current state of assignments performed $B_0\left(\eta^a, \eta^u\right)$ in accordance with formula (6.10) using subAlgorithm 1, step 2 (see below).

4. **If** there are 'non-alternative' assignments for components yet unassigned

   then

   executing the assignments;

   increasing cost F_0 in the value of expenses for such assignments;

   if at least one 'non-alternative' component can not be assigned

   then go to Roll-back Procedure

   else go to step 10

---

[1] We say that component i has the 'non-alternative' assignment if and only if one computer of DCS, for example $n = c(i)$, can meet computational resource requirements of component i, i.e. constraint $\Delta R_n \geq d_i, i \in \eta^u$ is satisfied only for computer n and fails for all other computers.

[2] At this step an order of summation with respect to index i in (6.10) is determined.

5. Choosing the next component l from ordered set η^u to perform his assignment, i.e.
   $l \Leftarrow l + 1$

6. $\left|\zeta_l\left(\eta^u\right)\right| > 0$ terminal vertices will be built at level l of the search-tree. Each of such vertices $v = \overline{1, \left|\zeta_l\left(\eta^u\right)\right|}$ corresponds to one computer $n \in \zeta_l\left(\eta^u\right)$, that component l could be assigned to.
   For all terminal vertices of level l, corresponding bounding values $B_{ln}\left(\eta^a \cup l, \eta^u \backslash l\right)$ are computed in accordance with formula (6.10) by subAlgorithms 1 and 2 solving subProblems 1 and subProblems 2.

7. Choosing the *terminal* vertex of level l that has the lowest bounding value in set $\{B_{ln}\left(\eta^a \cup l, \eta^u \backslash l\right), n \in \zeta_l\left(\eta^u\right)\}$. Suppose element $\hat{B}_{lk}$ corresponding to placement of component l into computer k is found.

8. **If** $\hat{B}_{lk} \geq Fbest$ **then** go to Roll-back Procedure

9. Assigning component l to computer k and recomputing cost function:
   $\eta^a \Leftarrow \eta^a \cup l;\ \eta^u \Leftarrow \eta^u \backslash l;\ \eta_n \Leftarrow \eta_n \cup l;$
   $\Delta R_k \Leftarrow \Delta R_k - d_l;\ F_0 \Leftarrow F_0 + f_k(d_l)$.

   Assigning arcs adjacent to component l and recomputing cost function:

   Every output arc (l, j) and input one (j, l) of component l, that were not yet assigned and connects the component with already assigned components $j \in \eta^a$, are assigned to channel $(k, c(j))$ and $(c(j), k)$ respectively and are included in sets of assigned arcs $\lambda^a, \lambda_{k, c(j)}, \lambda_{c(j), k}, \lambda_s$.

   Recomputing capacities of communication resources ΔA_s and channels R_{km}, R_{mk} of DCS that the arcs are assigned to.

   The algorithm for arc assignments is as follows:

   for every component $j \in Out(l) \cap \eta^a$
      for every shared resource $s \in \rho_{k, c(j)}$
         $\lambda^a \Leftarrow \lambda^a \cup (l, j)\ ;\ \lambda^u \Leftarrow \lambda^u \backslash (l, j)\ ;$
         $\lambda_s \Leftarrow \lambda_s \cup (l, j)\ ;\ \lambda_{k, c(j)} \Leftarrow \lambda_{k, c(j)} \cup (l, j)$
         $\Delta A_s \Leftarrow \Delta A_s - d_{lj};\ F_0 = F_0 + g_s(d_{lj})\ ;$

for every component $j \in In(i) \cap \eta^a$

for every shared resource $s \in \rho_{c(j),k}$

$\lambda^a \Leftarrow \lambda^a \cup (j, l)$; $\lambda^u \Leftarrow \lambda^u \setminus (j, l)$;

$\lambda_s \Leftarrow \lambda_s \cup (j, l)$; $\lambda_{k, c(j)} \Leftarrow \lambda_{k, c(j)} \cup (j, l)$

$\Delta A_s \Leftarrow \Delta A_s - d_{jl}$; $F_0 = F_0 + g_s(d_{jl})$;

for every component $j \in Out(l) \cap \eta^a$

$\Delta R_{k, c(j)} = min\{\Delta A_s, \forall s \in \rho_{k, c(j)}\}$;

for every component $j \in In(i) \cap \eta^a$

$\Delta R_{c(j), k} = min\{\Delta A_s, \forall s \in \rho_{c(j), k}\}$

10. **If** all components are assigned, i.e. $i = |\eta|$ or in other terms $\eta^a = \eta$

    then

    a. the vertex with value $\hat{B}_{|\eta|k} = F_0(\eta)$ determines an acceptable solution of problem (4.1) - (4.4). **If** another acceptable solution was obtained before, **then** choosing the best one out of the both and now $Fbest$ characterizes the cost of the current best solution. (At the beginning of Forward Procedure $Fbest = \infty$).

    b. Vertices for which

    $$B_{in} \geq Fbest,\ i = \overline{1, |\eta| - 1} \qquad (6.15)$$

    are not promising because they do not contain the optimal solution and, therefore, they are eliminated from further consideration.

    c. **If** $Fbest$ is equal to initial value of bounding function B_0, i.e. $Fbest = B_0$, or relation (6.15) is satisfied for all terminal vertices

    then found value $Fbest$ is optimal and, therefore, $F(x_{in}) = Fbest$, stop

    else the terminal vertices showing promise are branched and the search-tree is constructed further (by choosing the vertex and corresponding level l that is nearest one to the obtained solution and has a bounding value less than $Fbest$)[1] till relation (6.16) will be satisfied for all remaining terminal vertices. For that go to step 7.

11. **If** not all components are assigned, i.e. $l < |\eta|$ **then** go to step 4.

Let us consider in detail some steps of the algorithm.

At **step 1** and **4** 'non-alternative' component assignments are detected. For every such component i that have to be assigned to computer n it is checked whether capacities of adjacent channels of the computer are enough to place yet unassigned arcs of the component.

---

[1] For the vertex, all parameters $\eta^a, \eta^u, \eta_n, \lambda^a, \lambda^u, \lambda_{nm}, \lambda_s, \Delta R_n, \Delta R_{nm}, \Delta A_s, B_{in}$ have to be reset.

Note that assignment of component i to computer n may be possible if at least the following relations are satisfied:

- computer n has enough computational resource to locate component i, i.e.

$$d_i \le \Delta R_n(\eta_n) \tag{6.16}$$

- communication resources used for location of output and input arcs of component i (which are not yet assigned and connect the component with already assigned components) have enough available capacities, i.e.

$$\sum_{(n,m)\in\pi_s\left(\substack{j\in\eta^a \\ j/(c(j))=m}\right)} d_{ij} + \sum_{(m,n)\in\pi_s\left(\substack{j\in\eta^a \\ j/(c(j))=m}\right)} d_{ji} \le \Delta A_s(\lambda_s) \quad , \tag{6.17}$$

$$\forall s \in \rho_{nm} \cup \rho_{mn},\ \forall m / \left(j \in \eta^a \cap (In(i) \cap Out(i)) \right) \wedge (c(j) = m) .$$

Therefore the condition of resource capability to allocate a ‘non-alternative’ components is checked by following operation:

If for any ‘non-alternative’ component i, at least one relations of (6.17) is not satisfied

then go to Roll-back Procedure

else assigning the component i to computer n using procedure of step 9.

6.3.1 SubAlgorithm 1

The subAgorithm 1 computes bounding values for all vertices of a same level in the search-tree by examination every acceptable location of a component not yet assigned by the Forward Procedure. The locations of the components are considered as mutually independent. The subAlgorithm executes following key operations:

- Examining all acceptable locations of a not yet assigned component.
- Computing by formula (6.10) the bounding value for every acceptable location of the component using subAlgorithm 2 (see below).
- Choosing the best location of the component using the bounding values.

Suppose that we examine a level l of the search-tree at which we construct $\left|\zeta_l\left(\eta^u\right)\right|$ terminal vertices, each for one possible assignment of component l to corresponding computer of set $\zeta_l\left(\eta^u\right)$. Let us consider level l of the search-tree including vertices corresponding to possible assignments of component l. Then a representation of the subAlgorithm 1 is as follows.

for every computer $t \in \zeta_l\left(\eta^u\right)$, i.e. for every possible assignment of component l

1. Checking the resource capability of computer t and his adjacent channels to locate component l with his adjacent arcs:

   if relations (6.16) - (6.17) are satisfied for assignment of component l to computer t

   then 'virtual' assigning[1] component l to computer t and output and input arcs of component l, that were not yet assigned and connect component l with already assigned components $j \in \eta^a$, to corresponding channels of computer t.

   else corresponding terminal vertex of level l in the search-tree get bounding value $B_{lt} = \infty$ and considering the next possible location of component l, i.e the next computer $t \in \zeta_l\left(\eta^u\right)$, by return to the beginning of step 1.

2. Best placing of not yet assigned components $i \in \eta^u \backslash l$:

   for every unassigned component $i \in \eta^u \backslash l$

   $S_{ibest_u} = \infty$; (here S_{ibest} is cost value of the potential best placement of component $i \in \eta^u \backslash l$)

   a. **for** every computer $n \in \zeta_i\left(\eta^u\right)$, i.e. for every possible assignment of component i

      if the DCS has potentially sufficient computational and communication resources for assignment of component i to computer n, i.e. relations (6.16), (6,17) are satisfied

      then 'virtual' assigning component i and its arcs connecting with already assigned components $j \in \eta^a \cap (Out(i) \cup In(i))$; computing cost initial value of such assignments $S_{in} \Leftarrow M_{in}$ using formula (6.4)

      else component i can not be assigned to computer n, $S_{in} = \infty$, and go to step f

   b. **if** there are 'non-alternative' assignments of any component(s) adjacent to component i, i.e. of set $\eta^u \cap (Out(i) \cup In(i))$

      then

      'virtual' assigning 'non-alternative' components and their arcs connecting with component i; increasing cost S_{in} in the value of expenses for such assignments

      if not all 'non-alternative' components can be assigned

      then component i can not be assigned to computer n, $S_{in} = \infty$, and go to step f

      else go to step f

   c. **If** all components of $\eta^u \cap (Out(i) \cup In(i))$ are assigned

---

[1] 'Virtual' assignment means that only computational and communication resource expenses are calculated without real assignment of component to computers. The procedure of assignment is the same with one of step 9 of Forward procedure (see section 6.3) but work state-parameters are used instead of $\eta^a, \eta^u, \eta_n, \lambda^a, \lambda^u, \lambda_{nm}, \lambda_s, \Delta R_n, \Delta R_{nm}, \Delta A_s$, cost parameter B_{lt} is used instead of F_0 and t instead of k.

then an assignment of component i to computer n is acceptable and costs $S_{in} < \infty$, go to step f

d. Solving of subProblem 2 for given (i, n) to deduce the best potential placement of unassigned arcs of component i assigned (virtually) to computer n

e. **if** a solution of subProblem 2 is obtained
then increasing cost S_{in} in the got value ${}^{min}L_{in}$ of subProblem 2
else there is no solution for assignment of component i to computer n, $S_{in} = \infty$.

f. **if** $S_{in} < S_{ibest}$ **then** $S_{ibest} = S_{in}$

Recomputing lower bounding value for the terminal vertex corresponding to the assignment of component l to computer t: $B_{lt} \Leftarrow B_{lt} + S_{ibest}$

6.3.2 SubAlgorithm 2

The subAlgorithm 2 is used for problem (6.12) - (6.14) solution and performs locating all not yet assigned adjacent arcs of a component by the branch-bound method. It executes following key operations:

- Choosing next adjacent arc of given component i
- Examining all acceptable locations of the arc.
- Computing the bounding value for every acceptable location of the arc.
- Choosing the best location of the arc using the bounding values.

The objective function (6.12) of subProblem 2 takes into account only communication resource cost of mapping not yet assigned components adjacent to component i provided that component i is assigned to computer n. Constraints (6.13) guarantee that computational resources used by the adjacent components assigned to a computer do not exceed the available resource of the computer. Constraints (6.14) guarantee that summary capacity of the output and input arcs of component i placed onto communication resource s does not exceed the available capacity of the resource.

Let us consider an approach based on the branch-bound method to solve the problem (6.12) - (6.14).

Let us denote additional work parameters:

$\tilde{\eta}^a$ be the set of components assigned to computers including ‘virtual’ assignments performed by Forward Procedure;

$\tilde{\eta}_n$ be the set of components assigned to computer n including 'virtual' assignments, $\bigcup_n \tilde{\eta}_n = \tilde{\eta}^a$;

$\tilde{\lambda}_s$ be the set of arcs assigned to communication resource $s \in \rho$ including 'virtual' assignments;

$\tilde{\eta}^u$ be the set of components that are not yet assigned taking into account 'virtual' assignments performed, $\tilde{\eta}^u = \eta \backslash \tilde{\eta}^a$. Note, that subAlgorithm 2 gets sets $\tilde{\eta}^a$, $\tilde{\eta}_n$, $\tilde{\lambda}_s$ and $\tilde{\eta}^u$ from the subAlgorithm 1 and does not change its;

$\Delta\eta^u_{Out}$, $\Delta\eta^u_{In}$ be the variable sets of unassigned components, adjacent to component i, that are connected to component i by his output arcs and input ones respectively. The initial values of the sets $\Delta\eta^u_{Out} = \tilde{\eta}^u \cap Out(i)$, $\Delta\eta^u_{In} = \tilde{\eta}^u \cap In(i)$, $\Delta\eta^u = \Delta\eta^u_{Out} \cup \Delta\eta^u_{In}$;

$\Delta\eta^a_{Out}$, $\Delta\eta^a_{In}$ be the variable sets of components, adjacent to component i and corresponding to output and input arcs of the component, that are assigned by subAlgorithm 2. Initial values of the sets $\Delta\eta^a_{Out} = \varnothing$, $\Delta\eta^a_{In} = \varnothing$, $\Delta\eta^a = \Delta\eta^a_{Out} \cup \Delta\eta^a_{In} = \varnothing$. Always during the performance of subAlgorithm 2, the following relations are satisfied $\Delta\eta^a_{Out} \cup \Delta\eta^u_{Out} = \tilde{\eta}^u \cap Out(i)$, $\Delta\eta^a_{In} \cup \Delta\eta^u_{In} = \tilde{\eta}^u \cap In(i)$;

J be the initial number of arcs adjacent to component i that are not yet assigned, $J = \left|\Delta\eta^u \cup \Delta\eta^a\right|$;

$\Delta\lambda_s$ be the set of arcs $j \in \Delta\eta^a$ assigned by subAlgorithm 1 to communication resource s;

ζ_{Out} be the set of such computers that are connected to computer n by corresponding output channels (n, m) which can be used for placing yet unassigned output arcs of component i;

ζ_{In} be the set analogous with ζ_{Out} but for input channels (m, n) of computer n and input arcs of component i;

Inasmuch as index of component i is fixed and there is only one arc between any pair of adjacent components, we can use for numbering output and input arcs of component i the same indexes of components that are adjacent to component i.

A search-tree construction will be carried out with respect to levels $j = \overline{1, J}$ corresponding to arcs $j \in \Delta\eta^a$. Each vertex of level t is characterized by corresponding set $\vartheta_t = \{ (j_1, k_1), (j_2, k_2), \ldots, (j_t, k_t) \}$ which determines assignments of first t arcs to channels. Here pair (j_t, k_t) denotes that arc j_t is assigned to channel k_t[1] and $\Delta\eta^a = \{j_1, j_2, \ldots, j_t\}$.

Let us derive a lower bound of objective function (6.12) for the vertex of level t. Taking into account that $t < J$ arcs, adjacent to component i, are already assigned, the objective function can be rewritten as

$$^{min}L_{in}\left(\tilde{\eta}^u\right) = \frac{1}{2}\left(L_{\vartheta_t}\left(\Delta\eta^a\right) + L_{\vartheta_t}\left(\Delta\eta^u\right)\right) \quad ,$$

$$L_{\vartheta_t}\left(\Delta\eta^a\right) = \sum_{j \in \Delta\eta^a_{Out}} g_{nc(j)}(d_{ij}) + \sum_{j \in \Delta\eta^a_{In}} g_{c(j)n}(d_{ji}) \qquad , (6.18)$$

where $L_{\vartheta_t}\left(\Delta\eta^a\right)$ determines the cost of assignments of first t arcs $j \in \vartheta_t$ that were already performed. The term $L_{\vartheta_t}\left(\Delta\eta^u\right)$ is a cost value of further placements of arcs $j \in \Delta\eta^u$ that are not yet assigned.

To obtain a lower bound of $L_{\vartheta_t}\left(\Delta\eta^u\right)$ we relax the integer programming problem (6.12) - (6.14) to continuous one. The relaxation allows to place arcs and components adjacent to component i fractionally, i.e. more than one instance of an arc or a component may exist with each instance handling just a fraction of load imposed for the (individual, atomic, not replicated) arc or the component respectively. Let us define new solution variables

- z_{jn}, $j \in \Delta\eta^u$, that determines a part of capacity of arc j absorbed by computer n to which corresponding fraction of component j is assigned, i.e.

$$0 \le z_{jn} \le \frac{min\{\Delta R_n, d_j\}}{d_j} d_{ij} \le d_{ij} \qquad , (6.19)$$

  where ΔR_n is the available resource of computer n;

- z_{jk} $(j \in \Delta\eta^u, k \ne n, k \in \zeta_{Out} \cup \zeta_{In})$ denoting a value of capacity allocated for arc j into channel k. Arc j may obtain required capacity from one or some different channels. Thus

$$0 \le z_{jk} + z_{jn} \le d_{ij}, \ \forall j \in \Delta\eta^u_{Out}, \forall k \in \zeta_{Out} \qquad (6.20)$$

$$\sum_{k \in \zeta_{Out}} z_{jk} + z_{jn} = d_{ij}, \quad \forall j \in \Delta\eta^u_{Out} \qquad (6.21)$$

---

[1] To simplify notation we assume if j_t is an output arc of component i then k_t is the input channel of computer k_t (and the output channel of computer n). Otherwise if j_t is an input arc of component i then k_t is the output channel of computer k_t (and the input channel of computer n).

$$0 \le z_{jk} + z_{jn} \le d_{ji},\ \forall j \in \Delta\eta^{u}_{In},\ \forall k \in \zeta_{In} \tag{6.22}$$

$$\sum_{k \in \zeta_{In}} z_{jk} + z_{jn} = d_{ji},\ \forall j \in \Delta\eta^{u}_{In} \tag{6.23}$$

Now, taking into account formulation of suProblem 2 (6.12) - 6.14) and relations (6.19) - (6.22), the minimization problem for solving lower bound B_{ϑ_t} of $L_{\vartheta_t}\left(\Delta\eta^{u}\right)$ in (6.18) can be formulated as

$$L_{\vartheta_t}\left(\Delta\eta^{u}\right) \ge B_{\vartheta_t}\left(\Delta\eta^{u}\right) = min_{z_{jk}}\left\{ \sum_{k \in (\zeta_{Out} \cup n)} \sum_{j \in \Delta\eta^{u}_{Out}} z_{jk}a_{jk} + \sum_{k \in (\zeta_{In} \cup n)} \sum_{j \in \Delta\eta^{u}_{In}} z_{jk}a_{jk} \right\} \tag{6.24}$$

subject to

$$\sum_{j \in \Delta\eta^{u}} \frac{z_{jn}d_j}{d_{ij}} \le \Delta R_n(\tilde{\eta}_n) - \sum_{\left(\substack{j \in \Delta\eta^{a} \\ (j \therefore c(j) = n)}\right)} d_j = \Delta R^{t}_{n}\left(\Delta\eta^{a}\right) \tag{6.25}$$

$$\sum_{(n,k) \in \pi_s} \sum_{j \in \Delta\eta^{u}_{Out}} z_{jk} + \sum_{(n,k) \in \pi_s} \sum_{j \in \Delta\eta^{u}_{In}} z_{jk} \le \Delta A_s(\tilde{\lambda}_s) - \sum_{(a,b) \in \Delta\lambda_s} d_{ab} = \Delta A^{t}_{s}(\Delta\lambda_s), \tag{6.26}$$

$$\forall s \in \rho_{nk} \cup \rho_{kn},\ \forall k \in \zeta_{Out} \cup \zeta_{In}$$

where

$$a_{jk} = \frac{g_{nk}(d_{ij})}{d_{ij}} \quad \text{if } j \in \Delta\eta^{u}_{Out}, k \in \zeta_{Out},$$

$$a_{jk} = \frac{g_{nk}(d_{ji})}{d_{ji}} \quad \text{if } j \in \Delta\eta^{u}_{In}, k \in \zeta_{In},$$

$$\text{and } a_{jn} = 0 \text{ , since } g_{nn} = 0 \tag{6.27}$$

Here $z_{jk}a_{jk}$ is the cost of capacity $z_{jk} \le d_{ij}$ allocated for arc (i, j) in channel (n, k) if $k \in \Delta\eta^{u}_{Out}$, and in channel (k, n) if $k \in \Delta\eta^{u}_{In}$. If component $j \in \Delta\eta^{u}$ is placed into the same computer n already containing component i then arc connecting components i and j does not need a communication resource and therefore $a_{jn} = 0$.

Problem (6.24) - (6.26) can be solved, for example, by the simplex method. However we present a more simple and economical computation approach.

Let us order output channels $(n, k), k \in \zeta_{Out}$ and input ones $(k, n), k \in \zeta_{In}$ so that $g_{nk}(D) \le g_{n(k+l)}(D), l \ge 1$, and $g_{kn}(D) \le g_{(k+m)n}(D), m \ge 1$.

For every arc j adjacent to component i the row of cost values g_{nk} and g_{kn} of assignments to corresponding channels are determined. (Note that $g_{nn} = 0$). If arc j can not be assigned, for

example to channel (n, k), then we set $g_{nk} = \infty$ and do not take into account this channel, when arc j is considered.

Now the solution of problem (6.24) - (6.26) can be obtained by sequential filling

- at first, computer n by unassigned components j, adjacent to component i and ordered (in accordance with their arcs) in decreasing with respect to a_{jk}, till relation (6.25) is satisfied.
- then channels, ordered in increasing of cost functions g_{nk} and g_{kn}, by unassigned arcs adjacent to component i and ordered as mentioned above.

Let us present the algorithm for the problem (6.12) - (6.14) solution.

1. Ordering the output and input channels of computer n in increasing with respect to g_{nk} and g_{kn}, as mentioned above. Ordering the output and input arcs of component i in decreasing with respect to a_{jk}.
2. $t = 0, \vartheta_t = \varnothing$.
3. Constructing the next level of the search tree (by considering all possible placements of the next unassigned arc j_t adjacent to component i)

   $t \Leftarrow t + 1$;
4. $K_t > 0$ terminal vertices will be built at level t of the search-tree. (Here K_t denotes a number of acceptable assignments of arc j_t including the case of absorption the arc by computer n). For every such vertex corresponding to assignment (j_t, m) considered at level t, the bounding value is computed

   $$P_{\vartheta_t}(m) = L_{\vartheta_t}\left(\Delta\eta^a \cup j_t\right) + B_{\vartheta_t}\left(\Delta\eta^u \cup j_t\right) \quad , (6.28)$$

   where the first term is calculated by formula (6.18). For computing B_{ϑ_t}, the problem (6.24) - (6.26) is solved by sequential filling at first computer n then channels in the order as mentioned above by components and arcs respectively.
5. Choosing the *terminal* vertex of level t that has the lowest bounding value $\tilde{P}_{\vartheta_t} = P_{\vartheta_t}(k_t) = min_m \{P_{\vartheta_t}(m)\}$. Here chosen vertex corresponds to placement of arc j_t into channel (computer) k_t
6. **If** $\tilde{P}_{\vartheta_t} = \infty$ **then**

   return to the previous level $t \Leftarrow t - 1$ of the search tree resetting state parameters of assignments corresponding to this level. **If** $t > 0$ **then** go to step 5 **else** the original problem (6.12) - (6.14) has not an acceptable solution.
7. Assigning arc j_t to channel (computer) k_t.
8. **If** all arcs are assigned, i.e. $t = J$ **then**

   a. The vertex with value $\tilde{P}_{\vartheta_J} = \sum_{\substack{j \in \vartheta_J \\ j \in \Delta\eta^a_{Out}}} g_{nc(j)}(d_{ij}) + \sum_{\substack{j \in \vartheta_J \\ j \in \Delta\eta^a_{In}}} g_{c(j)n}(d_{ji})$ determines

an acceptable solution of problem (6.12) - (6.14). If another acceptable solution was obtained before, then choosing the best one from these both and restoring it's cost as L_{best}

b. Vertices for which

$$P_{\vartheta_t} \geq L_{best},\ t = \overline{1, J-1} \tag{6.29}$$

are not promising because they do not contain the optimal solution and, therefore, they are eliminated from further consideration.

c. **If** relation (6.29) is satisfied for all terminal vertices

then found value L_{best} is optimal and, therefore, $^{min}L_{in} = \frac{1}{2}L_{best}$

else the terminal vertices showing promise are branched and the search-tree is

constructed further (by choosing the *terminal* vertex that is nearest one to the obtained solution and has a bounding value less than L_{best})[1] till relation (6.29) will be right for all remaining terminal vertices. Go to step 4.

9. **If** $t < J$ **then** the vertex of level t with lowest bounding value $\tilde{P}_{\vartheta_t}$ will be branched with respect to the variable j_{t+1}, go to step 3.

At **step 2** and **step 4** computing bounding values $P_{\vartheta_t}(m)$ is performed:

a. Computing bounding value $P_{\vartheta_t}(n)$ for the case of absorption of arc j_t by computer n:

if component n has enough resource to place adjacent component j_t (in other words, to absorb arc j_t), i.e. $\Delta R_n \geq d_{jt}$

then

'virtual' assigning component j_t to computer n (that means absorption of arc j_t by computer n);

best 'virtual' placing yet unassigned arcs $j \in \Delta\eta^u$ adjacent to component i by sequential filling, at first computer n by absorbing the arcs, and then ordered channel, adjacent to the computer, by ordered arcs;

computing bounding value $P_{\vartheta_t}(n)$

else $P_{\vartheta_t}(n) = \infty$;

b. Computing bounding values $P_{\vartheta_t}(m)$ of placements of output arc j_t into every output channel (n, m):

if arc j_t is output one with respect to component i

then

[1] All parameters of the vertex have to be reset.

for every output channel (n, m) of computer n, i.e. for every possible assignment of arc j_t

if channel (n, m) has enough resource to place arc j_t, i.e. $\Delta R_{nm} \geq d_{ij_t}$

then

'virtual' assigning arc j_t to channel (n, m)

best 'virtual' placing yet unassigned arcs $j \in \Delta\eta^u$ adjacent to component i by sequential filling, at first computer n by absorbing the arcs, and then ordered channel, adjacent to the computer, by ordered arcs;

computing bounding value $P_{\vartheta_t}(m)$

else $P_{\vartheta_t}(m) = \infty$;

c. Computing bounding values $P_{\vartheta_t}(m)$ of placements of input arc j_t into every input channel (m, n) :

if arc j_t is input one with respect to component i

then

for every input channel (m, n) of computer n, i.e. for every possible assignment of arc j_t

if channel (m, n) has enough resource to place arc j_t, i.e. $\Delta R_{mn} \geq d_{j_t i}$

then

'virtual' assigning arc j_t to channel (m, n)

best 'virtual' placing yet unassigned arcs $j \in \Delta\eta^u$ adjacent to component i by sequential filling, at first computer n by absorbing the arcs, and then ordered channel, adjacent to the computer, by ordered arcs;

computing bounding value $P_{\vartheta_t}(m)$

else $P_{\vartheta_t}(m) = \infty$

At **step 7** assignment of arc j_t to channel (or computer) k_t is as follows:

$\Delta\eta^a \Leftarrow \Delta\eta^a \cup j_t$, $\Delta\eta^u \Leftarrow \Delta\eta^u \backslash j_t$, $\vartheta_t \Leftarrow \vartheta_t \cup (j_t, k_t)$;

if arc j_t is absorbed by computer n, i.e. $k_t = n$

then $\Delta\eta_n \Leftarrow \Delta\eta_n \cup j_t$; $\Delta R_n \Leftarrow \Delta R_n - d_{j_t}$

else

if arc j_t is output one of component i

then for every shared resource $s \in \rho_{nk_t}$

$\Delta\lambda_s \Leftarrow \Delta\lambda_s \cup (i, j_t)$; $\Delta A_s^t \Leftarrow \Delta A_s^t - d_{ij_t}$; $L_{\vartheta_t} \Leftarrow L_{\vartheta_t} + g_s(d_{ij_t})$;

Recomputing available capacity of output channel (n, k_t) :

$$\Delta R_{nk_t} = min\{\Delta A_s, \forall s \in \rho_{nk_t}\} \quad ;$$

if arc j_t is input one of component i

then for every shared resource $s \in \rho_{k_t n}$

$$\Delta\lambda_s \Leftarrow \Delta\lambda_s \cup (j_t, i)\ ;\ \Delta A_s^t \Leftarrow \Delta A_s^t - d_{j_t i};\ L_{\vartheta_t} \Leftarrow L_{\vartheta_t} + g_s(d_{j_t i})$$

Recomputing available capacity of output channel (k_t, n) :

$$\Delta R_{k_t n} = min\{\Delta A_s, \forall s \in \rho_{k_t n}\} \quad .$$

6.3.3 Example using SubAlgorithm 2

Let us consider an example of locating arcs of component *d* (see Figure 2.1). Assume that component *d* is assigned to computer *C* (see Figure 2.9) and no other components are yet assigned. In Figure 6.3 the part of DMA graph, that have to be assigned, and the part of DCS graph, used for the assignment, are depicted.

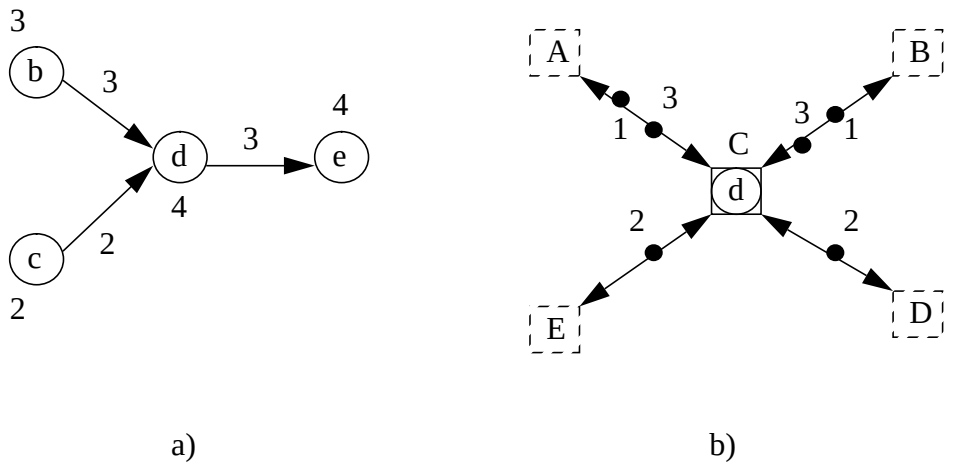


Figure 6.3,a) Component d of DMA graph with adjacent arcs and components, b) Computer C of DCS graph with adjacent channels

Table 6.1 presents initial data of components, arcs and channels ordered in accordance with step 1 of subAlgorithm 2. (The initial data are chosen to present more general and interesting case).

Every arc adjacent to component *d* is presented in the table by two columns: the first one for a component adjacent to component *d* and the second one for the arc connecting the component to component *d*.

Cells of the last row contain values of required resources of components and arcs adjacent to component *d*. Last four columns are used for values of available DCS resources: column *R* - for

resource of computer *C* and for capacities of virtual channels, columns A_1, A_2, A_3 - for capacities of communication resources over which corresponding channels are routed.

Cost values a_{jk} computed by formula (6.27) and used for ordering arcs and channels are presented in columns of components and arcs. The filling in the table by a_{jk} takes into account that input arcs of component *d* can be placed only into input channels of computer *C* and output arcs - into output channels. Rows of the table follow the order: the row for computer *C*, the rows for input channels of computer *C* and the rows for output channels.

The second row of the table corresponds to computer *C* which component *d* is assigned to. The computer can absorb (completely or partly) an arc adjacent to component *d.* A partly absorption of an arc in accord with formula (6.19) is acceptable provided that at least one not yet assigned component adjacent to component *d* can be previously completely allocated into computer *C*[1].

Table 6.1

	b	(b,d)	e	(d,e)	c	(c,d)	R	A1	A2	A3
C	0		0		0		2			
(D,C)		4				2	5		5	
(E,C)		4				2	5		5	
(A,C)		5				3	3	4		3
(B,C)		5				3	3	4		3
(C,D)				3			5		5	
(C,E)				3			5		5	
(C,A)				4			3	4		3
(C,B)				4			3	4		3
d	3	3	4	3	2	2				

Arc allocation are executed by filling in the table from link to right and from top to down directions. An illustration of arc *(b,d)* allocation will make this clear.

The search-tree of possible alternatives is depicted in Figure 6.4. The vertex *C* of the first level gets bounding value ∞ since computer *C* has not enough resource to locate component *b* and to absorb arc *(b,d)*. Let us consider the next vertex *(D,C)* of the first level. The allocation of arc *(b,d)* into channel *(D,C)* causes decrease (by $d_{bd} = 3$) of capacity of:

- channel *(D,C)*,
- all communication resources over which channel *(D,C)* is routed,
- all other channels that are routed over the communication resources, capacity of which were decreased.

[1] This condition allows to get more exact bounding value.

Table 6.2 represents the resource state of DCS after assignment of arc *(b,d)* to channel *(D,C)*. In the cell corresponding to allocation arc *(b,d)* into channel *(D,C)*, value 3 shows that arc *(b,d)* gets completely required capacity $d_{bd} = 3$ by placing into channel *(D,C)*. In column *R* and *A2*, capacities of channel *(D,C)*, communication resource 2 involved into the channel, and other channels that are routed over communication resource 2 are decreased from 5 to 2 by $d_{bd} = 3$.

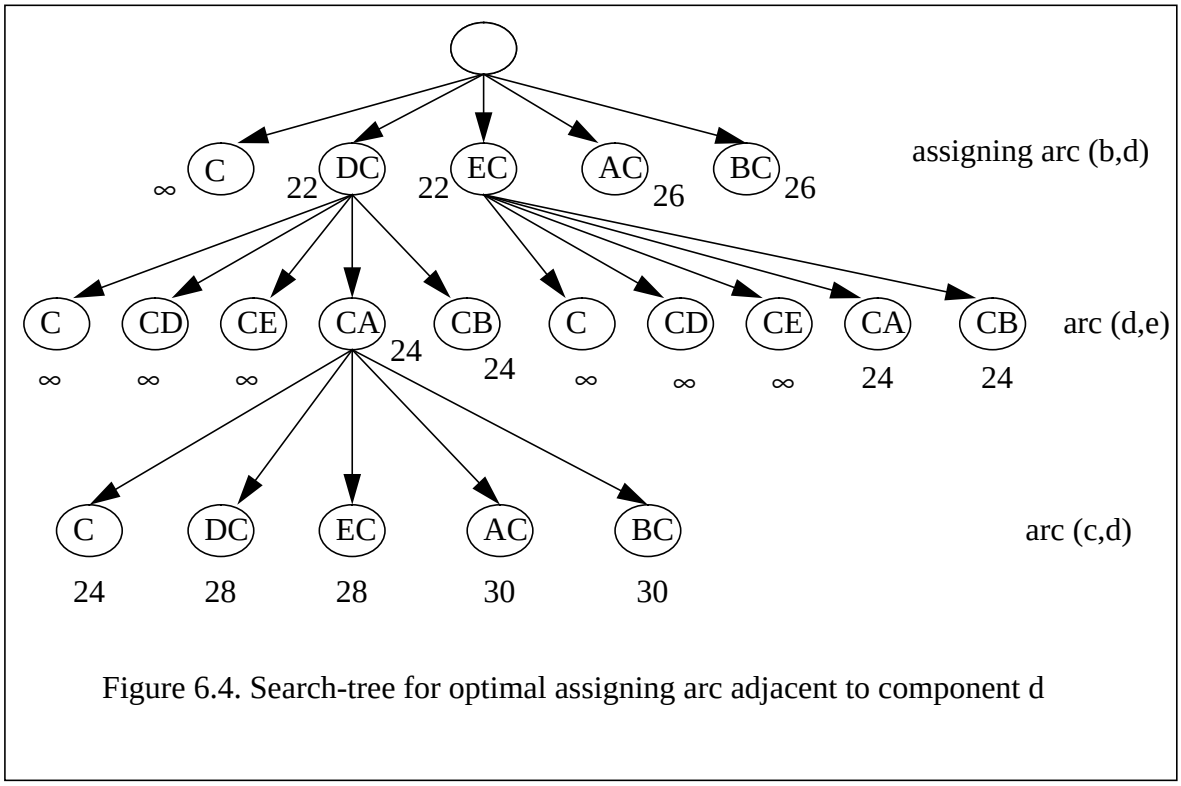


Figure 6.4. Search-tree for optimal assigning arc adjacent to component d

Table 6.2

	b	(b,d)	e	(d,e)	c	(c,d)	R	A1	A2	A3
C	0		0		0		2			
(D,C)		4 3				2	2 ~~5~~		2 ~~5~~	
(E,C)		4				2	2 ~~5~~		2 ~~5~~	
(A,C)		5				3	3	4		3
(B,C)		5				3	3	4		3
(C,D)				3			2 ~~5~~		2 ~~5~~	
(C,E)				3			2 ~~5~~		2 ~~5~~	
(C,A)				4			3	4		3
(C,B)				4			3	4		3
d	3	3	4	3	2	2				

In accordance with the table, next must be assigned arc *(d,e)*. First one attempts to locate component *e* into computer *C*. However available resource of computer *C* $\Delta R_C = 2$ is not enough to locate component *e* completely. To locate component *e* partly, first one must to locate at least one not yet assigned component into computer *C* completely. Next component *c* in the table is such one. After allocation of component *c* into computer *C*, available resource of computer *C* decrease to $\Delta R_C = 0$. Therefore arc *(e,d)* now can not be absorbed by computer *C* and it must be allocated into an input channel of computer *C*. One uses the following rule: locate an arc into the cheapest acceptable channel. Following the rule, arc *(e,d)* is located partly: 2 units of required capacity to channel *(C,D)* and 1 unit - to channel *(C,A)*. Table 6.3 represents the resource state of DCS after these assignments and allows to compute the bounding value of the vertex *(D,C)* by formula (6.24): $P_{DC} = \sum_{j,k} a_{jk} z_{jk} = 0 \cdot 2 + 4 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 = 22$.

Table 6.3

	b	(b,d)	e	(d,e)	c	(c,d)	R	A1	A2	A3
C	0		0		0 2		0 ~~2~~			
(D,C)		4 3				2	0 ~~2~~ ~~5~~		0 ~~2~~ ~~5~~	
(E,C)		4				2	0 ~~2~~ ~~5~~		0 ~~2~~ ~~5~~	
(A,C)		5				3	3	4		3
(B,C)		5				3	3	4		3
(C,D)				3 2			0 ~~2~~ ~~5~~		0 ~~2~~ ~~5~~	
(C,E)				3			0 ~~2~~ ~~5~~		0 ~~2~~ ~~5~~	
(C,A)				4 1			2 ~~3~~	3 ~~4~~		2 ~~3~~
(C,B)				4			2 ~~3~~	3 ~~4~~		2 ~~3~~
d	3	3	4	3	2	2				

Using the proposed procedure of filling in the table 6.1 for other vertices of the first level, one obtains bounding values for the vertices. Choosing the vertex *(D,C)* with minimum value, one continues the branching search-tree. At the third level one gets the first acceptable solution: arc *(b,d)* assigned to channel *(D,C)*, arc *(d,e)* - to channel *(C,A)* and arc *(c,d)* is absorbed by computer *C*. The solution has value of objective function *L = 24*.

Only one vertex *(E,C)* of the first level has value 22 less than *L*. Therefore one executes branching the vertex. However new bounding values, obtained at the second level, are not less than *L*. Thus the first solution obtained is optimal.

6.4 Roll-back Procedure

The Roll-back Procedure is as follows:

1. Storing the reached 'not full' DMA location in set Ψ that can be used further by Reduction Procedure if it will be performed.
2. Rolling back to the nearest level of the search-tree.
3. **If** the level is highest one, i.e. indexed by 0 and consists of a single root-vertex

   then there is no solution of Original Mapping Problem for given data, go to Reduction Procedure

   else the acceptable solution obtained is optimal, stop
4. **If** at the level there is at least one acceptable vertex (with bounding cost value $\hat{B} < Fbest$) not yet examined

   then choosing the best one, resetting state-parameters of the vertex ($\eta^a, \eta^u, \eta_n, \lambda^a, \lambda^u, \lambda_{nm}, \lambda_s, \Delta R_n, \Delta R_{nm}, \Delta A_s, B_{in}$), and go to step 7of the Forward Procedure

   else go to step 2

6.5 Penalty Algorithm

6.5.1 Criteria and method of choosing next assignment

The structure of the Penalty Algorithm is analogous with one of the Ord-Algorithm presented in Figure 6. However the content of the Forward Procedure distinguishes from the one of the Ord-Algorithm and will be considered below.

Let us examine the Forward Procedure.

A criteria of choosing next assignment (i, n) of component i to computer n based on cost matrix $\{S_{in}, i \in \eta^u, n \in \zeta(\eta^u)\}$ and suggested by Vogel's Approximation Method [4]. Element S_{in} computed by formula (6.9) determines the minimum attainable cost associated with assigning component i to computer n. The Vogel's Method, used for solving transportation problems, falls into a class of so-called penalty methods. It is based on the 'difference' associated with each row and column in the matrix $\{S_{in}\}$. A row or column 'difference' is here defined as the arithmetic difference between the second smallest and the smallest element in that row or column. This quantity provides a measure of priorities for making allocations to the respective rows and columns since it indicates the minimum unit penalty incurred by failing to make the assignment to the smallest cost cell in that row or column. The selection of an element

S_{in} is then made by selecting the row or column with the largest difference and choosing the smallest cost cell in that row or column.

Thus determination the assignment by the penalty method is as follows.

1. Computing the row difference by finding the arithmetical difference between the second smallest and the smallest element in each row of matrix $\{S_{in}\}$.
2. Computing the column differences in the same way.
3. Finding the largest of the row and column differences and then the smallest cost cell in that row or column. Suppose element S_{in} has been located in this way. Then assignment (i, n) will be done next.

6.5.2 Forward Procedure

Let us present the Forward Procedure.

1. Checking and executing 'non-alternative' assignments similarly to step 1 of Forward Procedure of Ord-Algorithm (see section 6.3).
2. Computing initial value of bounding function similarly to step 3 of Forward Procedure of Ord-Algorithm.
3. Checking and executing 'non-alternative' assignments similarly to step 4 of Forward Procedure of Ord-Algorithm.
4. Computing the bounding cost matrix $\{S_{in}, i \in \eta^u, n \in \zeta(\eta^u)\}$ by formula (6.9) using the subAlgorithm 1 for every component $i \in \eta^u$.
5. **If** there is at least one component yet unassigned that can not be assigned **then** go to Roll-back procedure.
6. Applying the penalty method to determine assignment (i, n) of component i to computer n that will be done next.
7. Assigning component i to computer n. Increasing cost F in the value of expenses of such assignment.
8. If all components are assigned then checking whether the solution obtained is optimal similarly to step 10 of Forward Procedure of Ord-Algorithm.
9. **If** the current value of cost F does not exceed the cost $Fbest$ of the best solution found previously **then** go to step 3 **else** go to Roll-back Procedure.

7 Reduction Procedure

The procedure obtains the DMA placement into DCS with minimum relaxation of resource requirements.

7.1 Problem Formulation

Let

$D_n(\eta_n) = \sum_{i \in \eta_n} d_i$ be the summary resource of computer n used by components η_n assigned to the computer;

$D_{nm}(\lambda_{nm}) = \sum_{(i,j) \in \lambda_{nm}} d_{ij}$ be the summary capacity of channel (n, m) used by DMA arcs λ_{nm} assigned to the channel.

Let us consider ratio $\delta_n = D_n / R_n$ which means a fraction of computer resource R_n used by all components assigned to computer n. If $\delta_n \le 1$ then location of the components can be done onto computer n without resource requirement relaxation. Otherwise, $\delta_n - 1 > 0$ is the measure of resource gap of computer n. If we have a procedure for a distribution of the resource gap between all components assigned to the computer[1], then we can get the resource gap δ_{in} of the computer with respect to individual component i. Value δ_{in} can be used further for evaluation of QoS relaxation.

The communication resource requirement relaxation can be determined similarly by utilization factor $\varepsilon_s = \sum_{(n,m) \in \pi_s} D_{nm}(\lambda_{nm}) / A_s$ for every DCS communication resource $s \in \rho$.

Following mapping problem formulation allows to get an acceptable DMA location if DCS resources are enough to meet resource requirements of DMA, otherwise to get an optimal DMA location that guarantees minimum resource requirement relaxation with respect to computers and communication resources of DCS:

$$G(\bar{x}) = min\left\{ max\left\{ \left(\frac{\sum_{i \in \eta^u} d_i x_{in}}{R_n}, \forall n \in \zeta \right), \left(\frac{\sum_{(n,m) \in \pi_s} \sum_{(i,j) \in \lambda} d_{ij} x_{in} x_{jm}}{A_s}, \forall s \in \rho \right) \right\} \right\} \quad (7.1)$$

subject to

$$\sum_{n \in \zeta_i(\eta^u)} x_{in} = 1, \forall i \in \eta \quad (7.2)$$

[1] This is another optimization problem does not examined here.

7.2 Problem Solution Technique

An algorithm proposed is based on a branch-bound method that is analogous with one of Optimization Procedure (see section 6.1).

Suppose that for any terminal vertex of the search-tree, $t = |\eta^a| < |\eta|$ component assignments η_n, $n \in \zeta$, have already been made, $\bigcup_{n \in \zeta} \eta_n = \eta^a$. The DMA arc assignments to DCS channels λ_{nm}, $(n, m) \in \pi$, $\bigcup_{(n, m) \in \pi} \lambda_{(n, m)} = \lambda^a$, depends in a unique fashion on the component assignments. On the other hand, having the arc assignments to channels one can get corresponding arc assignments to DCS communication resources $\lambda_s = \bigcup_{(n, m) \in \pi_s} \lambda_{nm}$.

To take into account different features of component and arc locations some lower bounds of objective function (7.1) are proposed.

Evidently, maximum utilization of DCS computers produced by assigned components can be used as one of bounds:

$$G_1 = max\{\delta_n(\eta_n), n \in \zeta\} \tag{7.3}$$

For computing lower bound for a terminal vertex of the search tree, best placements of not yet assigned components are sought. Suppose that placement of such component *k* to computer *l* is examined. Then the conditional utilization of the computer will be

$$G_2^{kl} = \delta_l + d_k / R_l.$$

The placement of component *k* calls for locations of his adjacent arcs that connect the component to already assigned components. These arc locations onto channels adjacent to computer *l* can cause an utilization change of DCS communication resources over which the channels are routed. Let G_3^{in} be maximum utilization of such communication resources provided that component *k* is placed into computer *l*, i.e.

$$G_3^{kl} = max_s\{\varepsilon_s / k \to l\},$$

where $\varepsilon_s = \sum_{(n, m) \in \pi_s} D_{nm}(\lambda_{nm}) / A_s$ and *(n,m)* are the channels adjacent to computer *l*.

Therefore maximum utilization of DCS resources obtained provided that component *k* is placed into computer *l* is

$$G_{23}^{kl} = max\{G_2^{kl}, G_3^{ikl}\}.$$

The best placement of component *k* is characterized by utilization factor

$$G_{23}^{k} = min_l\{G_{23}^{kl}\}.$$

Thus the other bound of objective function can be maximum utilization reached by best placement of one of not yet assigned components:

$$G_{23} = max_k \{ G_{23}^k \} \tag{7.4}$$

If there is a gap of computational resources in the DCS, next third bound of objective function is useful too.

Let us order computers $(n_1, n_2, \ldots, n_{|\zeta|})$ in decreasing with respect to computational resources available to mapped DMA:

$$\Delta R_{n_1} \geq \Delta R_{n_2} \geq \ldots \geq \Delta R_{n_z} \geq \ldots \geq \Delta R_{n_{|\zeta|}} \tag{7.5}$$

where $\Delta R_{n_z} = R_{n_z} - D_{n_z}$,

If available resources of two (or more) computers are equal, then these computers are ordered in increasing with respect to utilization factors

$$\delta_{n_1} \leq \delta_{n_2} \leq \ldots \leq \delta_{n_z} \leq \ldots \leq \delta_{n_{|\zeta|}} \tag{7.6}$$

Assume that not yet assigned components η^u will be distributed between computers with most available resources such that the utilization of these computers would be uniform. Indexes of these computers are the first $z = min\{|\eta^u|, |\zeta|\}$ elements of the sequence $(n_1, n_2, \ldots, n_z, \ldots, n_{|\zeta|})$ ordered corresponding to relation (7.5) and (7.6). Then the average utilization factor of each such computer is

$$G_4 = \frac{1}{z}\left(\sum_{k=1}^{z} \delta_{n_k} + \frac{\sum_{i \in \eta^u} d_i}{\sum_{k=1}^{z} R_{n_k}} \right) \tag{7.7}$$

Evidently, objective function $G \geq G_4$.

Thus a lower bound of the objective function (7.1) for the vertex of performed assignments $\eta_n, n \in \zeta$ can be computed as follows:

$$BG(\eta_n, n \in \zeta) = max\{G_1, G_{23}, G_4\} \leq G \tag{7.8}$$

Efficiency of the bound proposed depends on the order of not yet assigned components used for construction of next level of the search-tree, i.e. the method of choosing next component relative to which the branching of the chosen vertex has to be done is important. We propose to choose such component using following criterions:

a. component with most number of adjacent arcs that connect, the component to already assigned components (this property is important for the bound G_{23}),

b. if there are some components that have equal most numbers of such adjacent arcs, then one chooses the component of these ones that requires most computational resource (this property is important for G_1 and G_4),

c. if there is no component with such adjacent arcs, then one chooses the component with most required resource.

7.3 Algorithm

The algorithm of the Reduction Procedure is as follows:

1. Computing initial lower bound GI_0 using formulas (7.4), (7.7), (7.8) for η^a and η_n, $n \in \zeta$ including only attached components that are already placed by the Attached Component Location Procedure. Let $G_{best} = \infty$.
2. **If** not all components are assigned, i.e. $|\eta^u| > 0$

   then choosing next component $i \in \eta^u$ using criterions mentioned above

   else go to step 6
3. Constructing terminal vertices at level $l = |\eta^a| + 1$ of the search-tree. Each of such vertices corresponds to one computer n that component i could be assigned to.

   For every terminal vertex, bounding value $BG_n(\eta^a \cup i)$ are computed using formula (7.3) (7.4), (7.7), (7.8) provided that component i is placed into corresponding computer n
4. Choosing the terminal vertex of level l that has the lowest bounding value. Suppose, it is the vertex with value BG_n. **If** $BG_n > G_{best}$ **then** go to step 7.
5. Including the vertex into the current solution. Assigning component i to computer n. Go to step 2.
6. An acceptable solution is obtained. Suppose, value of the solution is BG.

   If $BG = BG_0$ **then** the solution is optimal, stop.

   If $BG < G_{best}$ **then** storing the solution and let $G_{best} = BG$.
7. Excluding the last vertex from the solution.

   Rolling back to the previous level l.

   If $l = 0$

   then stop, optimal the solution is obtained

   else restoring vertices of level l and go to step 4.

Remark. Reduction Procedure can use set Ψ of 'not full' solutions of DMA that are storing by Roll-back Procedure (see section 6.4). Then before step 2, one has to choose the best 'not full' solution of set Ψ and to restore the state parameters η^a, η^u, η_n, λ^a, λ_s of the solution as initial ones. The best 'not full' solution could be a solution with maximum number of assigned components, $max_{\Psi}\{|\eta^a|\}$. The proposed approach allows to obtain quickly an acceptable solution, value of which can be used further to improve the solution.

The obtained optimal solution could be used further in the following ways:

- Distributing the resource gap in every overflowed computer and channel between components and arcs assigned to them respectively.
- Checking the reduction level of QoS caused by the DCS resource gap.
- **If** the reduction level of QoS is held within allowable limits vertices **then** an acceptable relaxed solution is obtained, stop.

8 Examples

8.1 Example 1

Let us consider an example of mapping DMA depicted in Figure 2.1 to DCS depicted in Figure 2.7, system graph of which is presented in Figure 2.9. Required computational resources of components and capacities for component communications are shown in Figure 2.1 as weights of nodes and links of DMA graph respectively. Available capacities of DCS communication resources LAN1, LAN2 and WAN are $A_1 = 5, A_2 = 6, A_3 = 3$ respectively. Initial data of available capacities of virtual channels and components of DCS are presented by matrix (2.3).

Let us determine cost functions. To simplify computations, we consider linear cost functions of capacity and computational resource. Suppose, that costs of one unit of the computational resource for computers of DCS are following: $\alpha_A = 2, \alpha_B = 1, \alpha_C = 2, \alpha_D = 2, \alpha_E = 1$. Costs of one unit of capacity for LAN1 is $\alpha_1 = 2$, for LAN2 - $\alpha_2 = 1$ and for WAN - $\alpha_3 = 3$. Then one can calculate cost factor for every channel (n, m) of DCS by formula $\alpha_{nm} = \sum_{i \in \rho_{nm}} \alpha_i$, where ρ_{nm} is the set of communication resources of DCS over which channel (n, m) is routed. So the final cost factors for computers and virtual channels of DCS can be represented by matrix α:

	A	B	C	D	E
A	2	2	5	6	6
B	2	1	5	6	6
C	5	5	2	4	4
D	6	6	4	2	1
E	6	6	4	1	1

Assume, that components f and g are attached to computers E and D respectively. Now all input data are given.

In accord with the procedure for location of attached components (see section 5), one gets:

- $x_{fE} = 1, x_{fn} = 0, n = A, B, C, D$; $x_{gD} = 1, x_{gn} = 0, n = A, B, C, E$;
- constant term of cost objective function $F_0 = \alpha_E \cdot d_f + \alpha_D \cdot d_g = 1 \cdot 3 + 2 \cdot 3 = 9$;
- modified matrix of required resources *d:*

	a	b	c	d	e	f	g
a	1				1		
b		3		3			
c			2	2			
d				4	3		
e					4	3	3
f						0	
g							0

- modified matrix of available resources *R*:

	A	B	C	D	E
A	5	5	3	3	3
B	5	4	3	3	3
C	3	3	6	6	6
D	3	3	6	5	6
E	3	3	6	6	4

Further computations are executed in accord with Optimization Procedure (see section 6).

At step 1 of Forward Procedure, ‘non-alternative’ assignments are not found.

At step 2, ordering components, using for example cost function $f_E(d_i) + \sum_j [g_E(d_{ij}) + g_E(d_{ji})]$, results in the sequence of components *e, d, b, c, a* with cost

values 14, 12, 6, 4, 2 respectively. This component sequence is used further to construct levels of the search-tree depicted in Figure 8.1.

At step 3 for the vertex of level 0, one computes initial bounding value B_0 of best independent placements of all unassigned components $\eta^u = \{e, d, b, c, a\}$ using subAlgorithm 1.

In Figure 8.2, bounding values computed by subAlgorithms 1 and 2 are presented for all possible placements of every unassigned component. The value in brackets corresponds to term M_{in} in formula (6.4), i.e. to cost of placements of component *i* into computer *n* and arcs that are adjacent to the component and connect it to already assigned components. For example, bounding value for assignment of component *e* to computer *C* is equal to 39.5, the part of which 32 corresponds to summary cost of placement of component *e* to computer *C* and placements of arcs *(e,f)* and *(e,g)* connecting component *e* to already assigned components *f* and *g* (see Figure 2.1). Therefore 39.5 - 32 = 7.5 is the bounding value of L_{in} in formula (6.5) that is the half of the cost of placements yet unassigned adjacent arcs *(a,e)* and *(b,e)*.

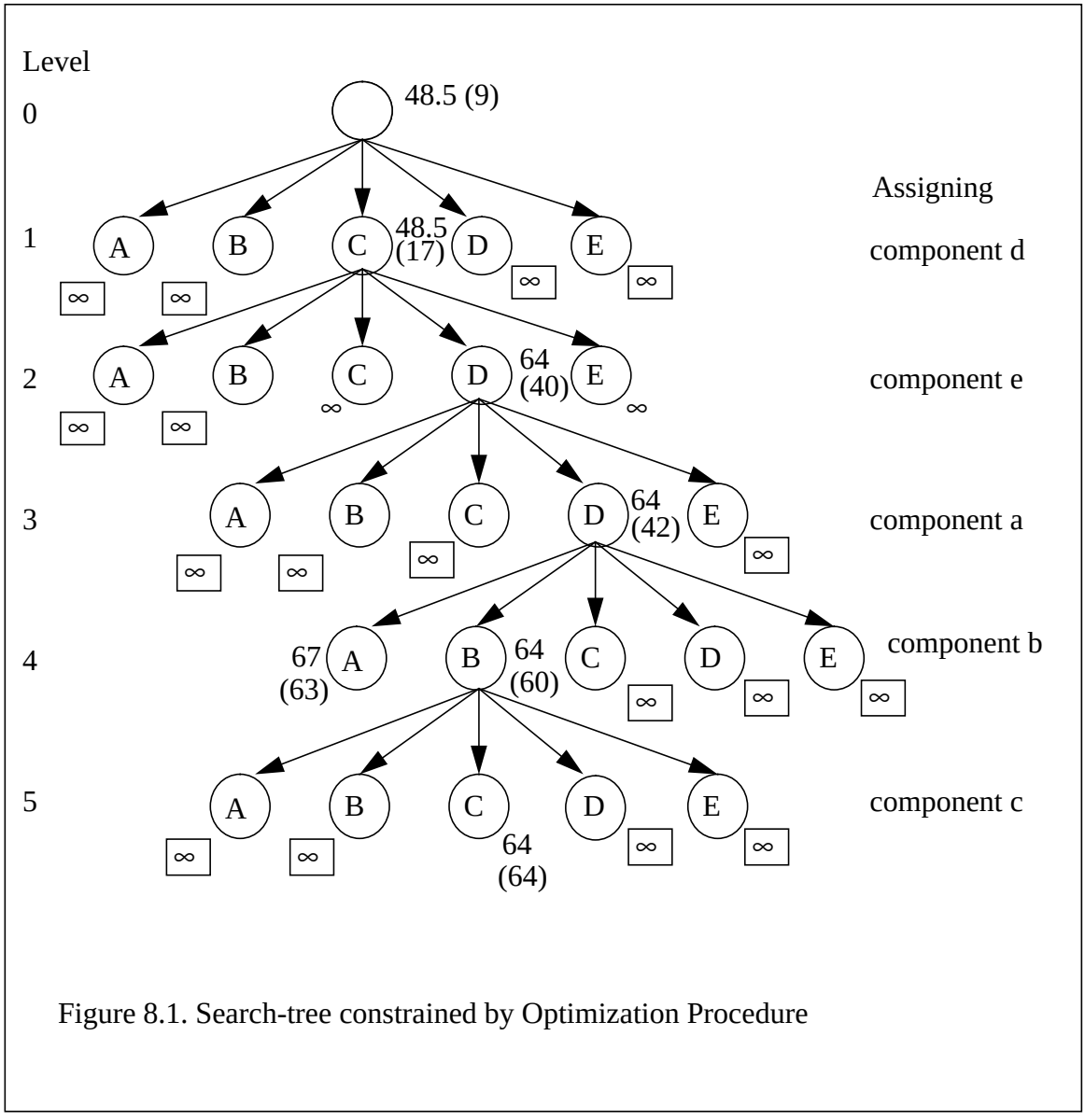


Figure 8.1. Search-tree constrained by Optimization Procedure

Choosing the best placement of every unassigned component that is characterized by minimum cost value, one obtains the initial bounding value $B_0 = F_0 + 10.5 + 20 + 4.5 + 3 + 1.5 = 48.5$. In Figure 8.1, value of already assigned components *f* and *g* $F_0 = 9$ is shown in brackets near the value 48.5. In the process of B_0 computation, one obtains cost bounding values matrix *M* of all possible placements of every isolated unassigned component:

	A	B	C	D	E
a	2	2	2	2	1.5
b	9	6	12	7.5	4.5
c	6	4	4	5	3
d	∞	∞	20	∞	∞
e	∞	∞	39.5	12.5	10.5

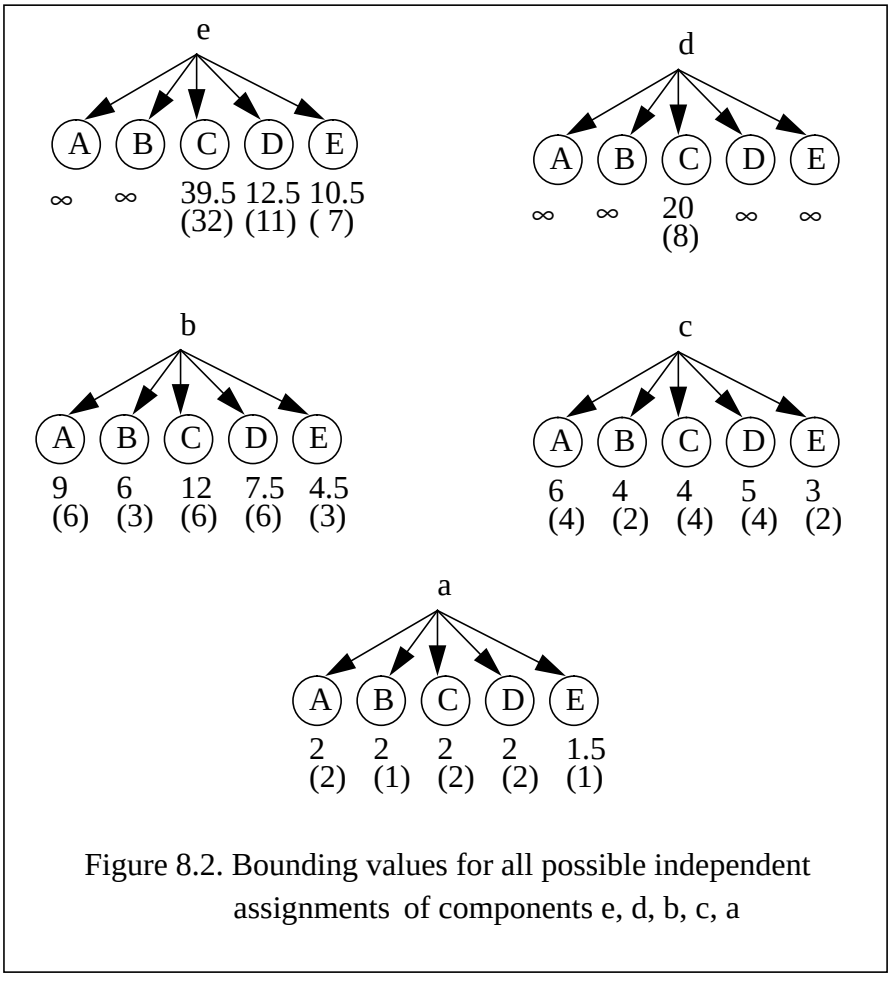


Figure 8.2. Bounding values for all possible independent assignments of components e, d, b, c, a

The matrix contains lowest bounding values for every possible placements of every component and shows:

- component *d* can not be placed into computers *A,B,D,E*;
- component *e* can not be placed into computers *A* and *B*;
- component *d* can be placed only into one computer that is *C*.

This information allows not to compute bounding values for vertices of next levels corresponding to such placements of components *d* and *e*.

In accord with step 4 of Forward Procedure, 'non-alternative' assignment of component *d* to computer *C* is considered at the next first level of the tree (see Figure 8.1). The bounding value of the vertex *C* is equal to 48.5 and value in brackets 17 shows the cost of already assigned components and arcs. Other vertices obtain value ∞ in accord with matrix *M* without additional computations. (In Figure 8.1 these values ∞ are enclosed in rectangles). After this assignment, computational resource of computer *C* is decreased to $\Delta R_C = 2$.

At the second level of the tree, possible assignments of component *e* are examined. In accord with bounding matrix *M* vertices *A* and *B* get value ∞. Computations by subAlgorithm 2 (see section 6.3.2 and 6.3.3) result in value ∞ for vertices *C* and *E*. Really, assignment of component *e* to computer *C* is impossible because available resource of the computer $\Delta R_C = 2$ is not enough to meet resource requirement $d_e = 4$ of component *e*. Assignment of the component to computer *E* is not acceptable too because, after this assignment, available capacity of communication resource $\Delta A_2 = 0$ and then arc *(a,e)* of component *a* can not be placed.

SubAlgorithm 2 obtains value 64 for vertex *D* of second level. Value 40 in brackets corresponds to the cost of already assigned components *f, g, d, e* and their arcs *(e,f), (e,d), (d,e)* connecting the components with each other.

At the step 7 of Forward Procedure the vertex *D* of second level is chosen for further branching. In the process of computing bounding value of the vertex *D* (at step 6), rows of cost bounding matrix *M* that correspond to not yet assigned components *a,b,c* are modified:

	A	B	C	D	E
a	∞	∞	∞	2	∞
b	14	12	∞	∞	∞
c	21	18	∞	∞	∞

Following Forward Procedure, one returns from step11 to step 4 and can see from these rows of matrix *M* that component *a* has 'non-alternative' assignment to computer *D*. This assignment is done at the next third level of the search-tree. At that vertices A, B, C, E get value ∞ in accordance with matrix M without additional computations.

As shown in Figure 8.1 the first acceptable solution obtained has cost value 64 and is optimal. Figure 8.3 depicts the solution.

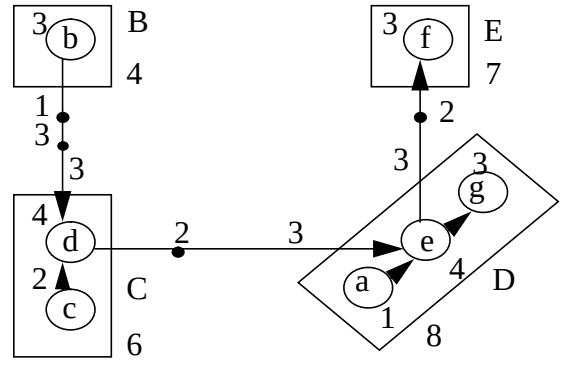


Figure 8.3. Optimal solution obtained by Optimal Procedure

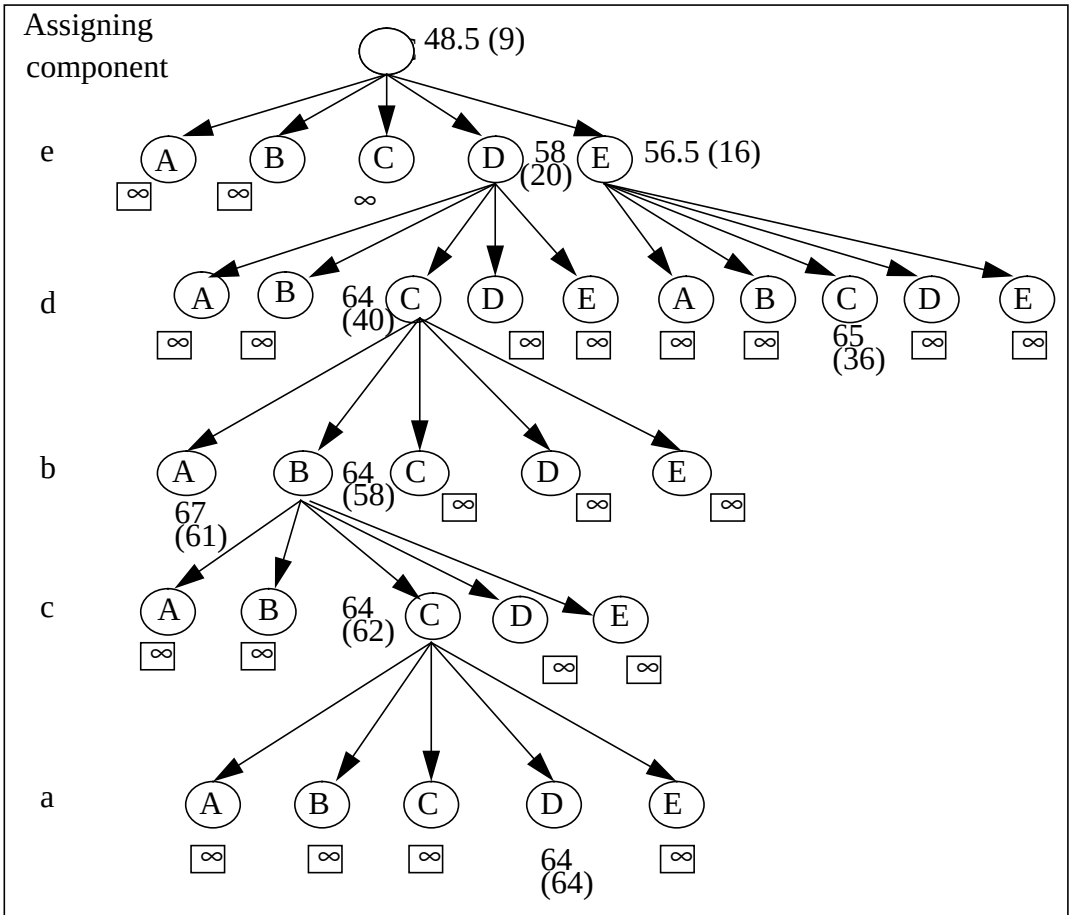


Figure 8.4. Search-tree obtained by Optimal Procedure without use of ‘non-alternative’ assignments

Note, that checking ‘non-alternative’ assignments at step 1 and step 4 of Forward Procedure allows to reduce the search-tree by decrease of number of vertices examined. By way of comparison, Figure 8.4 depicts the search-tree constructing without use of the ‘non-alternative’ assignments and consisting of 31 vertices examined (instead of 26 ones in Figure 8.1).

8.2 Example 2

Let us consider the previous example of mapping the DMA graph depicted in Figure 2.1 to the DCS graph depicted in Figure 2.9. However suppose, that components *f, g* and *a* are attached to computers *E, D* and *C* respectively, and components *b* and *c* can be placed only into computers *A* and *B*. For this case, Optimization Procedure obtains initial value $B_0 = 80.5$ and matrix *M* with following initial cost bounding values of all possible placements of unassigned components

	A	B	C	D	E
b	9	6	∞	∞	∞
c	6	4	∞	∞	∞
d	∞	∞	20	∞	∞
e	∞	∞	39.5	∞	∞

The matrix shows that components *d* and *e* have ‘non-alternative’ assignments to computer *C*. However the computer has not enough resource to locate the components. Therefore the Reduction Procedure is used to obtain the DMA placement into DCS with minimum resource gap.

To compute bounds I_1, I_{23}, I_4 of objective function (7.1), following tables are useful: table 8.1 presenting values concerned with allocations of components onto computers and table 8.2 displaying the interdependence between capacities of virtual channels and communication resources. The tables contains initial data for Reduction Procedure.

To compute bound I_4 (see formula (7.7)), in table 8.1 computers are arranged in decreasing order with respect to computational resource available to mapped DMA $\Delta R_n = R_n - D_n$. Computers *C* and *D*, having the same available resources ($\Delta R_C = \Delta R_D = 5$), are ordered in increasing with respect to utilization factor δ_n. The column next to the last contains sums of utilization factors. The element, that is *i*-th of the order in this column, is equal to the sum of *i* first values δ_{n_k}, i.e. $\sum_{k=1}^{i} \delta_{n_k}$. Similarly, the last column contains sums of available resources of computers.

The row d_{i_k} contains values of required resources of unassigned components. The row $\sum_k d_{i_k}$

used for computing bound G_4 contains sums of required resources of unassigned components. The element, that is *i*-th of the order in this row is equal to the sum of all last elements d_{i_k} of the previous row, starting from element d_{i_j}.

In table 8.1, components are arranged in decreasing order with respect to required computational resources. However each time, when next component have to be chosen for placement (see step 2 of Reduction Procedure in section 7.3), components are rearranged so that the first component must be one with most number of adjacent arcs that connect the component to already assigned components. At the start of Reduction Procedure, such component is *e*.

Table 8.1

Computer	Component							D_n	R_n	ΔR_n	δ_n	$\sum_k \delta_{n_k}$	$\sum_k \Delta R_{n_k}$
	unassigned				attached								
	e	d	b	c	a	e	f						
A								0	5	5	0	0	5
C					1			1	6	5	1/6	1/6	10
D						3		3	8	5	3/8	13/24	15
B								0	4	4	0	13/24	19
E							3	3	7	4	3/7	163/168	23
d_{i_k}	4	4	3	2	1	3	3				max 3/7		
$\sum_k d_{i_k}$	13	9	5	2									

Table 8.2

Communication resource	Virtual channel connections																				A_s	$\sum_{(nm)} D_{nm}$	ε_s
	A, B	A, C	A, D	A, E	B, A	B, C	B, D	B, E	C, A	C, B	C, D	C, E	D, A	D, B	D, C	D, E	E, A	E, B	E, C	E, D			
1	*	*	*	*	*	*	*	*	*	*			*	*			*	*			5	0	0
2			*	*			*	*			*	*	*	*	*	*	*	*	*	*	6	0	0
3		*	*	*		*	*	*	*	*			*	*			*	*			3	0	0
D_{nm}																							max 0

Table 8.2 is used for computation of utilization factors ε_s of communication resources $s \in \rho_{nm}$ over which virtual channels are routed. For example, if an arc with required capacity 3 is assigned to channel *(A,E)*, then the table shows (see symbols *) that this assignment causes utilization of 3 units of capacity of communication resources LAN1, LAN2 and WAN, utilization factors of which become equal to 5/3, 6/3 and 3/3 respectively.

The search-tree constructed by Reduction Procedure is depicted in Figure 8.5. The optimal solution is to assign component *e* to computer *C*, component *d* to *A*, component *b* to *B*, and component *c* to *A*. Figure 8.6 depicts the solution obtained and tables 8.3 and 8.4 present computational resources and capacities of communication resources used by mapped DMA. Computer *A* has 20% of the resource gap ($\delta_A = 6/5$, *gap* = 6 - 5 = 1) and communication resource LAN1 has 20% of the capacity gap ($\varepsilon_1 = 6/5$, gap = 6 - 5 = 1) that causes the capacity gap of virtual channels *(B,A)* and *(A,C)* which are routed over LAN1.

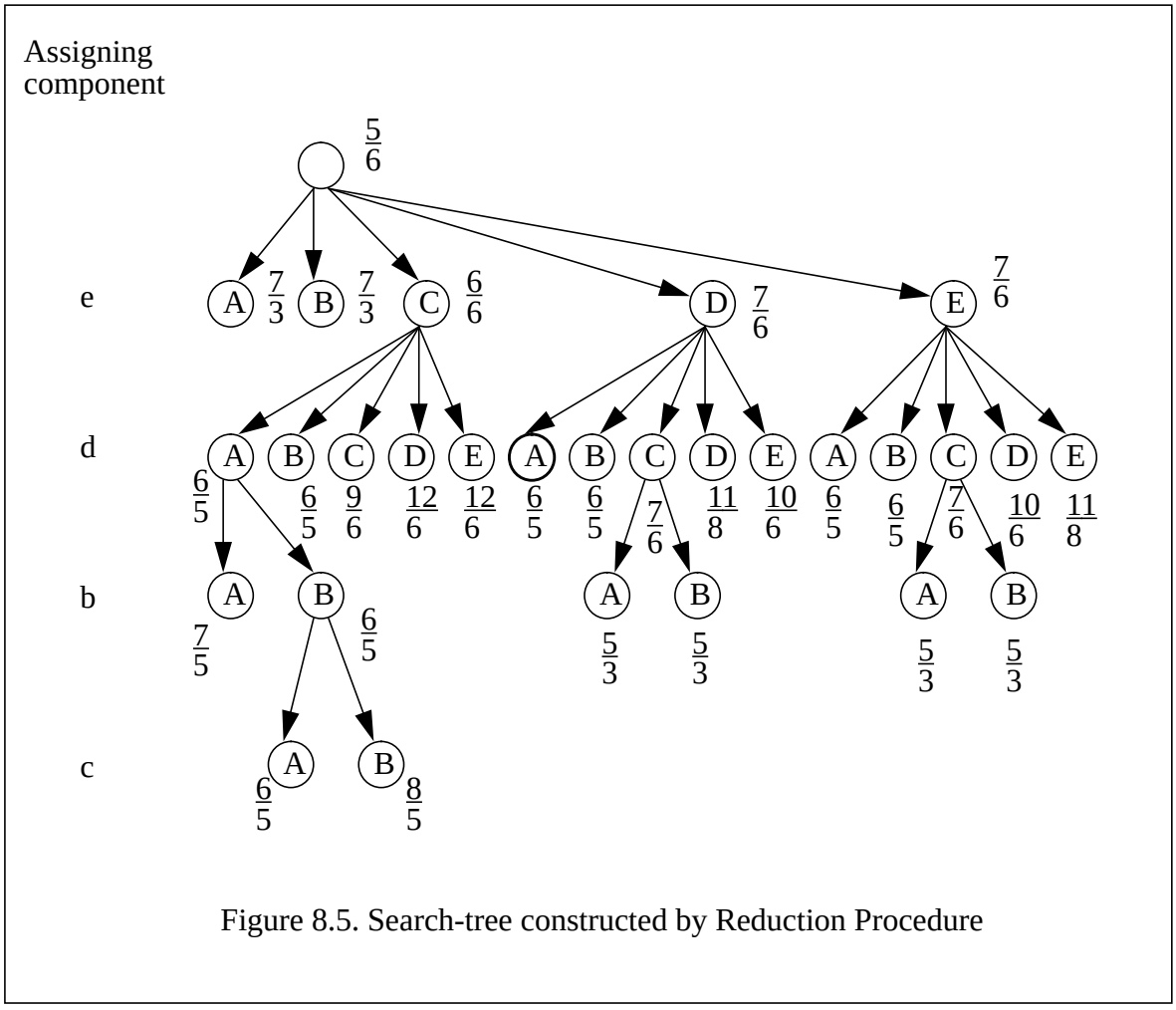


Figure 8.5. Search-tree constructed by Reduction Procedure

If the gap of the computer resource can be divided between assigned components *d* and *c* proportional to resource required for the components, then the resource relaxation for component *d* is equal to $1 \cdot \frac{2}{6} = 0.33$, i.e. $\frac{0.33}{2} \cdot 100\,\% = 16.6\%$ and for component c: $1 \cdot \frac{4}{6} = 0.67$, i.e.

$\frac{0.67}{4} \cdot 100\,\% = 16.6\%$. Thus component *c* gets 1.67 units of computational resource instead of required 2 units, and component *d* gets 3.33 units instead of 4.

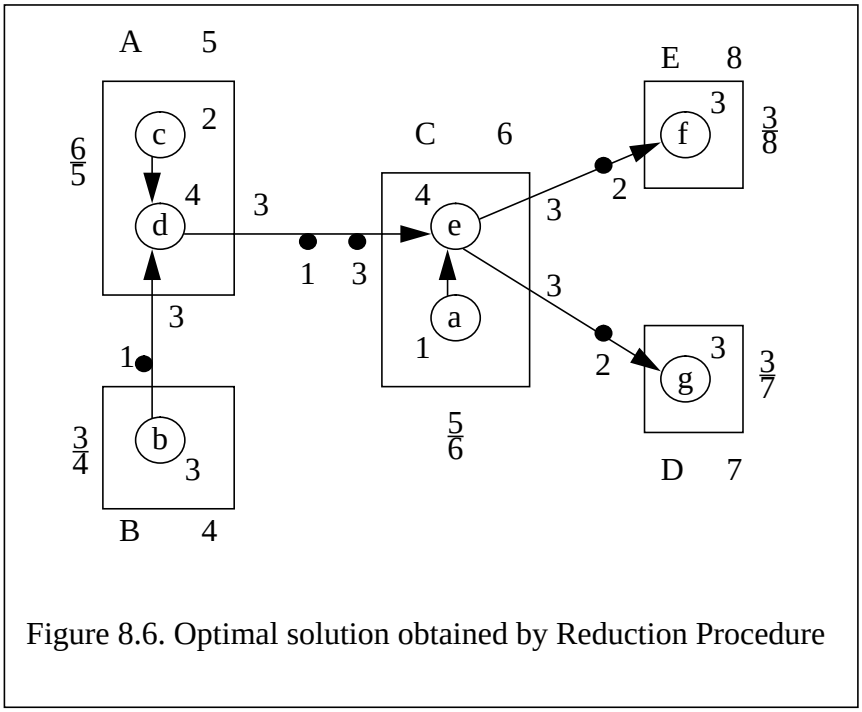


Figure 8.6. Optimal solution obtained by Reduction Procedure

Table 8.3

Com-puter	Component							D_n	R_n	ΔR_n	δ_n	$\sum_k \delta_{n_k}$	$\sum_k \Delta R_{n_k}$
	assigned				attached								
	e	d	b	c	a	g	f						
A		4		2				6	5	-1	6/5		
B			3					3	4	3	3/4		
C	4				1			5	6	1	5/6		
D						3		3	8	5	3/8		
E							3	3	7	4	3/7		
d_{i_k}	4	4	3	2	1	3	3				max 6/5		
$\sum_k d_{i_k}$													

Similarly for the communication resources. Let us divide the gap of LAN1 capacity between channels *(B,A)* and *(A,C)*, routed over LAN1, proportional to the capacity required by data streams from component *d* to component *e* and from *b* to *d*. Then the first stream gets 2.5 units of the capacity instead of required 3 units (that corresponds to 16.6% of the capacity requirement relaxation) and the second stream is characterized by the same values.

Table 8.4

Communication resource	Virtual channel connections																				A_s	$\sum_{(nm)} D_{nm}$	ε_s
	A, B	A, C	A, D	A, E	B, A	B, C	B, D	B, E	C, A	C, B	C, D	C, E	D, A	D, B	D, C	D, E	E, A	E, B	E, C	E, D			
1	*	*	*	*	*	*	*	*	*	*											5	6	6/5
2			*	*			*	*			*	*	*	*	*	*	*	*	*	*	6	6	6/6
3		*	*	*		*	*	*	*	*			*	*			*	*			3	3	3/3
D_{nm}		3			3						3	3											max 6/5

Let us illustrate computing the bound for one of vertices using the tables 8.1 and 8.2. Suppose, that component *e* is assigned to computer *D* and location of component *d* into computer *A* is examined. The corresponding vertex is depicted in Figure 8.1 by the bold circle. Tables 8.5 and 8.6, corresponding to this vertex, are used to compute bounds G_1, G_{23}, G_4 and *BG* in accordance with formulas (7.3), (7.4), (7.7), (7.8).

Note that in two last columns of table 8.5, only two first elements are computed and used further for calculation of bound G_4 . The number of these elements is not more than the number of unassigned components (see parameter *z* in formula (7.7)). Moreover, unassigned components *b* and *c* can be placed only into components *A* and *B*. Therefore only these computers are ordered in the table.

The bound G_1 equal 7/8 is presented in table 8.5.

In accord with formula (7.7) and table 8.5 the bound $G_4 = \frac{1}{2} \cdot \left(\frac{4}{5} + \frac{5}{9} \right) = \frac{61}{90}$.

To compute bound G_{23} for the vertex, the best placement for every unassigned component is determined. Taking into account that unassigned components *b* and *c* can be placed only into computers *A* and *B*, one can calculate utilization factors of computers G_2^{bn}, G_2^{cn} and of communication resources G_3^{bn}, G_3^{cn} for every acceptable placements of these components, *n* = *A,B*. Results of the computations of bounds G_{23}^{b} and G_{23}^{c}, that correspond to the best placements of components *b* and *c*, are presented in tables 8.7 and 8.8 respectively.

In accord with formula (7.4), maximum utilization reached by the best placement of unassigned components *b* and *c* is $G_{23} = max\{G^b_{23}, G^c_{23}\}$ = 6/5.

Table 8.5

Com-puter	Component							D_n	R_n	ΔR_n	δ_n	$\sum_k \delta_{n_k}$	$\sum_k \Delta R_{n_k}$
	unassigned				attached								
	e	d	b	c	a	g	f						
B								0	4	4	0	0	4
A		4						4	4	1	4/5	4/5	9
C					1			5	6	1	1/6		
D	4					3		7	8	1	7/8		
E							3	3	7	4	3/7		
d_{i_k}	4	4	3	2	1	3	3				max 7/8		
$\sum_k d_{i_k}$	13	9	5	2									

Table 8.6

Communi cat-ion resou rce	Virtual channel connections																				A_s	$\sum_{(nm)} D_{nm}$	ε_s
	A, B	A, C	A, D	A, E	B, A	B, C	B, D	B, E	C, A	C, B	C, D	C, E	D, A	D, B	D, C	D, E	E, A	E, B	E, C	E, D			
1	*	*	*	*	*	*	*	*	*	*											5	3	3/5
2			*	*			*	*			*	*	*	*	*	*	*	*	*	*	6	7	7/6
3		*	*	*		*	*	*	*	*			*	*			*	*			3	3	3/3
D_{nm}			3								1					3							max 6/5

Table 8.7

b	G_2^{bn}	G_3^{bn}	G_{23}^{bn} max
A	7/5	7/6	7/5
B	3/4	6/5	6/5
G_{23}^{b} min			6/5

Table 8.8

c	G_2^{cn}	G_3^{cn}	G_{23}^{cn} max
A	6/5	7/6	6/5
B	2/4	7/6	7/6
G_{23}^{c} min			7/6

Table 8.9 illustrates computing utilization factors of communication resources provided that component *b* is placed into computer *B*. In accord with formula (7.8), the final bound of the vertex examined is $BG = max\{G_1, G_{23}, G_4\} = max\{\frac{7}{8}, \frac{6}{5}, \frac{61}{90}\} = \frac{6}{5}$.

Table 8.9

Communication resource	Virtual channel connections																				A_s	$\sum_{(nm)} D_{nm}$	ε_s
	A, B	A, C	A, D	A, E	B, A	B, C	B, D	B, E	C, A	C, B	C, D	C, E	D, A	D, B	D, C	D, E	E, A	E, B	E, C	E, D			
1	*	*	*	*	*	*	*	*	*	*											5	6	6/5
2			*	*			*	*			*	*	*	*	*	*	*	*	*	*	6	7	7/6
3		*	*	*		*	*	*	*	*			*	*			*	*			3	3	3/3
D_{nm}			3		3						1					3							max 6/5

9 Computational Complexity Analysis of Algorithms

9.1 Algorithms of Optimization Procedure

Let us evaluate the computational efficiency of the algorithms. The complete set of all possible locations of DMA, consisting of $I = 5$ unassigned components, into DCS, consisting of $N = 5$ computers connected with each other, is consists of $N^I = 3125$ alternatives. The algorithm proposed have constructed the search-tree depicted in Figure 8.1 that consists of 26 vertices from which only for 9 ones bounding values are computed.

Bounding values of the root has most computational complexity: 25 possible independent locations of 5 isolated components onto 5 computers are examined (see cost bounding matrix *M*). In general case for a vertex of level *i*, number of examined such locations is equal to *(I - i) N + 1*. Then summary number of such independent locations needed to get first (may be optimal, as for the example considered) acceptable solution, provided that Roll-back Procedure will be not used is not more than

$$L = I \cdot N + \sum_{i=1}^{I} ((I-i)N + 1)N = \frac{1}{2}N^2 I(I-1) + 2NI \qquad (9.1)$$

For the example considered $L = 300$. Really, excluding vertices that obtain value ∞ without computations (see values ∞ enclosed in rectangles in Figure 8.1), one gets $L = 5 \cdot 5 + 1 + (3 \cdot 5 + 1) \cdot 3 + 1 + (1 \cdot 5 + 1) \cdot 2 + 1 = 88$, where the first and third 1 correspond to ‘non-alternative’ assignments.

Thus only 88 independent locations of isolated components were examined by the algorithm instead of 3125 possible locations of DMA into DCS. The gain increases with increase of values *N* and *I*.

If component *i* can be placed only into $N_i \le N$ computers, then following relations evaluates computational complexity

$$L = \sum_{i=1}^{I} N_i + \sum_{i=1}^{I} \left(1 + \sum_{j=i+1}^{I} N_j\right) N_i = \sum_{i=1}^{I} N_i \left(2 + \sum_{j=i+1}^{I} N_j\right) \qquad (9.2)$$

The computational complexity of one component location is determined by the computational complexity of subAlgorithm 2, based on the branch-bound method, and depends on number *K* of *unassigned* arcs adjacent to the component and number *W* of channels adjacent to computer which the component is assigned to. A high bound of the summary number of *unassigned* arc locations is

determined by following formula

$$L_{arc} = \sum_{k=1}^{K} (K - k + 1) W = \frac{W(K+1)K}{2}$$

If necessary, some approaches can be proposed to decrease the computational complexity of arc locations:

1. First to assign components that have large number of adjacent arcs. For that at step 4 of Forward Procedure, the ordering of components have to be done in accord with the criterion. This approach allows to decrease value *K*.

2. Instead of subAlgorithm 2 to use an exhaustive method for $K^W \le 10$ to find an optimal placement of K adjacent unassigned arcs.

3. Instead of subAlgorithm 2 to use approximate trivial one that locate arcs (without fractional distribution), ordered in decreasing required capacity, into channels in sequence of increase of capacity cost. This approach gets approximate values for vertices of the search-tree constructed by Forward Procedure and therefore obtains a suboptimal solution.

9.2 Algorithm of Reduction Procedure

To estimate computational complexity of Reduction Procedure based on the branch-bound method, formulas (9.1) and (9.2) can be used to determine summary number of independent component locations needed to get a first acceptable solution provided that Roll-back Procedure will be not used. Taking into account numbers of arithmetical and logical operations needed for computation of parameters G_1, G_2^{kl}, ε_s, G_3^{kl}, G_{23}^{kl}, G_4, order of computational complexity can be evaluated by value $a \cdot N^2 \cdot I^2$ arithmetical and logical operations, where *a* is a constant.

More detailed evaluation of computational complexity of algorithms proposed can be obtain by computer statistical experiments.

10 Conclusions and Future Works

In this paper, the problem of mapping DMA to a DCS is examined. We have proposed models of weighted precedence graphs both for DMA description (including representation of topology, data streams between components, computational and communication resource requirements) and for DCS description (including representation of feasible channel connections between computers, communication resources over which the channels are routed, parameters of computational and communication resources available to mapped DMA).

An approach based on the solving two kinds of the mapping problem is proposed. The first one is formulated as a nonlinear integer programming problem with cost function under constraints

on DCS resources available to mapped DMA. If the first one has not an acceptable solution, then other problem, formulated as minimax nonlinear integer one to find the DMA location into the DCS with minimum DCS resource gap, is solved. To solve both problems, effective algorithms based on the branch-bound method are proposed. Computational efficiency of the algorithms is examined and illustrated by numerical examples.

The nearest future work is to evaluate computational complexity of the algorithms proposed by computer statistical experiments and to develop algorithms taking into account the results of the experiments. We plan to implement the developed algorithms in CINEMA project [2] aimed at developing powerful abstractions for multimedia processing in distributed environments.

The proposed algorithms can be successfully used for an advance reservation service when the provider has to do some planning for future allocations using such parameters of the client requests as the starting time, duration and resource requirements of DMA. We plan to adapt algorithms proposed for an immediate reservation service done in real-time mode.

Other algorithm developments includes taking into account multicasting mode and peculiarities of variable filters used in DMA [14].

At last there is another problem addressed in the paper: development of a method for division the resource gaps of DCS between components and data streams of DMA, if DCS resources available to mapped DMA can not meet the DMA resource requirements.

11 References

1. Furth B., Multimedia systems: an overview. *IEEE Multimedia Systems*, 1, 1994, p. 47 - 59.

2. Rothermel K., Barth I., Helbig T., CINEMA - an architecture for configurable distributed multimedia applications. *Tech. Report 3/1994*, Universität Stuttgart, Fakultät Informatik, pp. 20.

3. Norman M.G., Thanisch P., Models of Machines and Computation for Mapping in Multicomputers. *ACM Computing Surveys*, 3, 1993, p. 263 - 302.

4. Berman F., Snyder L., On mapping parallel algorithms in parallel architecture. *J. Parall. Distr. Comput.*, 5, 1987, p.439 - 458.

5. Houstis C.E., Aboelaze M., A comparative performance analysis of mapping applications to parallel multiprocessor systems: a case study. *J. Parall. Distr. Comput.*, 1991, pp. 17 - 29.

6. Efe K., Heuristic models of task assignment scheduling in distributed systems. *Computer*, 6, 1982, p. 50 - 56.

7. Kapelnikov A., Muntz R.R., Ercedovac M.D.,A modelling methodology for the analysis of concurrent systems and computations. *J. Parall. Distr. Comput.*, 6, 1989, p. 568 - 597.

8. Ammar M.H., Cheung S.Y., Scoglio C.M., Routing multipoint connections using virtual paths in an ATM network. *IEEE INFOCOM' 93*, 1993, p. 98 - 105.

9. Kleinrock L., The latency/bandwidth tradeoff in gigabit networks. *IEEE Commun. Magazine*, April, 1992, p. 36 - 40.

10. Ferrari D., Gupta A., Ventre G. Distributed advance reservation of real-time connections.- htp://www.icsi.berceley.edu, 1995.

11. Minoux M., *Mathematical Programming, Theory and Algorithms*, John Wiley and Sons, 1986.

12. Reinfeld N.V., Vogel W.R., *Mathematical Programming*, Englewood Cliffs: Prentice-Hall, 1958.

13. Schwarz M., *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison-Wesley Publ. Comp., 1987.

14. Dermler G., Fiederer W., Barth I., Rothermel K., A Negotiation and REsource Reservation Protocol (NRP) for Distributed Multimedia Applications. *Tech. Report 11/1995*, Universität Stuttgart, Fakultät Informatik, pp. 27.