Proposal of Asynchronous Distributed Branch and Bound

Atsushi Sasaki†, Tadashi Araragi†,

Shigeru Masuyama‡

†NTT Communication Science Laboratories,

NTT Corporation,

Kyoto, Japan

‡Dept. of Knowledge-based Information Eng.,

Toyohashi University of Technology、

Toyohashi, Japan

Outline

- Proposal of a new framework of asynchronous branch and bound to obtain optimal solutions for discrete optimization problems where each variable denotes a host, e.g., load balancing.
- This framework is promising as it has more flexibility and robustness than conventional ones with some centralized control.

Load Balancing

- Fundamental problem to affect the performance of distributed systems
- NP-hard（e.g., Multiprocessor Scheduling [Garey, Johnson 79], Mobile Agent Allocation [Sasaki et al 05]）
- Most conventional solution methods for discrete optimization are kinds of local search [Shiragi et al, 95]
  - Further improvement from a local optimum is difficult
- Few studies focus on optimization

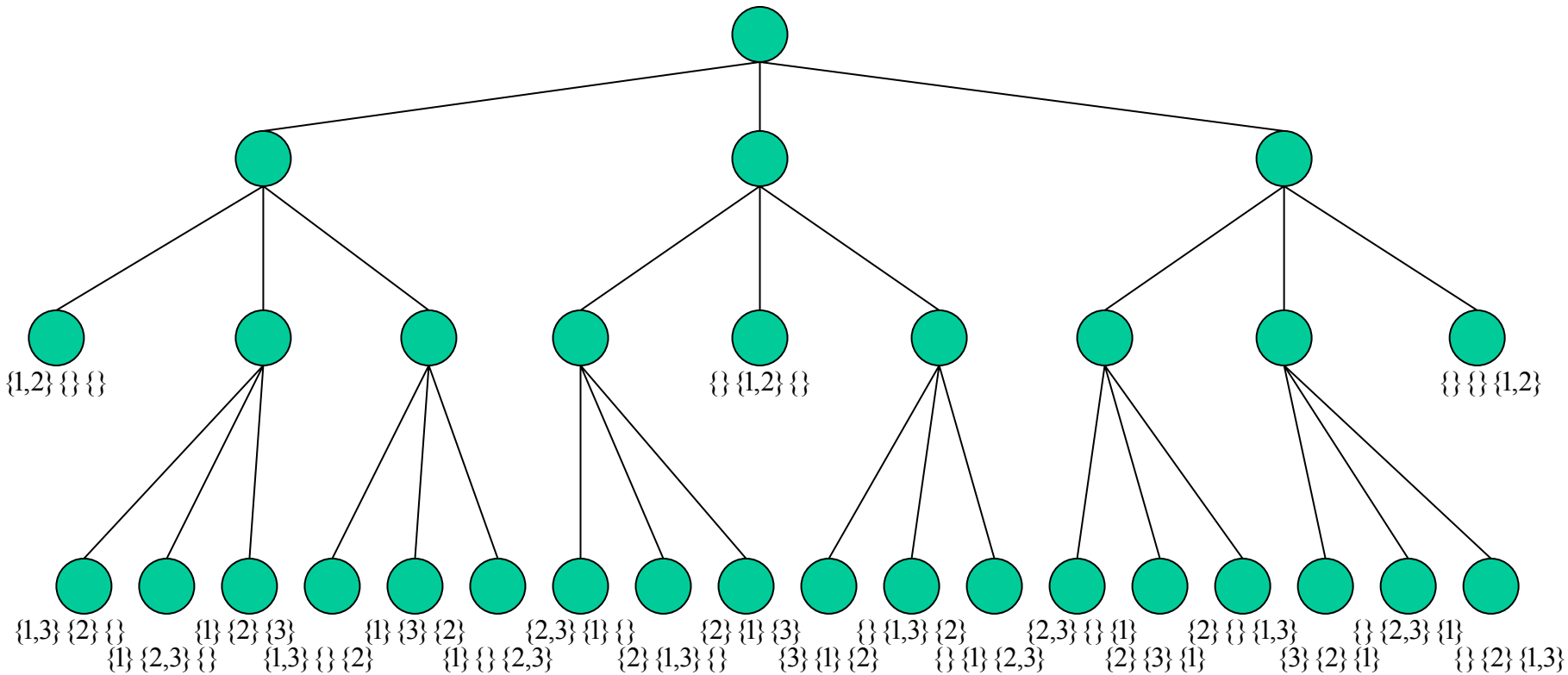
Desirable Properties

- Exact optimal solution is obtained
  （from any state, non-optimal solution can be improved）
- Fully distributed control
- Can be used as an approximation algorithm
  （especially in a large-scale system）
- Asynchronous operation
- Fault tolerance and adaptation to dynamic changes
- High performance

Conventional Branch and Bound under Distributed Environment

- Synchronous branch and bound [Yokoo, 01]
  - Simulating the sequential branch and bound
    - Assigning an agent to each variable and execute just like sequential branch and bound
  - Exactly one agent operates at a time (e.g., when branching is executed) so that a unique branching tree is maintained
- Distributed branch and bound[Barta et al, 02]
  - Assigning each partial problem obtained by branching operation to a different host (natural way in a distributed environment)
  - Essentially the same as the parallel branch and bound

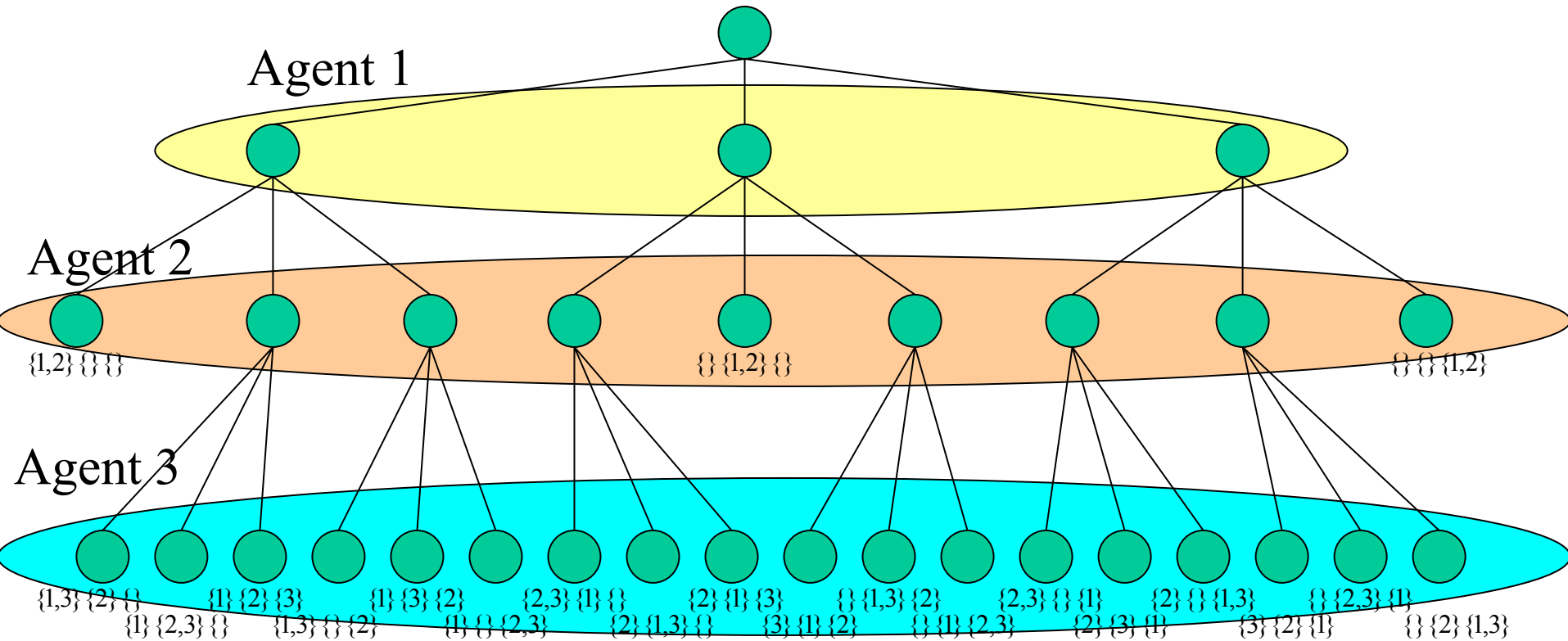
An Example of Branching Tree in a Conventional Distributed Branch and Bound with Central Control



※ The case where both the number of tasks and that of hosts are 3.

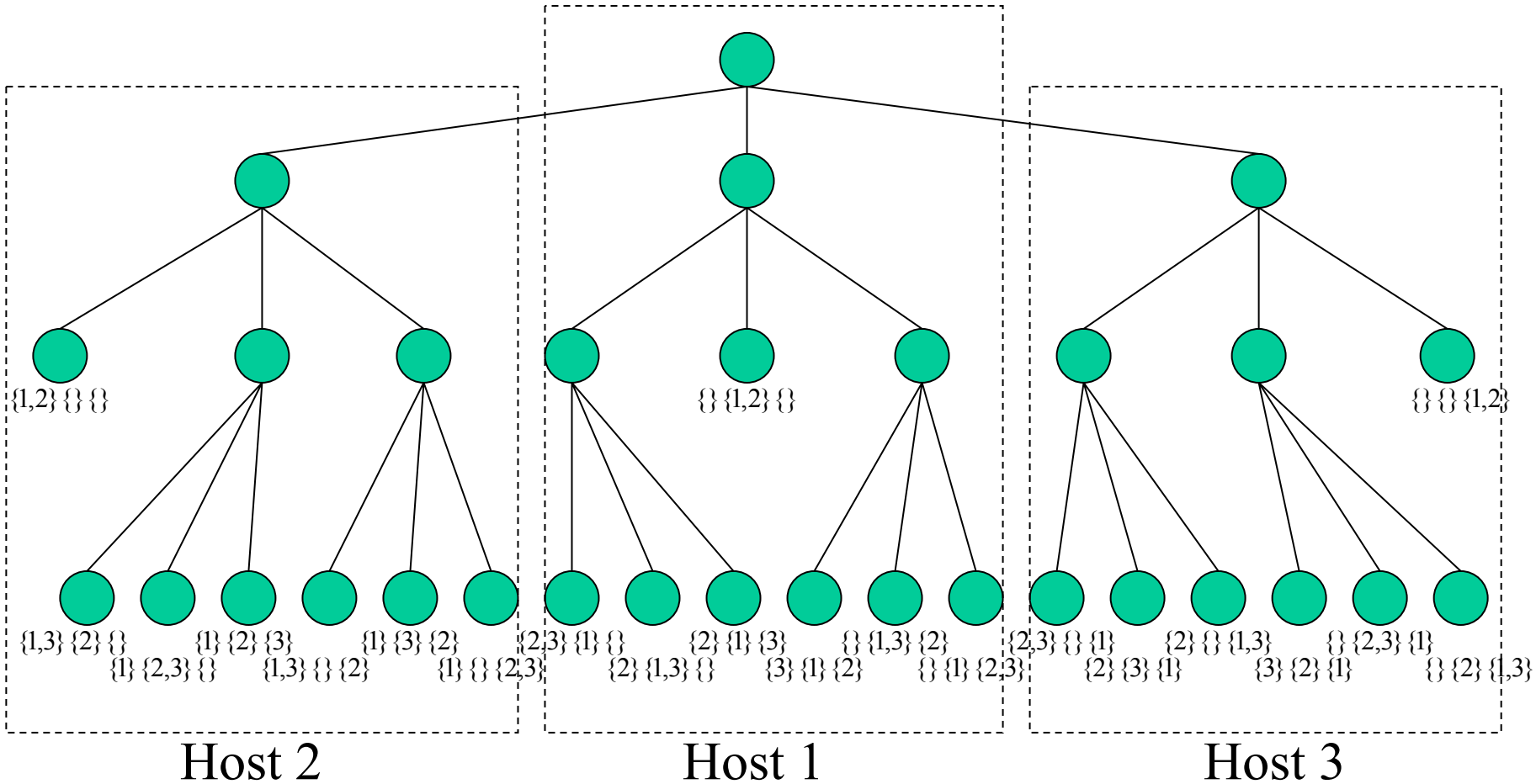
※ Both of them maintain a unique branching tree.

Synchronous Distributed Branch and Bound



※ Both the number of tasks and that of hosts are 3.

Distributed Branch and Bound



※ Both the number of tasks and hosts are 3.

Drawback of the Conventional Distributed Branch and Bound

- Fragile to fault and dynamic changes
- Essentially centralized control
  - Difficult to apply to large-scale systems（as an approximation algorithm）

Strategy of Our Research

- Each host operates asynchronously and cooperate to enumerate (implicitly) all the feasible solutions
- Each host processes only information relevant to the host
- Utilize the fact that the initial state is feasible

Definition of Geographical Optimization

$$\underset{x}{\text{Minimize}} \max_{i} y_i(x^0, x)$$

$$\text{s.t. } x = (x_1\ x_2 \cdots x_m)^T$$

$$x_i : 1 \le x_j \le n$$

$$y_i(x^0, x) = f_i(x) + g_i(x) + k_i(x^0, x)$$

$$f_i(x) = p_i \sum_j w_j [x_j = i]$$

$$g_i(x) = \sum_{j,r} q_{x_j x_r} c_{jr} [x_j = i][x_r \neq i]$$

$$k_i(x^0, x) = \sum_j q_{x_j^0 x_j} b_j [x_j = i][x_j^0 \neq i]$$

$$[z] = \begin{cases} 1 & z : true \\ 0 & otherwize \end{cases}$$

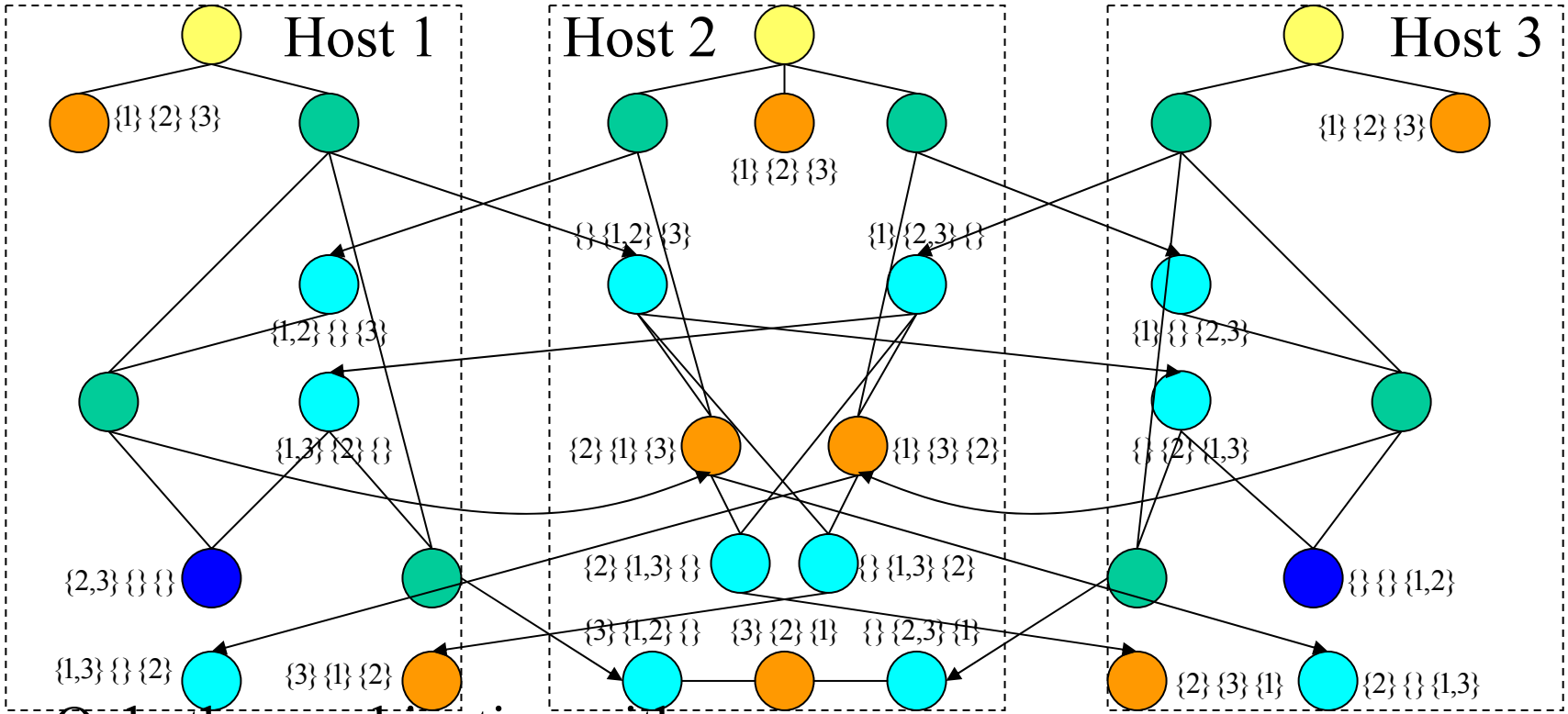
Notations for static version of Mobile Agent Allocation [Sasaki et al, 05]

x : positions of agents (described by host ID)
x^0: initial positions of agents
y: finishing time
f: CPU cost
p: CPU power of a host
w: load of an agent
g: communication cost between agents
q: communication speed of a link
c: communication amount between agents
k: migration cost of an agent
b: size of an agent at migration]

Asynchronous Distributed Branch and Bound（Asynchronous DBB)

(n=3,m=3) Host 1 Host 2 Host 3

Undirected edges denote those of the network, and (1)(2) (3): initial state



: Only the combination with initial state is deleted.

: terminated as its lower bound >y_0 ({1}{2}{3})

: a state with migration to an adjacent host

: feasible solution left（when y_0 unchanged）

Basic Operations of the Proposed Asynchronous Distributed Branch and Bound(Asynchronous DBB)

- Update of the incumbent value:
  - New value is spread in the system by flooding
- Branching Operation:
  - Generate a new state by combining at least two states
  - Notify adjacent hosts of the state with migration
  - Compute cooperatively the objective function of each state
- Bounding Operation:
  - Terminate a state from which no optimal solution is derived
  - Notify adjacent hosts of the terminated state, if necessary

※ State: combination of variable values that is a candidate of a solution where migration is also considered

Messages used in the Asynchronous DBB

- Update of the incumbent value : *update*(*y'*,*a'*)
- Branch operation :
  - Branch : *migrate*(x_j,*s*), *local_improve*(*s*,*y'*)
  - Computation of objective function : *local_max*(*s*,*y'*,*a'*), *local_max_fix*(*s*,*y'*,*a'*)
- Bound operation : *bound*(*s*,*a'*)

s:state（represented by the difference from the initial state）, *y'* : objective function value of *s*, *a'*: host from which the message was sent (source host of the message)

Outline of the Operation at initial state

- Incumbent value ← the current value of the objective function
- Generate a state where variables are migrated to adjacent hosts from the current hosts
- Notify adjacent hosts of the state s and variable x_j to be migrated using message $migrate(x_j, s)$

Other operations are triggered by some message

An Example of Operations at the Initial State

- The case where h_1 (adjacent host : h_2) has only variable x_1 :
  - Initialization of the incumbent value : $y=y_0$, $a=a_0$
  - Generate a state where variable x_1 is to be migrated to h_2, then send $migrate(x_1, x^0)$ to h_2

- The generated state is put into I

Set S : set of enumerated states at present :
S has only x^0 at the initial state.

Set S' : a terminated state whose descendants may yield a better value than the incumbent value : Φ at the initial state

Operation when *Update* is Received

- If the value v carried by the message is smaller than the incumbent value z, update the incumbent value to v and send it to hosts other than the source of the message.
  - If there is a state terminated by the update of the incumbent value, then bounding operation is executed.
- If $v > z$, then do nothing.
  - If $v = z$, then tie breaking is done according to ID of each host.

Operation when *bound* is Received

- If the *bound* has been received previously, ignore it.
- Remove the state attached to the message from *S*
  - If further branching may yield a good solution, then append the state to *S'*
  - If migration of a variable from some other host becomes impossible by the removal of the state, then send *bound* to the host.
- If the attached state does not have migration, then send it to hosts excluding the source of the message.

Operation when *Migrate* is received

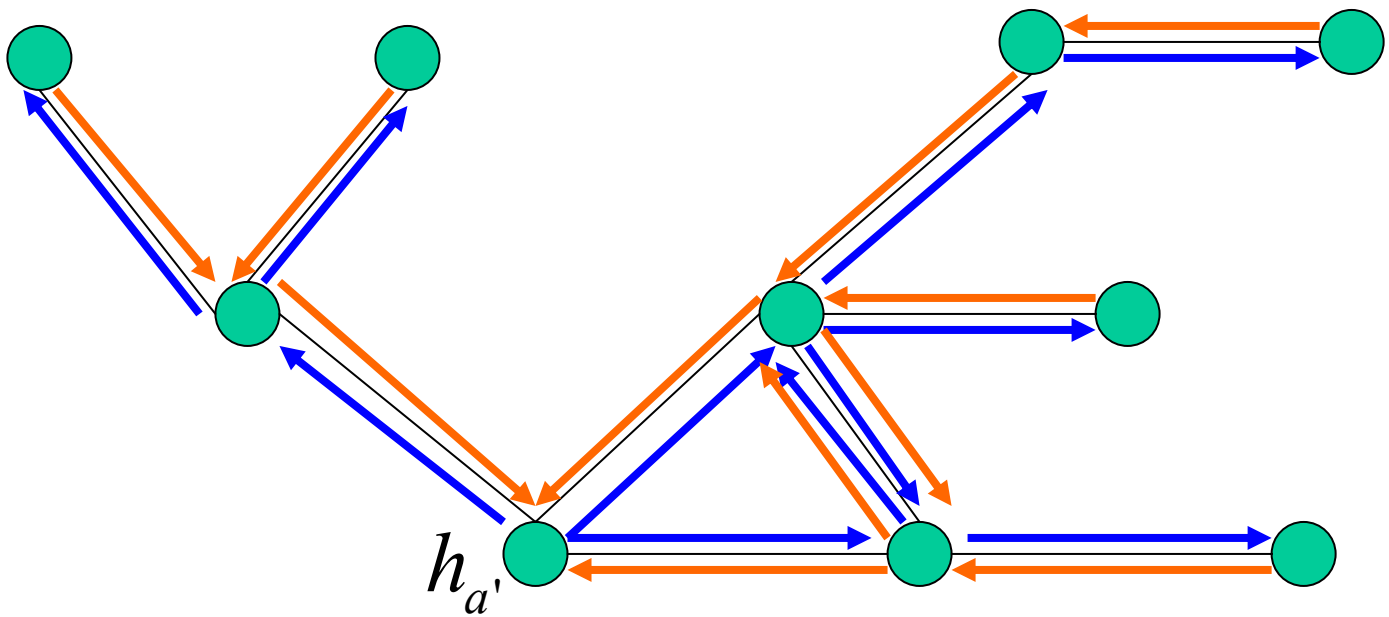
- Generate a state where variables to be migrated are set in it and combine the state and states in *S*, S' to generate a new state.（enumeration）
- Assign its objective function value to *v* of the generated state. If $v>z$ where *z* is the incumbent value, then terminate the state.
- If the non-terminated generated state has migration, then notify the destination of the migration by sending *migration*.
- If all the variables in the non-terminated generated state is fixed, then send *local_max* to compute its objective function value.

Computation of the Objective Function Value

- *local_max*(*s,y',a'*)
  - Spread the object function value *y* of state *s* at host $h_{a'}$
- *local_max_fix*(*s,y',a'*)
  - Used for fixing the objective function value spread by the above operation.
  - The objective function value is fixed to *y'* when host $h_{a'}$ receives this message from all the adjacent hosts.
- Computation is done only at hosts that is changed from the initial state.
  - If this is not the case of h_{a_0}, then *local_improve* is sent to h_{a_0}
- Multicast, instead, can be applicable.

※ h_{a_0}: host that gives the objective function value at the initial state.

How to Obtain the Value of the Objective Function



Termination

- Terminate when no message is in the distributed system

⇒ incumbent values at different hosts are the same

- A state which gives the incumbent value exists in S at some host ⇒ the incumbent value is the optimal solution

※ S: set of states currently enumerated

Correctness

- The incumbent value is integer and monotonically decreasing⇒it reaches to the optimal value if enumeration is realized
- Enumeration（excluding terminated states）: realized by message *migrate* and message *local_improve*

Discussion on Properties and Future Prospects

- Approximation
- Combining some other approximation methods
- Coping with large-scale systems
- Asynchronous operation
- Fault tolerance and flexibility for the dynamic change
- Efficiency

Approximation

- The solution corresponding to the incumbent value provide an approximation solution as is the case of sequential branch and bound
- If migration costs are high, optimal solution may be obtained earlier.

Using Some Other Approximation Method to Obtain an Upper Bound

- Introducing an upper bound computed by some approximation algorithm may helpful for cutting the branching tree
- Thus, developing an efficient distributed approximation algorithm may help

Coping with Large-scale Systems

- Seamless decomposition is realized by restricting the length of movement for each variable.
- By this restriction, the number of messages may be reduced drastically from $O(m)$
- However, the incumbent value should be carefully treated.

Asynchronous Operation

- Proposed asynchronous DBB is highly asynchronous as operations at each host are triggered by messages and relation among processing of different hosts are not strong.
- This property may help improve the efficiency, e.g., by assigning priority of processing for each message
  → This is one of the important topics for future research

Fault Tolerance and Coping with dynamic change

- Asynchronous DBB can partialy cope with them.
  - Failure at a host where the objective function at initial state is not maximal and different from the current incumbent value can be torelated
  - Appending a new host
  - Appending a new variable（only when the value of the objective function does not exceed the incumbent value).
- Other cases are left for future research
  - Including the modeling issues, e.g., how to treat variables on the failure host

Efficiency

- Searching strategy used in sequential branch and bound cannot be straightforwardly applied to asynchronous DBB → searching strategies fit for asynchronous DBB should be developed.
- The number of messages and that of memories required is very large
  - Some reduction method of messages and memories is required

Conclusion

- A new framework of asynchronous distributed branch and bound (asynchronous DBB) was proposed.
- Asynchronous DBB is promising from the viewpoint of fault tolerance and flexibility

- This may become an infrastructure for future large-scale distributed system

Future Research Topics

- Considering fault tolerance and adaptaion to dynamic changes（including modeling）
- Considering how to improve efficiency
  - Examination of detailed operations（e.g.，whether a message can be sent or not）

  （e.g., message reduction）
  - Considering branching order
  - Reduction of space complexity
- Considering good distributed approximation algorithm for obtaining upper bound
- Experiments for evaluation
- Coping with mixed integer programming