University of Ljubljana, Faculty of Computer and Information Science

Intelligent agents

Prof Dr Marko Robnik-Šikonja, Intelligent Systems, January 2019

Contents

- types of agents and agent architectures
- multiagent learning
- distributed constraint satisfaction
- distributed shortest path finding

Literature

Yoav Shoham, Kevin Leyton-Brown: *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009

This lecture is only a teaser, we cover part of Chapters 1 and 2, but things get useful with game theory.

Some terminology

- agent: many definitions, many areas
- Intelligent agent: autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals
- Biological, artificial
- Software agents, intelligent agents
- Types and features
- Agent architectures
- A fundamental concept in intelligent systems

Agent

- Control system
- Sensors
- Actuators
- Environment

- Autonomous: acts independently and makes its own decisions
- Social ability: can interact with other agents
- Reactivity: reacts to stimuli
- Proactivity: pursues its own goals and acts in its own (self-)interest

Multiagent systems

- Distributed program solving
  - autonomous,
  - flexible,
  - collectively organized actions
  - (goal oriented).
- Examples: drones, kilobots, helicopters, boats etc, see any of the videos on this topic: an example

Interactive environment

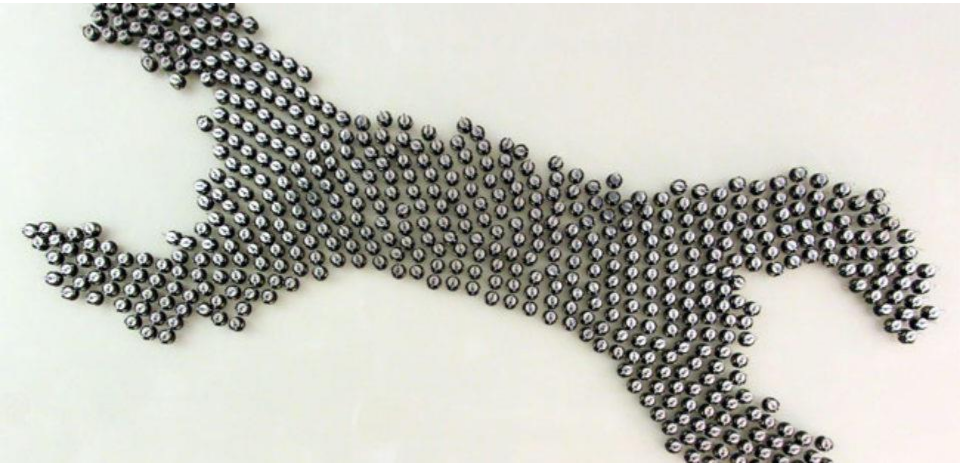
- agent percepts information
- actions affect the environment
- internet, game playing, robotics (e.g., robo-soccer)

Autonomous and flexible

- Without explicit directives
- Control
- Model of the environment
- (learning from experience)
- Responsiveness,
- Planning, goal oriented

Collective actions

- Interactions with agents and humans
- Goal oriented cooperation: strategies, negotiation, specialization
- Distributed, asynchronous
- Objects and agents



Applications

- Production
- Automatic process control
- Telecommunications
- Information management
- e-business
- Interactive games
- Services
- ...

Types of agents

- reactive agents
- collaborative agents
- interface agents
- mobile agents
- information-gathering agents
- hybrid agents

Reactive agents

- Response according to pre-specified rules
- Mail sorter, spam filter, calendar management
- Learning and revision of rules

Goal oriented agents

- Following the goal
- Planning and search
- Tickets, products

Utility based agents

- Utility functions
- Goals and utility maximization, Multiobjective decision making
- Rationality (e.g., in games one can loose on purpose)
- Kahneman, Tversky: (A. 100% 3000, B. 80% 4000 C. 20 % 4000 D. 25% 3000),

Interface agents

- Personal assistants,
- Learning
- Tutoring systems, preference learning in search

Mobile agents

- Physical and virtual mobility
- virus, supervision program

Information-gathering agents

- Internet, intranet, mail
- Precision, recall
- Learning
- Noisy data, data relevance

Collaborative agents

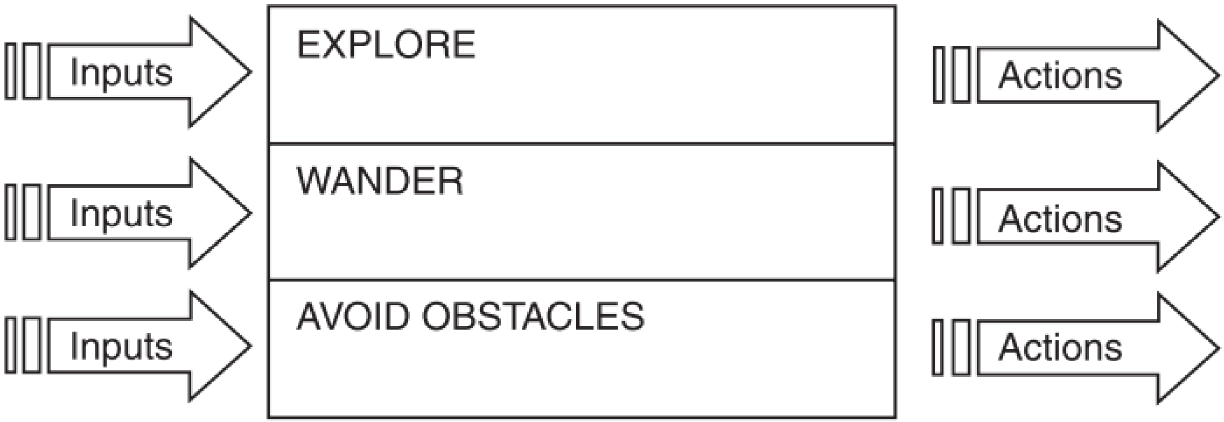
- Weakly interactive agents (ants, genetic algorithms)
- redundancy, parallelism

Agent architecture

- a blueprint for software agents and intelligent control systems, depicting the arrangement of components

Subsumption architecture

- Brooks, 1985,
- intelligence without representation
- Multilevel
- Each level follows its own goal
- Higher levels can block lower levels



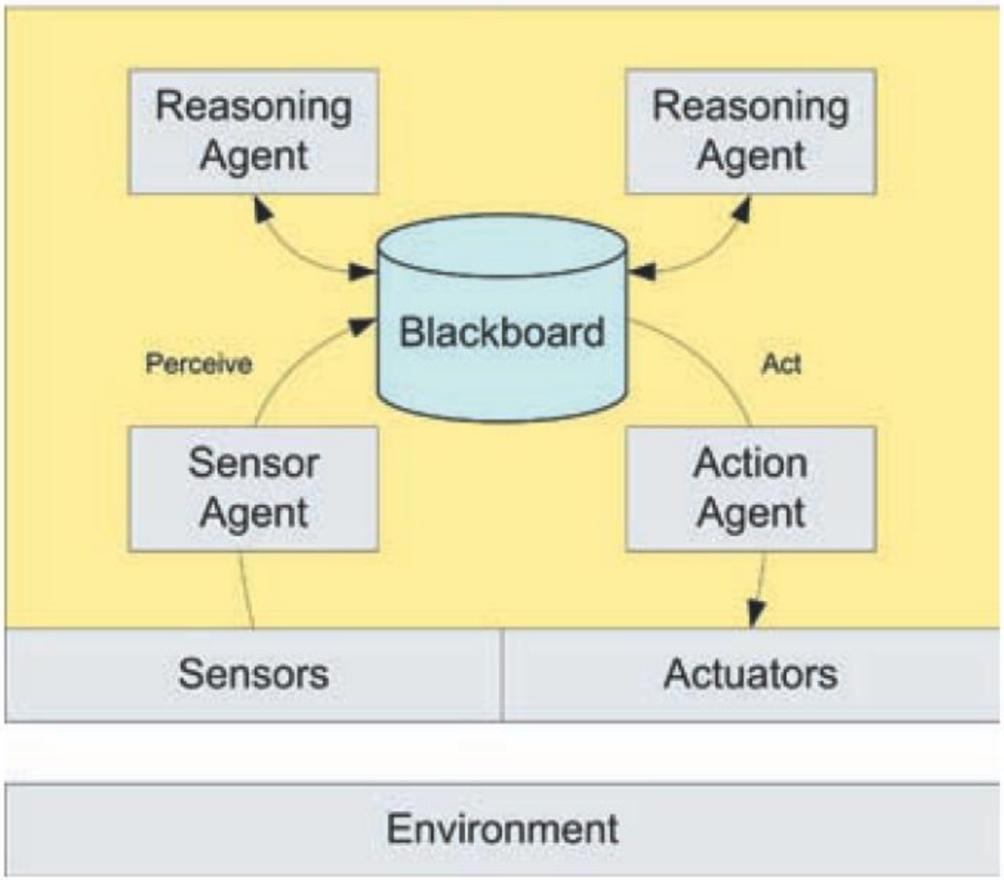
- Each level contains its own rules, e.g., if-then rules
- Adding new levels is easy
- Debugging is hard

BDI architecture

- BDI (Belief Desire Intention)
- planning
- Bold and cautious

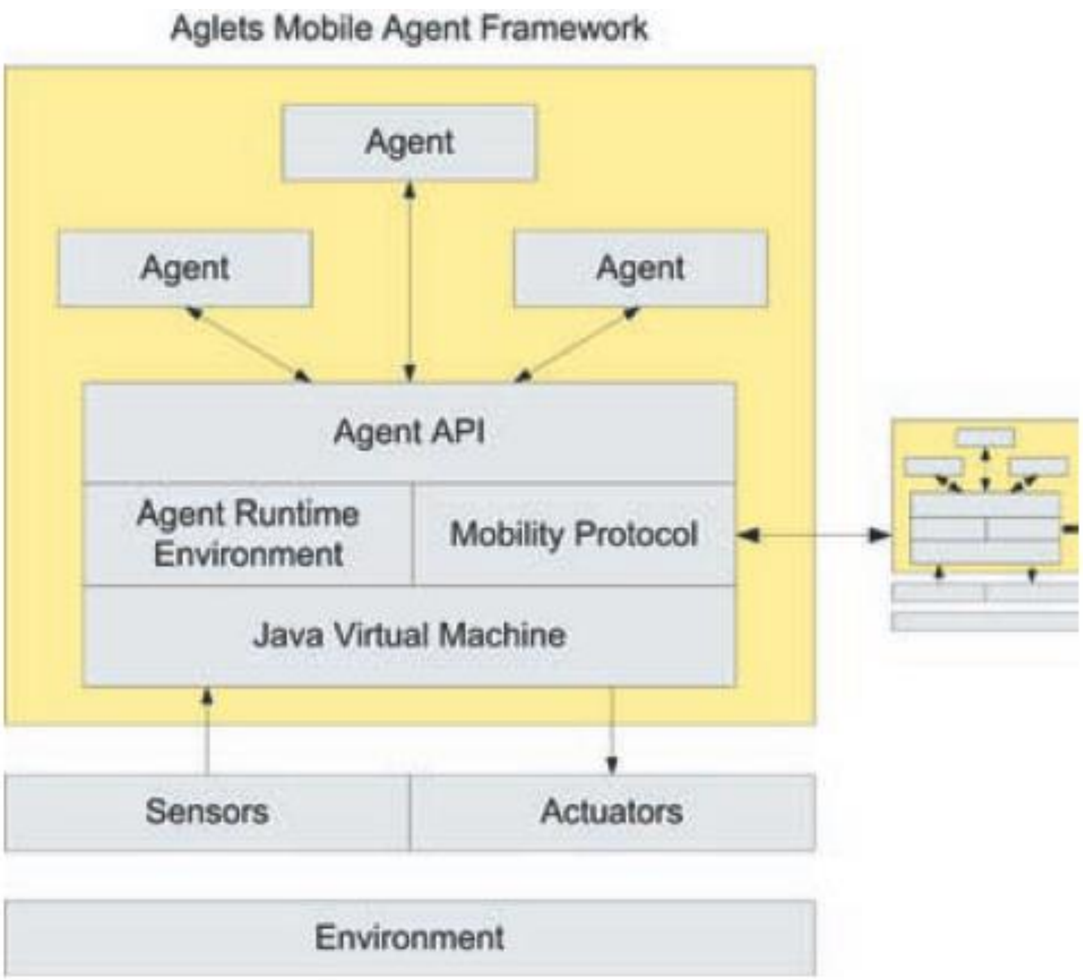
Blackboard architecture

- Sharing common work area
- Specialization
- Coordination
- Threads



Mobile architecture

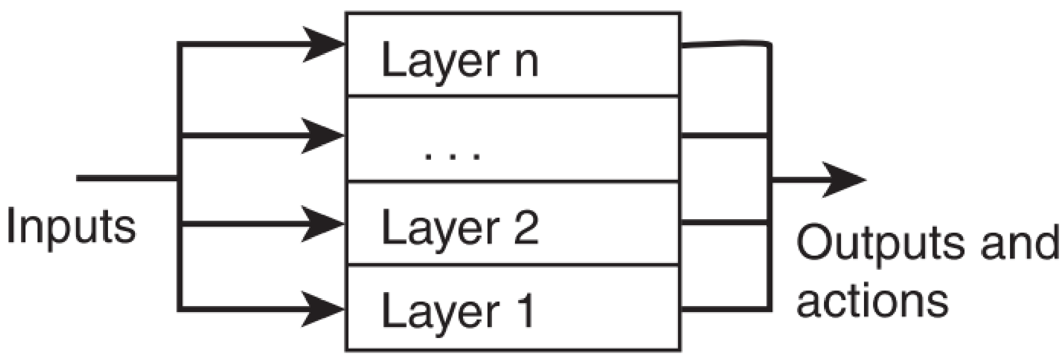
- Agent-based modeling software (many frameworks)
- Early example: Aglets, IBM 1990
  - Java, serialization, sandbox



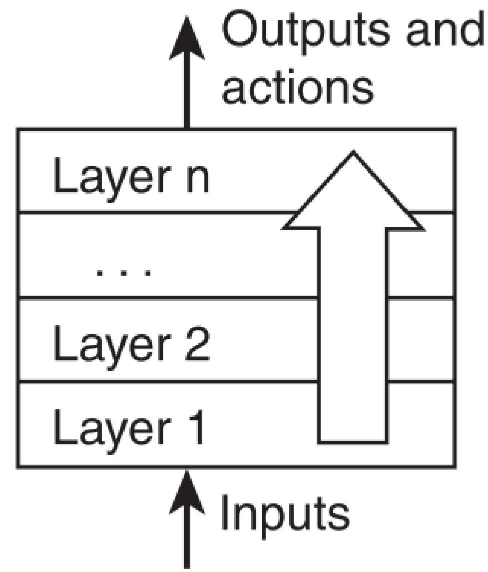
JADE

- JAVA Agent DEvelopment Framework
- an open source platform for peer-to-peer agent based applications
- conforms to FIPA standard (Foundation for Intelligent Physical Agents)

Horizontal and vertical architectures



Horizontal Architecture



Vertical Architecture

Environment

- Deterministic
- Nondeterministic

Learning agents

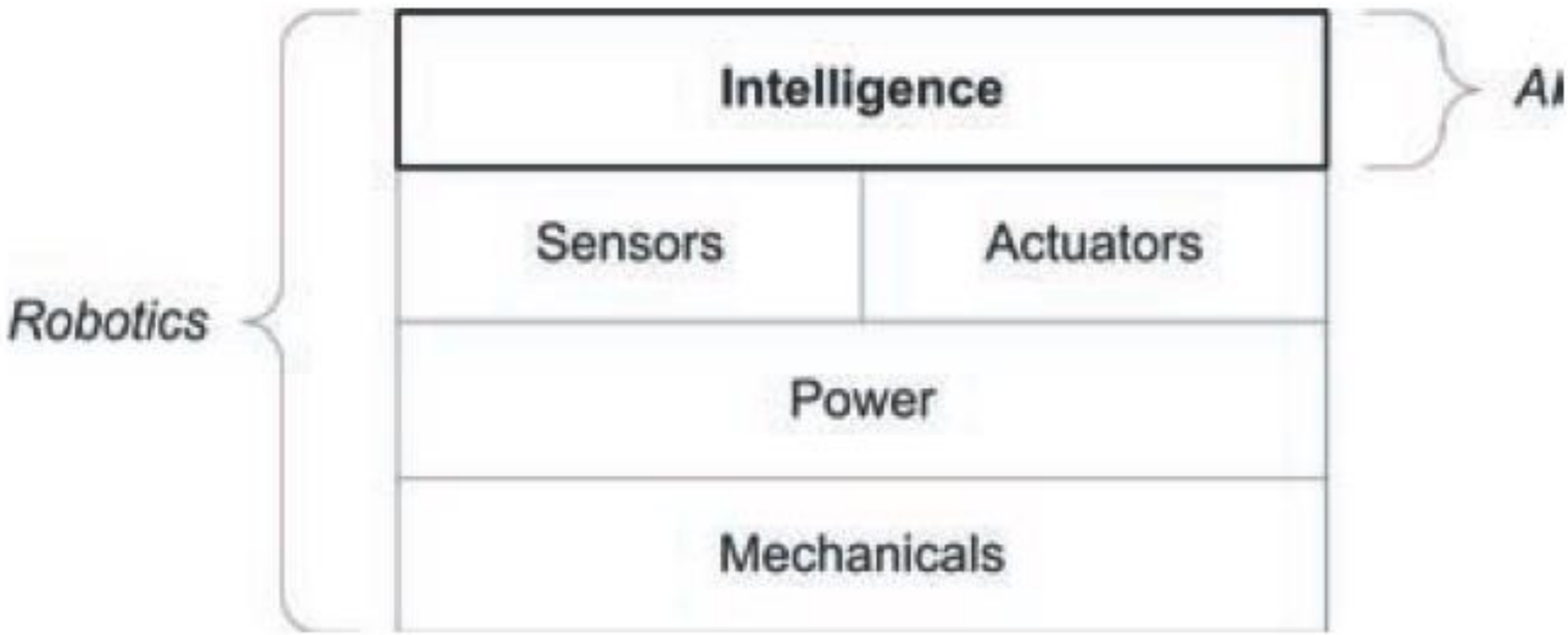
- Learning is an adaptation
- Multiagent learning
  - Centralized
  - Distributed

Robotic agents

- Complexity of real-world environment
- Risk management
- Industrial robots
- Robot explorers (Mars, autonomy, moving around, insects)

Robotics : Intelligent Systems view

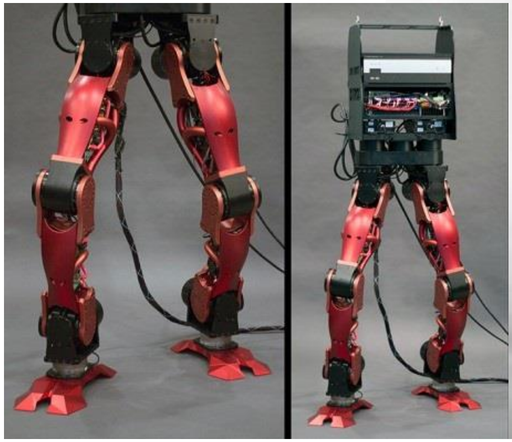
- Testbed for intelligent systems



Taxonomy

- fixed (industrial, robotic hand with several degrees of freedom)

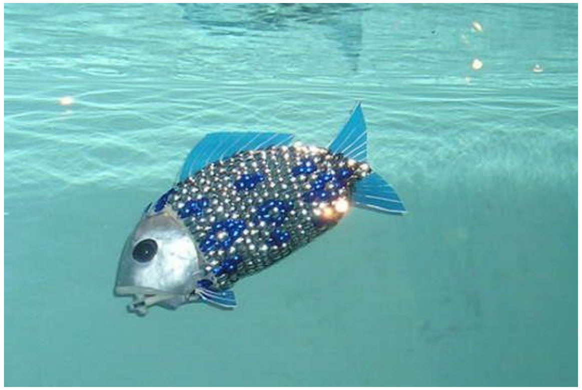
- legs (1,2,4,6,8)



Taxonomy

- Wheels

- Underwater and amphibious (fish, crabs, worms)



Taxonomy

- Airborne (drones, quadropters, satellites)

- Polymorphic, swarms
- Physical and softbots

Sensors

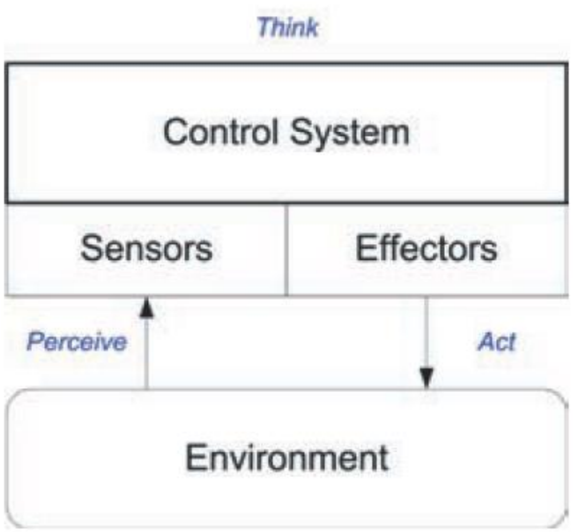
- vision (electromagnetic waves)
- hearing (air)
- Taste and smell (chemical receptors)
- touch (pressure)
- echolocation (ultrasound)
- electroception (electric stimuli, current and field)
- magnetoception (detect magnetism, magnetic field)
- equilibrioception (balance, acceleration)
- thermoception (temperature)
- ...

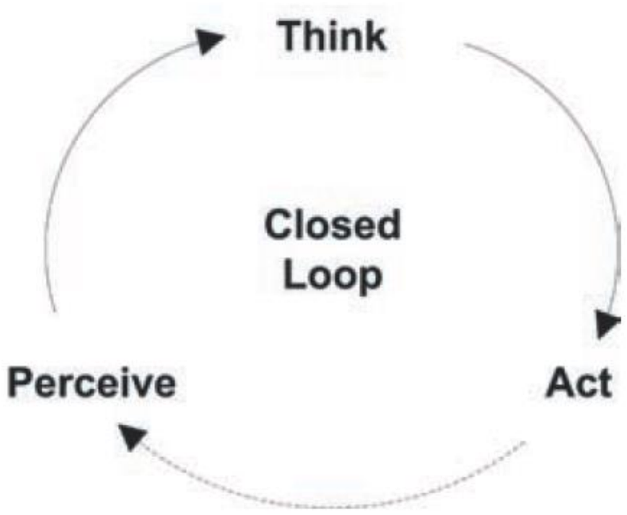
Actuators

- Wheels, legs, motors,
- Hands, grasps,

Control system

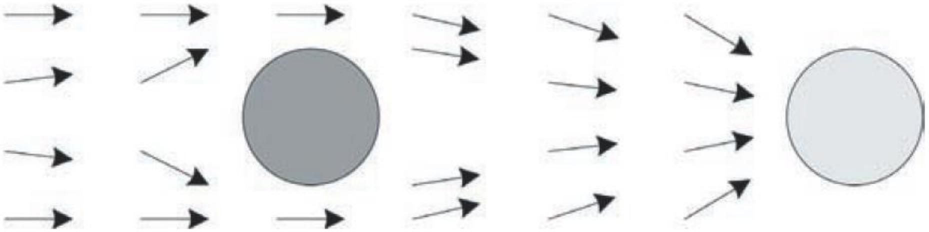
- reactive
- Subsumption
- Neural networks, evolutionary approaches





Planning

- Essential component of intelligent behavior
- Anytime planning: always ready, but improves with time
- Cell decomposition
- Potential field
- Landmarks
- Visibility graph



Tools

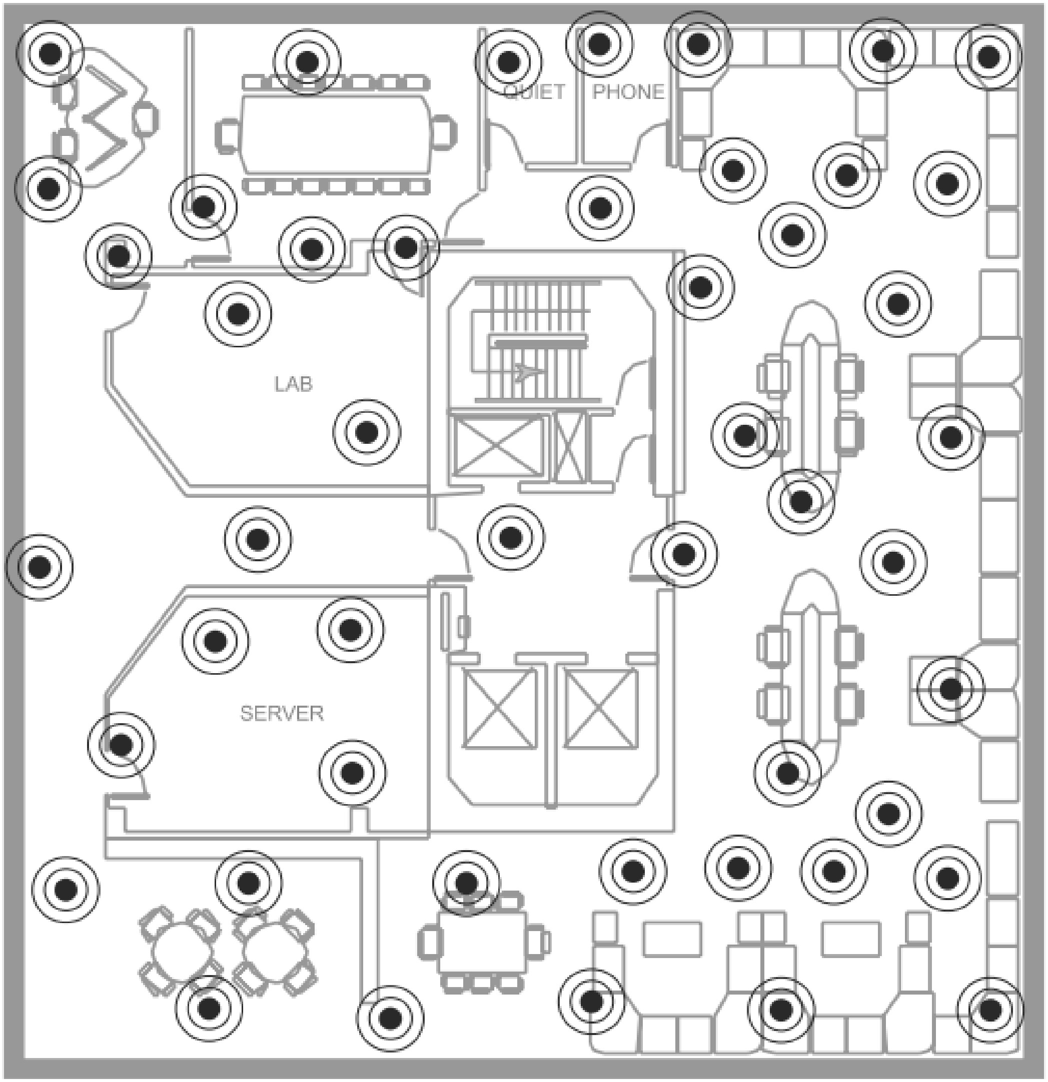
- Robotic languages
- Simulators
- Robot operating systems (ROS)

Distributed computing with agents

- Agents collaborate to achieve a common goal defined by central authority
- Autonomous agents, only local communication
- The goal is to find a solution with global constraints
- The task: prepare an algorithm for the agents

An example: sensor network

- Limited computational resources
- Local communication
- Global constraints and solutions

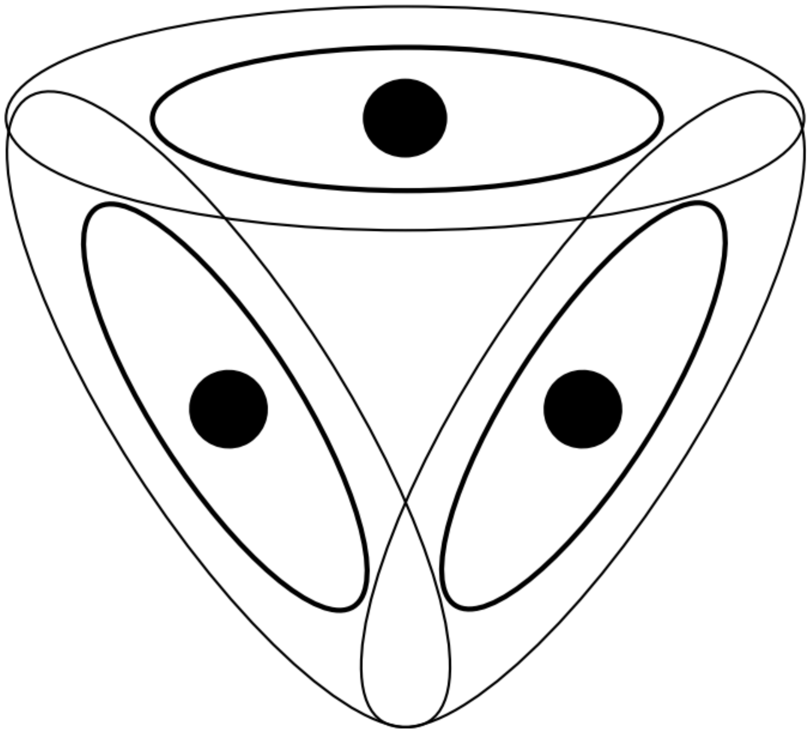


Constraint satisfaction problem

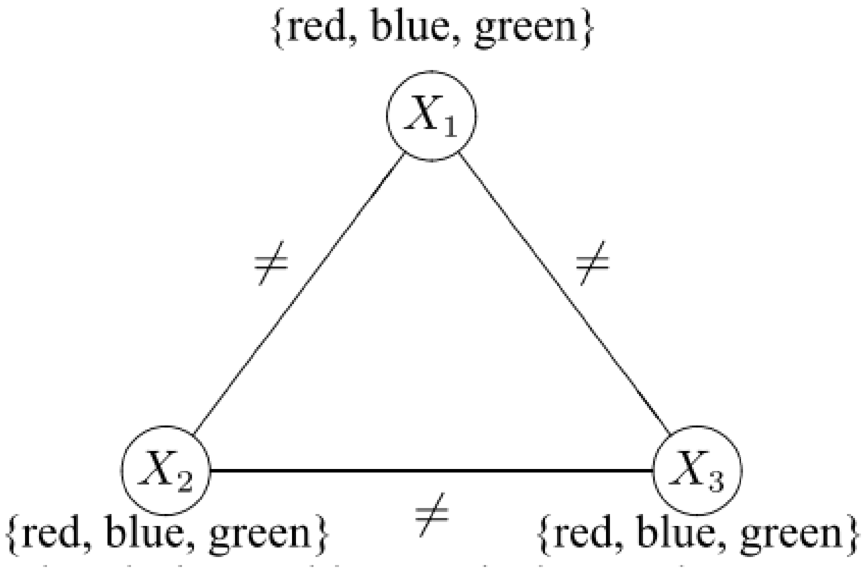
- Set of variables with their domains and constraints on values taken by the variables
- The task: assign values to the variables satisfying all constraints or proclaim that there is no such an assignment
- Several applications: planning, vision, NLP, theorem proving, scheduling

An example: find non-overlapping frequencies for the sensors

- Three sensors
- Overlapping reach
- The task: assign non-overlapping frequencies to the sensors from the domain of allowed frequencies



An example



- Equivalent to graph coloring
- Set of variables X={ X_1, X_2, X_3 }
- Domain D_i for each variables is {red, blue, green}
- Set of constraints { $X_1 \neq X_2$, $X_1 \neq X_3$, $X_3 \neq X_2$}

Constraint satisfaction terminology

- Variable assignment
  - Legal, illegal
- Solution
- Distributed constraint satisfaction: each agent is a variable, the solution is to be found without central control

Domain pruning algorithms

- Nodes communicate with neighbors to prune forbidden values from their domains
- arc consistency algorithm
- Each vertex X_i with domain D_i repeatedly executes the program for each of its neighbors X_j

void revise(x_i, x_j) {

foreach ($v_i \in D_i$)

if (there is no value $v_j \in D_j$ such that v_i is consistent with v_j)

$D_i = D_i - \{v_i\}$

}

Arc consistency

- Stop when one of domains is empty (no solution), or no more eliminations takes place.
- If there is a single value left in each domain, we have a solution; otherwise the result is inconclusive: we do not know if the solution exists.
- Algorithm terminates and is sound (the solution if found, is correct), but it is not complete (no guarantee that the solution will be found).

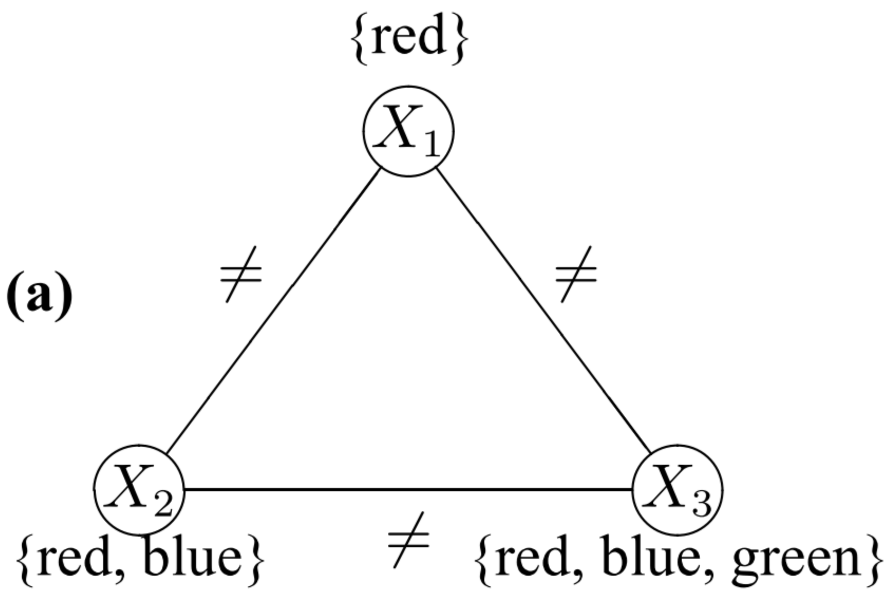
An example of domain pruning a)

- First only messages to node X1 are efficient, therefore X_2 and X_3 eliminate value *red*

X2={blue} X3={blue, green}

- Next X_3 can eliminate *blue*;
- The result is correct

(a)

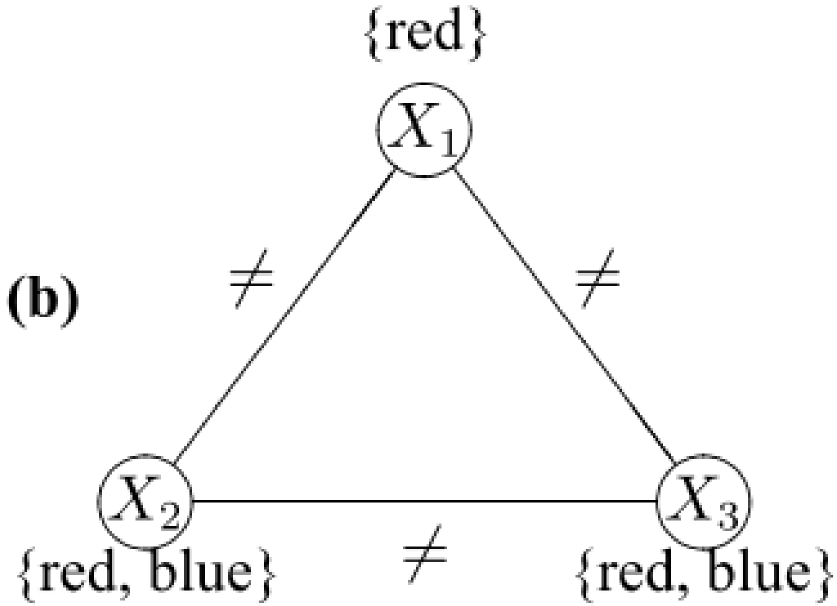


An example of domain pruning b)

- As before X_2 and X_3 eliminate *red*

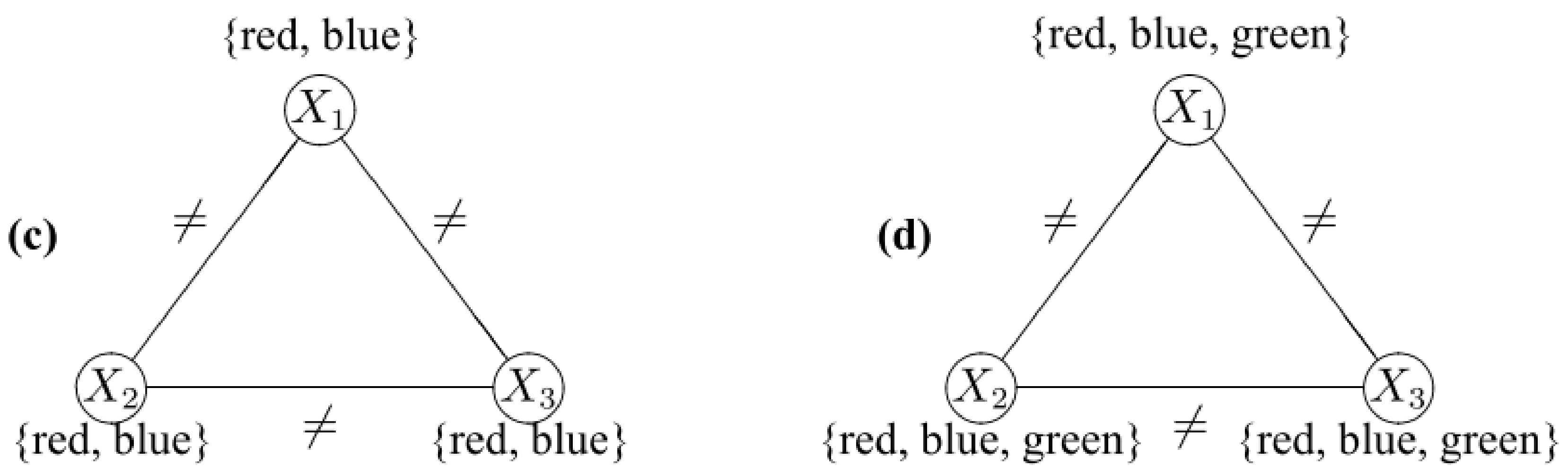
X2={blue} X3={blue}

- Next both X2 and X_3 eliminate *blue;*
- Empty domain is left, so they proclaim there is no solution.



An example of domain pruning c), d)

- No node can eliminate any value
- Inconclusive termination



Equivalence to logic resolution

- Arc consistency is too weak, can be used as preprocessing
- Value elimination is equivalent to unit resolution in propositional logic
- Inference rule

  A1

  $\neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$

  ——————————————————

  $\neg(A_2 \wedge \ldots \wedge A_n)$

- We write constraints as forbidden value combinations called *Nogoods*, e.g., x_1=red $\wedge$ x_2=red

  x_1=red

  $\neg(x_1\text{=red} \wedge x_2\text{=red})$

  ——————————————————

  $\neg(x_2\text{=red})$

Hyper-resolution

- A generalization of unit resolution
  $A_1 \vee A_2 \vee \cdots \vee A_m$
  $\neg(A_1 \wedge A_{1,1} \wedge A_{1,2} \wedge \cdots)$
  $\neg(A_2 \wedge A_{2,1} \wedge A_{2,2} \wedge \cdots)$
  ...
  $\neg(A_m \wedge A_{m,1} \wedge A_{m,2} \wedge \cdots)$
  ————————————————————————
  $\neg(A_{1,1} \wedge \cdots \wedge A_{2,1} \wedge \cdots \wedge A_{m,1} \wedge \cdots)$
- Sound and complete for propositional logic
- at least one of the literals in the top disjunction $A_1 \vee A_2 \vee \cdots \vee A_m$ is true, therefore the conjunction of all the remaining literals in the negated terms has to fail, too

Hyper-resolution algorithm

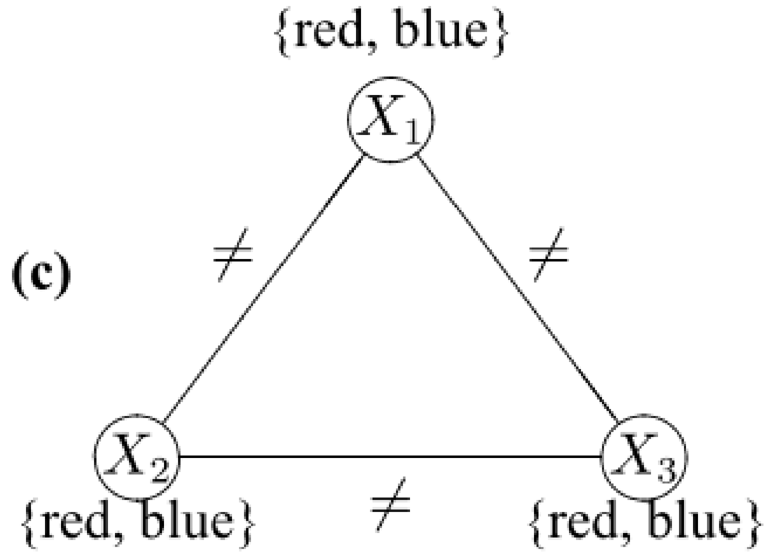
- each agent repeatedly generates new constraints for his neighbors, notifies them of these new constraints, and prunes his own domain based on new constraints passed to him by his neighbors.
- NG_i is the set of all *Nogoods* of which agent i is aware
- NG^*_j is a set of new *Nogoods* communicated from agent *j* to agent *i*.

```
void reviseHR(NG_i, NG*_j) {
    do {
      NG_i ← NG_i ∪ NG*_j
      NG*_i ← hyperresolution(NG_i , D_i)
      if ( NG*_i ≠ {} )
        NG_i ← NG_i ∪ NG*_i
        send the Nogoods NG*_i to all neighbours of i
        if ( {} ∈ NG*_i)
          stop
    } while (there is a change in NG_i)
}
```

algoritem terminates after finite number of steps

If the solution exists, the algorithms finds it

Hyper-resolution for c)

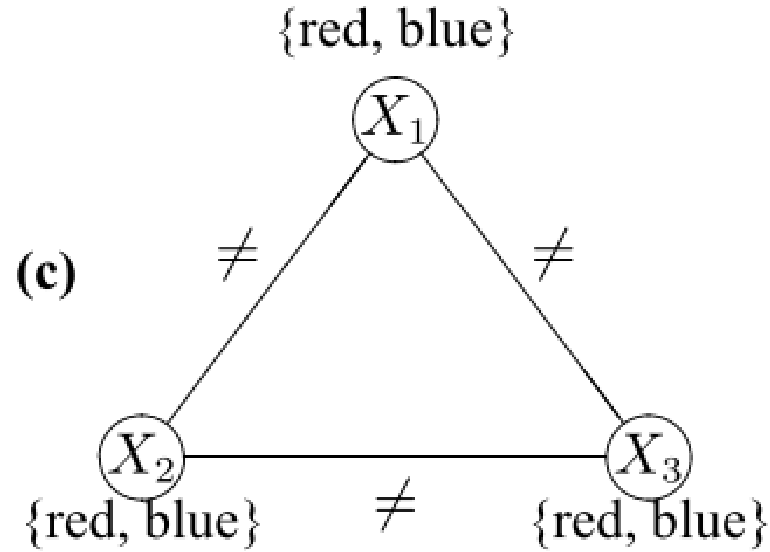


- X_1 has initially the following constraints in its Nogoods:
  $\{x_1 = red, x_2 = red\}$,
  $\{x_1 = red, x_3 = red\}$,
  $\{x_1 = blue, x_2 = blue\}$,
  $\{x_1 = blue, x_3 = blue\}$
- X_1 can be assigned values $x_1 = red \vee x_1 = blue$.
- With hyper-resolution X_1 can reason

$$
\begin{array}{l}
x_1 = red \vee x_1 = blue \\
\neg(x_1 = red \wedge x_2 = red) \\
\neg(x_1 = blue \wedge x_3 = blue) \\
\hline
\neg(x_2 = red \wedge x_3 = blue)
\end{array}
$$

And adds constraints {x2 = red, x3 = blue} to its Nogoods

Hyper-resolution for c)



- Similarly it adds
  $\{x_2 = \text{blue}, x_3 = \text{red}\}$ to its Nogoods
- x_1 sends both Nogoods to its neighbors x_2 and x_3
- x_2 can reason (based on its domain, Nogoods and received inferences)
  $x_2 = \text{red} \vee x_2 = \text{blue}$
  $\neg(x_2 = \text{red} \wedge x_3 = \text{blue})$
  $\neg(x_2 = \text{blue} \wedge x_3 = \text{blue})$
  ———————————————
  $\neg(x_3 = \text{blue})$
- Based on the other received Nogood x_2 constructs $\neg(x_3 = \text{red})$
- When both Nogoods are send to the neighbor x_3, x_3 generates {} and algorithms terminates proclaiming that no solution exists.

Weaknesses of hyper-resolution

- Number of generated Nogoods can be very large
- Asynchronous and parallel processing would generate even more Nogoods.
- The problem lies in the least-commitment nature of these algorithms; they are restricted to removing only provably impossible value combinations.
- The alternative is to explore a subset of the space, making tentative value selections for variables, and backtracking when necessary.

Heuristic search with constraints

- Centralized trial and error
- Sort variables, e.g., $x_1, x_2, \ldots, x_n$
- Call chooseValue(x_1, {}), with values $\{v_1, v_2, \ldots, v_{i-1}\}$ already assigned to $\{x_1, x_2, \ldots, x_{i-1}\}$

procedure ChooseValue($x_i, \{v_1, v_2, \ldots, v_{i-1}\}$)
$v_i \leftarrow$ value from the domain of x_i that is consistent with $\{v_1, v_2, v_{i-1}\}$
if *no such value exists* **then**
    backtrack[1]
else if $i = n$ **then**
    stop
else
    ChooseValue($x_{i+1}, \{v_1, v_2, \ldots, v_i\}$)

Weaknesses of chronological backtracking

- Exhaustive search
- Agents work sequentially

Naïve parallel asynchronous solution

- executed by all agents in parallel and asynchronously

select a value from your domain
repeat
  if *your current value is consistent with the current values of your neighbors, or if none of the values in your domain are consistent with them*
  then
    do nothing
  else
    select a value in your domain that is consistent with those of your neighbors and notify your neighbors of your new value
until *there is no change in your value*

Correct but incomplete solution: it may not terminate, it may not find a solution

Asynchronous backtracking

- We need stronger algorithms with ideas from before: global order and message passing
- ABT (asynchronous backtracking)
- Agents are prioritized, messages pass from higher priority agents to lower priority agents
- Parallel execution
- Agents instantiate their variables concurrently and send their assigned values to the agents that are connected to them by outgoing links. All agents wait for and respond to messages. After each update of his assignment, an agent sends his new assignment along all outgoing links. An agent who receives an assignment (from the higher-priority agent of the link), tries to find an assignment for its variable that does not violate a constraint with the assignment it received

ABT communication

- Agent send messages ok?
- Agents stores received values into his data structure agent_view
- agent checks if his current assignment is consisted with his agent_view.
- If it is, the agent does nothing, otherwise it searchers for a new consistent value
- If the agent finds it, it assigns the found value to a variable and sends ok? message to all connected lower priority agents
- If the agent does not find it, it starts backtracking

ABT - backtracking

- The backtrack operation is executed by sending a Nogood message.
- Nogood is an inconsistent partial assignment (assignments of specific values to some of the variables that together violate the constraints on those variables)
- Nogood consists of A_i's agent_view
- The Nogood is sent to the agent with the lowest priority among the agents whose assignments are included in the inconsistent tuple in the Nogood.
- Agent A_i who sends a Nogood message to agent A_j assumes that A_j will change its assignment. Therefore, A_i removes from his agent_view the assignment of A_j and tries to find an assignment for A_i's variable that is consistent with the updated agent_view.

ABT properties

- Greedy hyper-resolution
- agents make tentative choices of a value for their variables, only generate Nogoods that incorporate values already generated by the agents above them in the order,
- communicates new values only to some agents and new Nogoods to only one agent.

ABT communication

when *received (**Ok?**, (A_j, d_j))* **do**

add (A_j, d_j) to *agent_view*

check_agent_view

when *received (**Nogood**,* nogood*)* **do**

add *nogood* to Nogood list

forall $(A_k, d_k) \in$ nogood, *if* A_k *is not a neighbor of* A_i **do**

add (A_k, d_k) to *agent_view*

request A_k to add A_i as a neighbor

check_agent_view

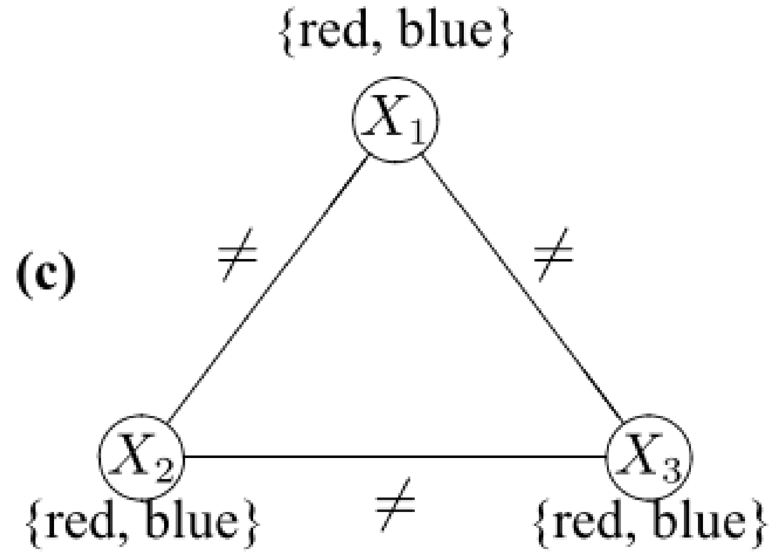
ABT check consistency

```
procedure check_agent_view
when agent_view and current_value are inconsistent do
    if no value in D_i is consistent with agent_view then
        backtrack
    else
        select d ∈ D_i consistent with agent_view
        current_value ← d
        send (ok?, (A_i, d)) to lower-priority neighbors
```

ABT backtracking

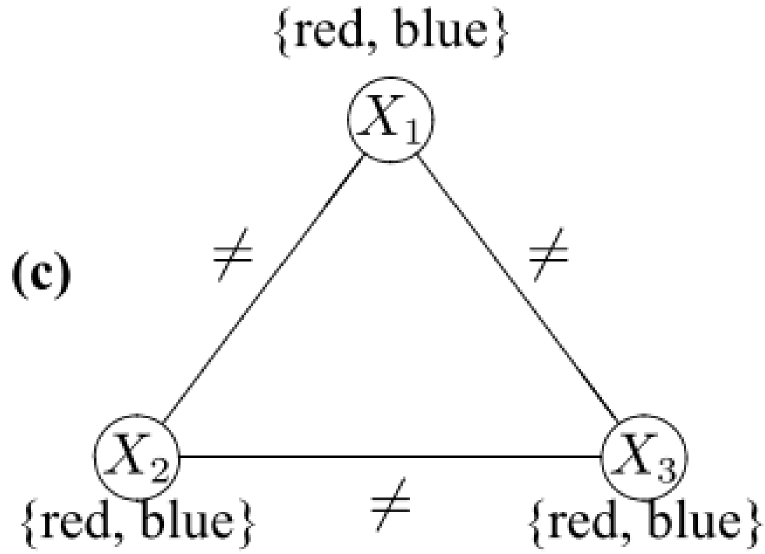
procedure backtrack
nogood $\leftarrow$ some inconsistent set, using hyper-resolution or similar procedure
if nogood *is the empty set* **then**
    broadcast to other agents that there is no solution
    terminate this algorithm
else
    select $(A_j, d_j) \in$ *nogood* where A_j has the lowest priority in *nogood*
    send (**Nogood**, *nogood*) to A_j
    remove (A_j, d_j) from *agent_view*
    check_agent_view

ABT for c)



- Priority x_1, x_2, x_3
- They initially all start with a random value, e.g., all "blue"
- x_1 informs x_2 and x_3, x_2 informs x_3
  x_2 adds to its agent_view {x_1=blue}, x_3 adds {x_1=blue, x_2=blue}.
- x_2 and x_3 have to check consistency with their own value
  - X_2 detects conflict, modifies its value to "red" and informs x_3
  - In that time x_3 detects conflict , modifies its value to "red", informs no one
  - x_3 receives second message from x_2 and modifies its agent_view to {x_1 = blue, x_2 = red}.

Asynchronous backtracking for c)



- x_3 cannot find consistent value so using hyper-resolution it generates Nogood {x_1 = blue, x_2 = red}
- Sends this Nogood to x_2, because of lowest priority in Nogood
- Now x_2 cannot find consistent values and generates Nogood {x_1 = blue} and sends it to x_1.
- x_1 detects inconsistency, modifies its value to "red" and informs x_2 and x_3
- As before, x_2 modifies its value to blue, x_3 cannot find consistent value and generates Nogood {x_1 = red, x_2 = blue},
- After that x_2 generates Nogood {x_1 = red} and sends it to x_1
- Now x_1 has Nogood {x_1 = blue} and {x_1 = red}, uses hyper-resolution to generate Nogood {}. Algorithm terminates by proclaiming that no solution exists.

Distributed Optimization

- agents shall, in a distributed fashion, optimize a global objective function
- We illustrate **distributed path planning in directed graph** with n nodes and m edges
- edge (a,b) has assigned a cost $c(a,b)$;
- objective: find a minimal cost path from the starting node s to any of the goal nodes $t \in T$.
- Applications: transport, telecommunications, planning
- Difference to standard algorithms (Dijkstra, Bellman-Ford) is a distributed approach (agents communicate only locally, each agent contributes to the globally optimal solution)

Asynchronous dynamic programming

- dynamic programming (incremental divide and conquer)
- if node x lies on a shortest path from s to t, then the portion of the path from s to x (and from x to t) must also be the shortest paths between s and x (x and t
- the shortest distance from any node *i* to the goal node *t* is represented with $h^*(i)$.
- Shortest path from *i* to *t* via neighboring node *j*
  $f^*(i, j) = c(i, j) + h^*(j)$
- Shortest path from *i* via arbitrary neighboring node

  $h^*(i) = \min_j f^*(i, j)$

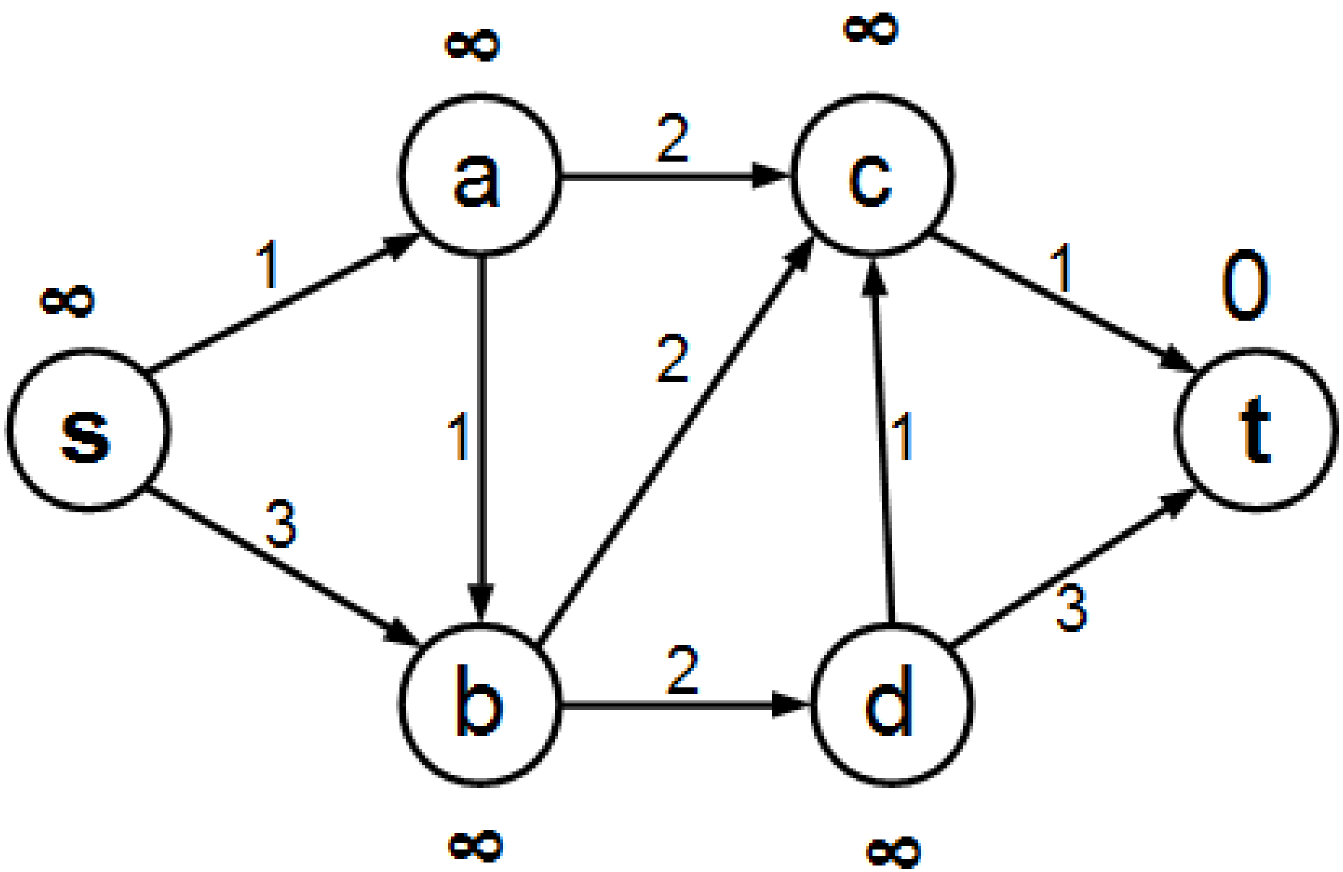
Algorithm details

- Every node i stores a value $h(i)$, as an approximation to $h^*(i)$
- Initialization, each h(i)=∞,
- During execution, the h(i) values decrease and converge to the $h^*(i)$
- Convergence takes one step for every node on the shortest path
- Weakness: we need an agent for every node

Pseudo code ADP for shortest path executed on every node

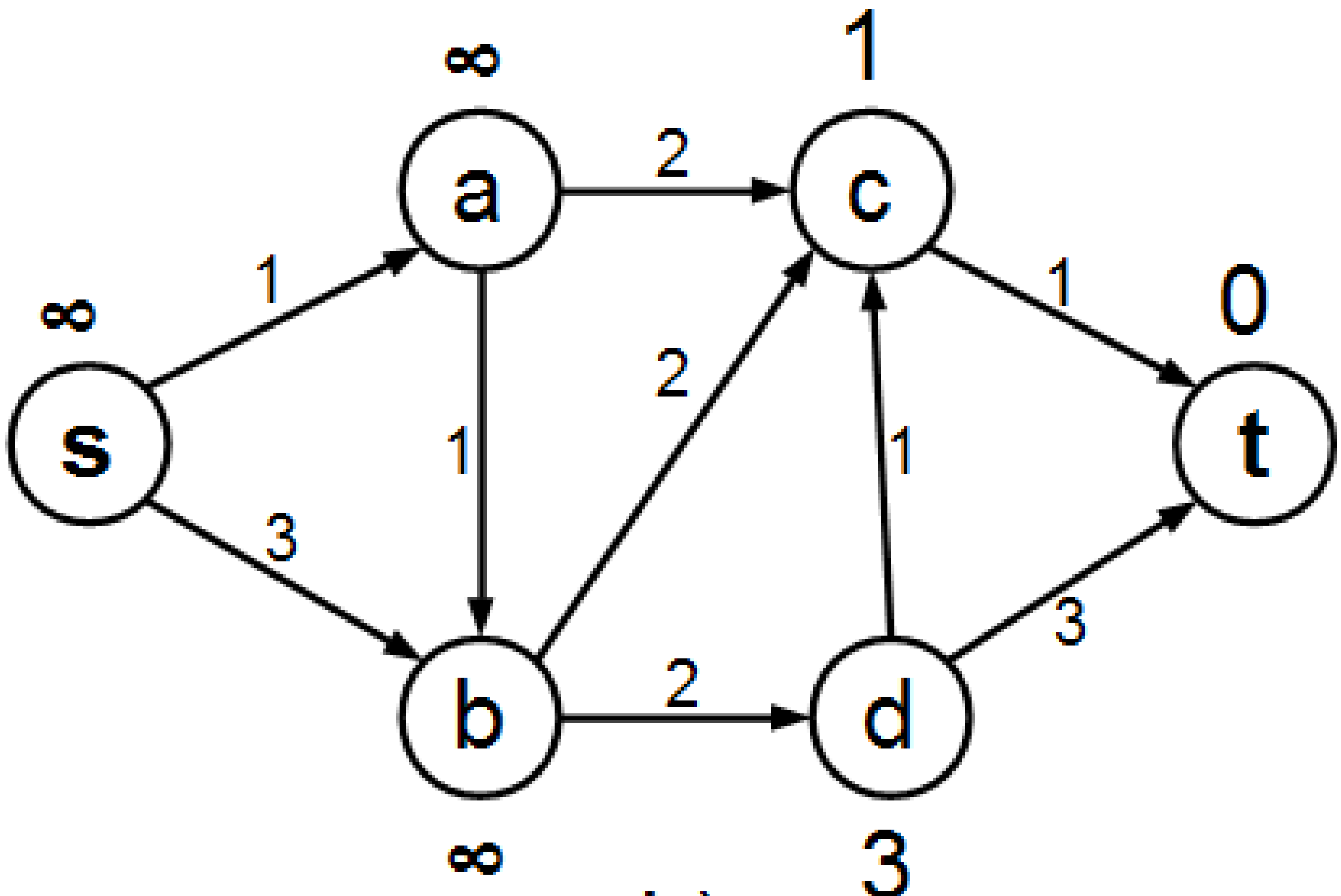
procedure ASYNCHDP (node i)
if *i is a goal node* **then**
    $h(i) \leftarrow 0$
else
    initialize $h(i)$ arbitrarily (e.g., to ∞ or 0)
repeat
    forall *neighbors j* **do**
        $f(j) \leftarrow w(i, j) + h(j)$
    $h(i) \leftarrow \min_j f(j)$

ADP 1/4



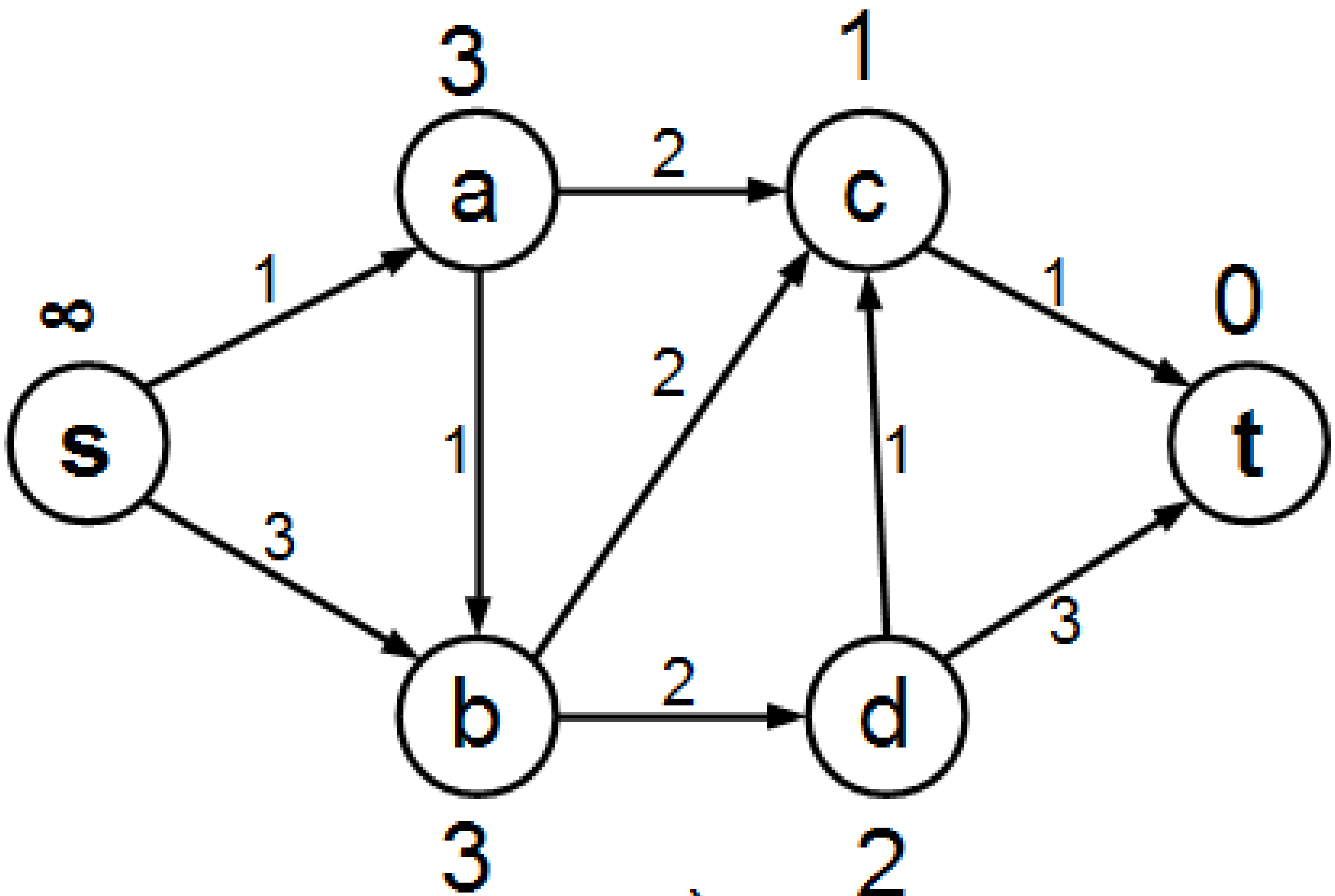
a)

ADP 2/4



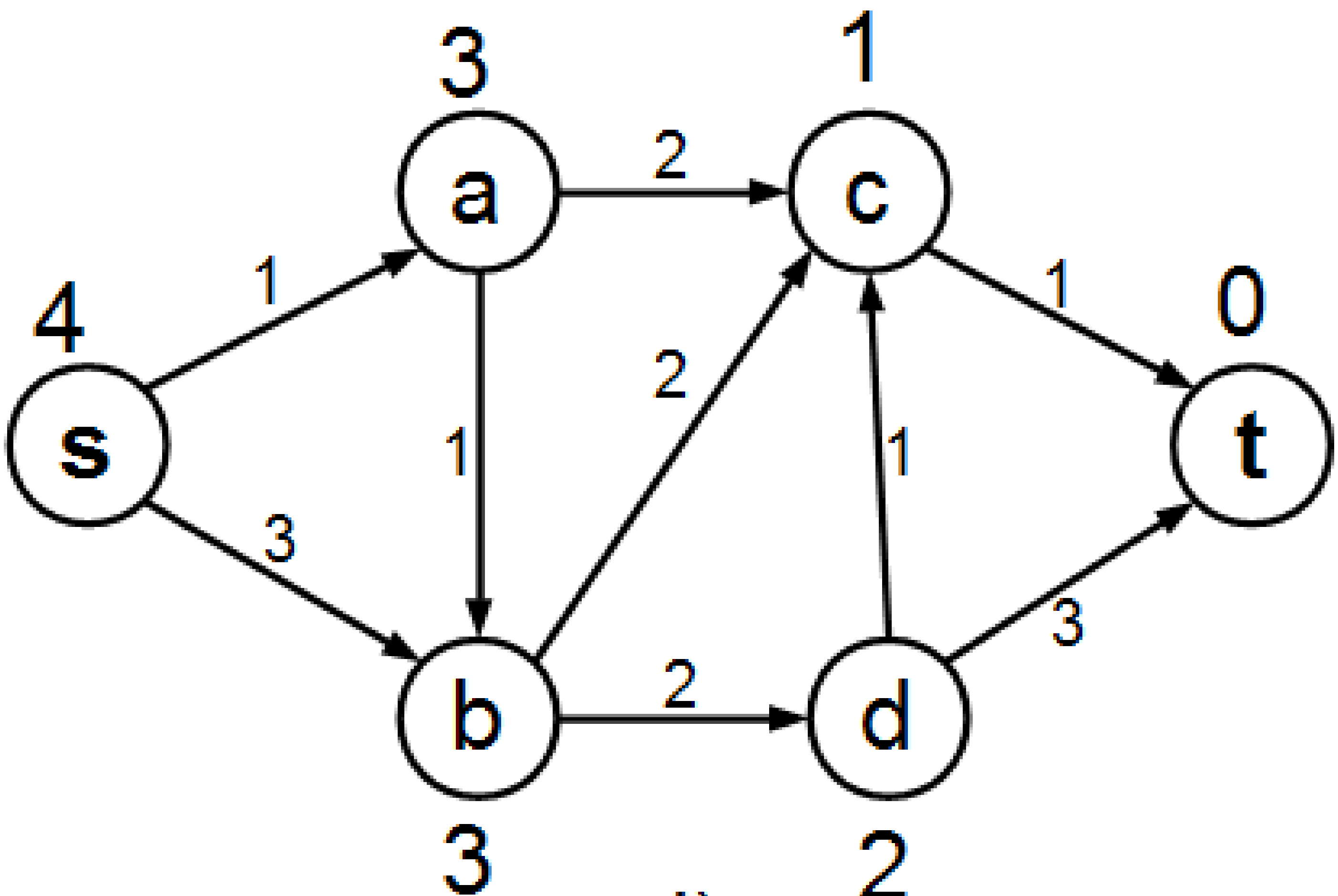
b)

ADP 3/4



c)

ADP 4/4



d)

*LRTA**

- Algorithm *LRTA** (*learning real-time A**) uses one or more agents
- Heuristic search with improved heuristic
- Initialize: $h(i)=0$ (or any better informed admissible heuristics)
- Agent repeatedly executes an algorithm improving $h(i)$
- An example: a single agent execution

Pseudo code of LRTA*

procedure LRTA*

$i \leftarrow s$

while *i is not a goal node* **do**

  foreach *neighbor j* **do**

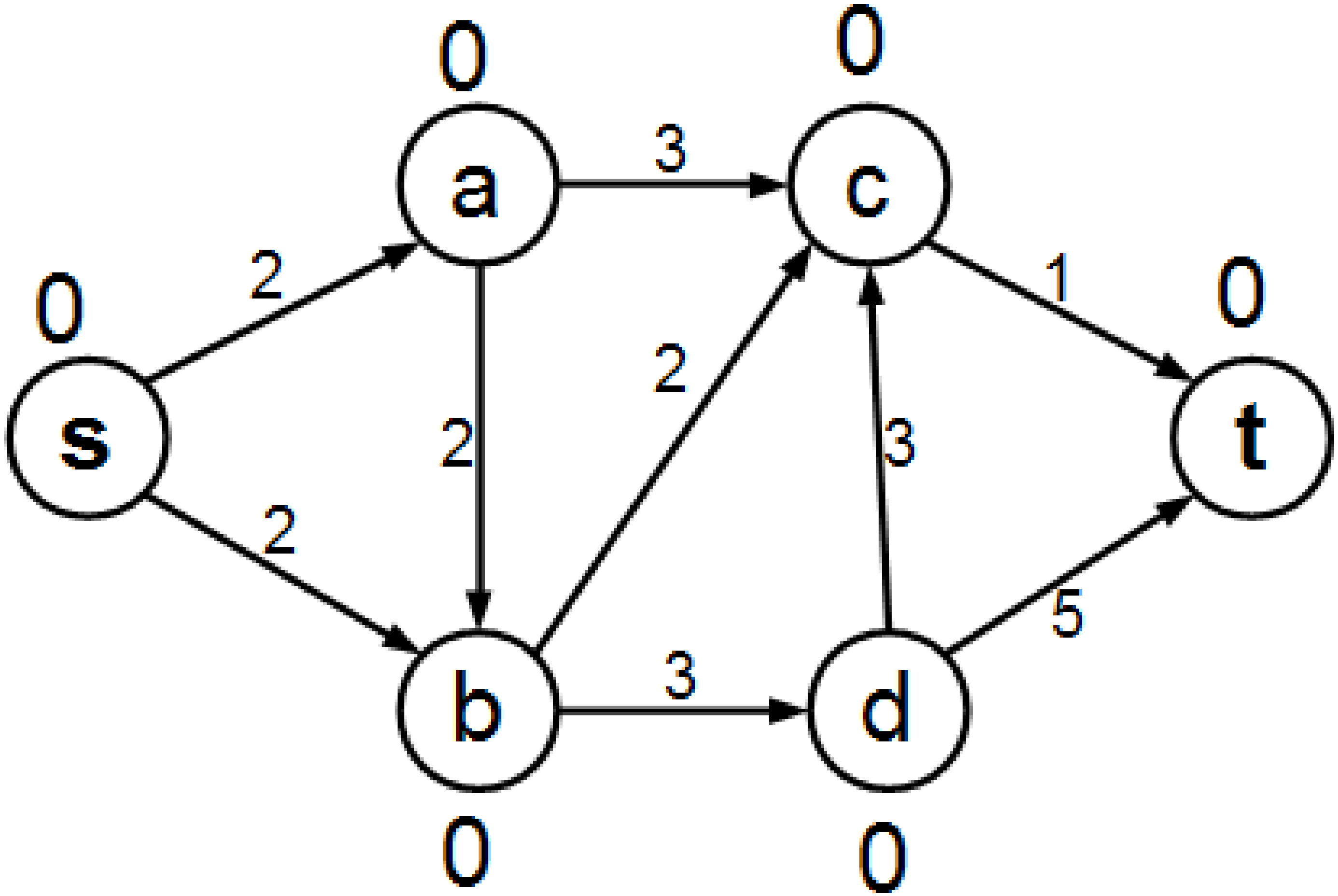
    $f(j) \leftarrow w(i,j) + h(j)$

  $i' \leftarrow \arg\min_j f(j)$

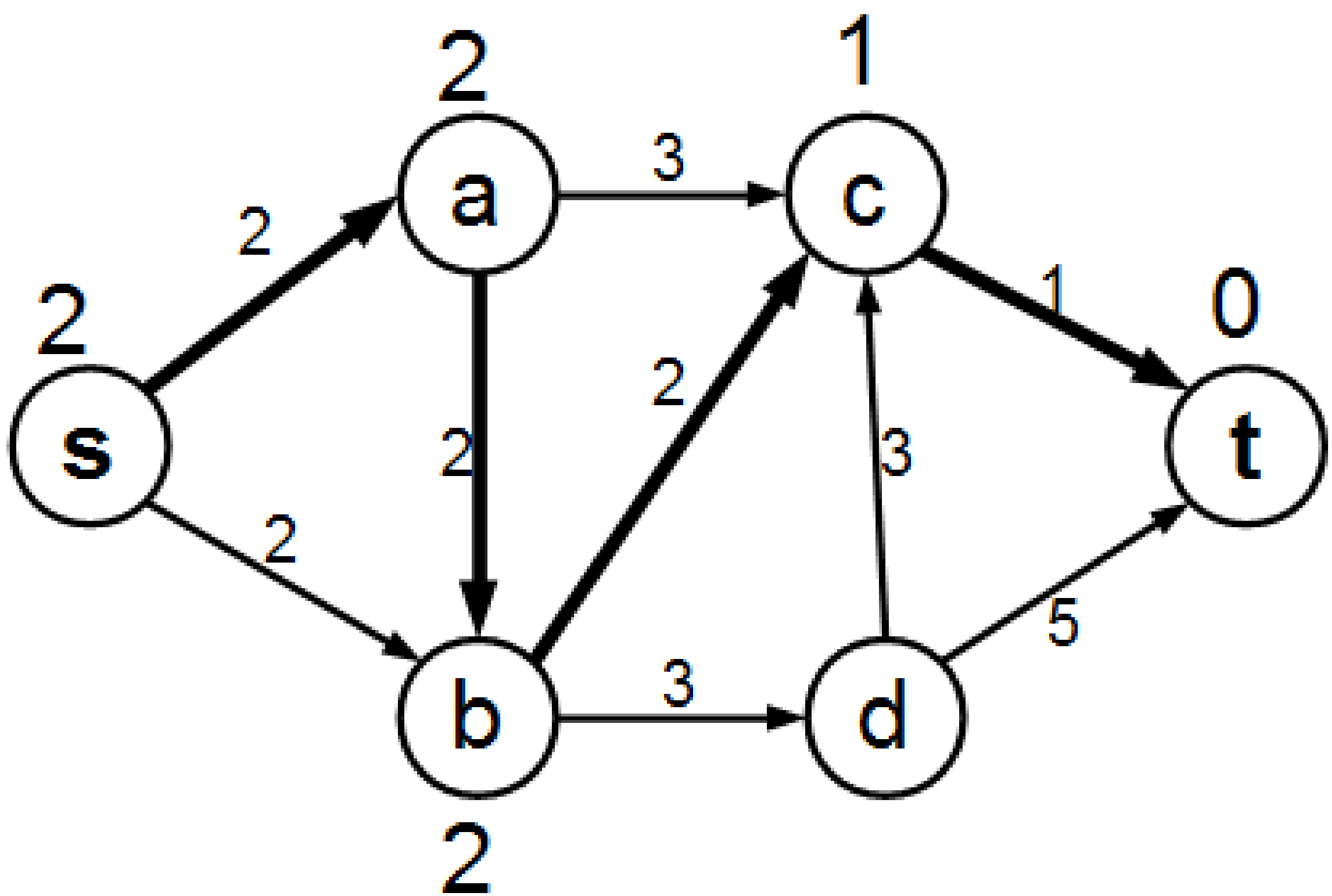
  $h(i) \leftarrow \max(h(i), f(i'))$

  $i \leftarrow i'$

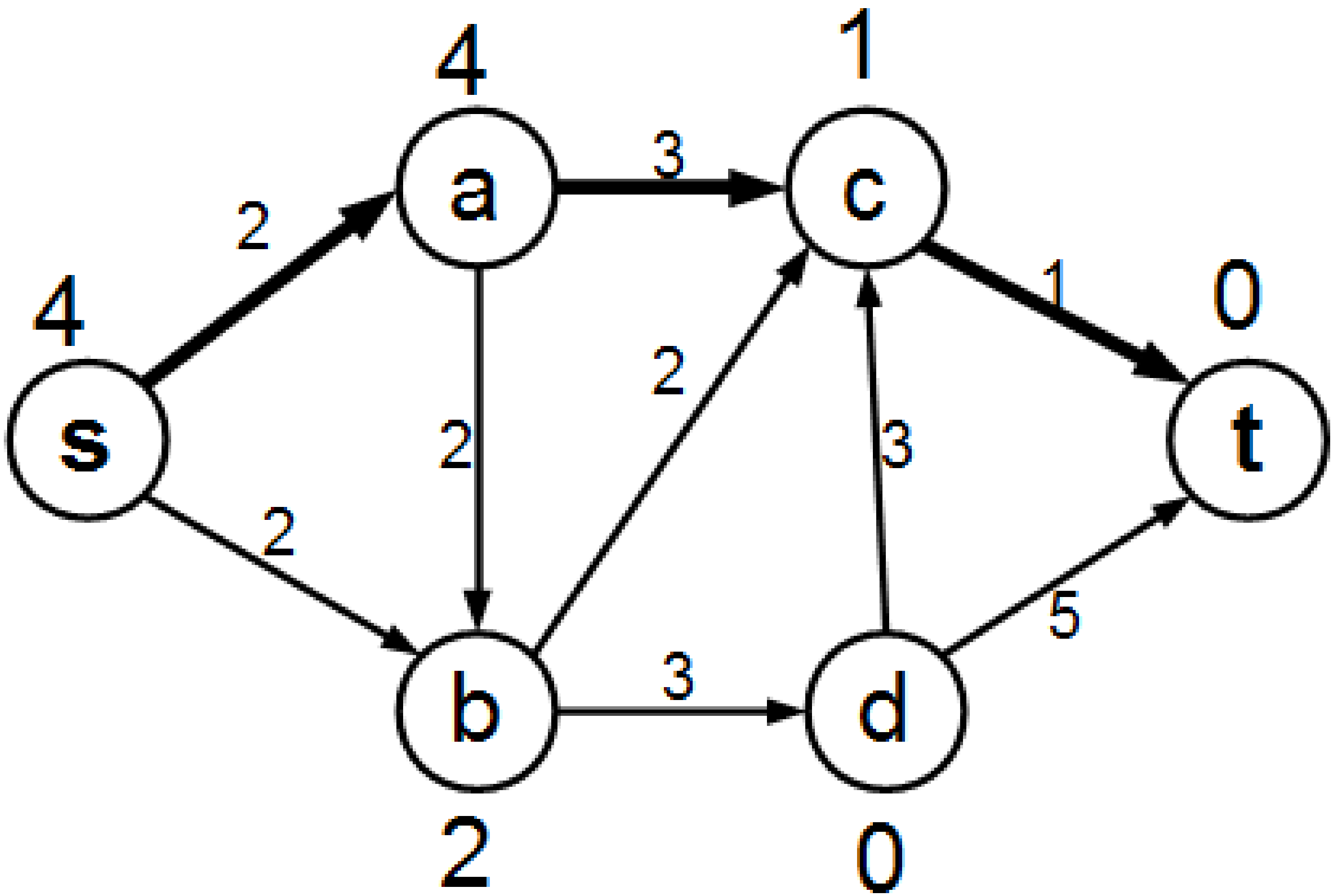
An example:



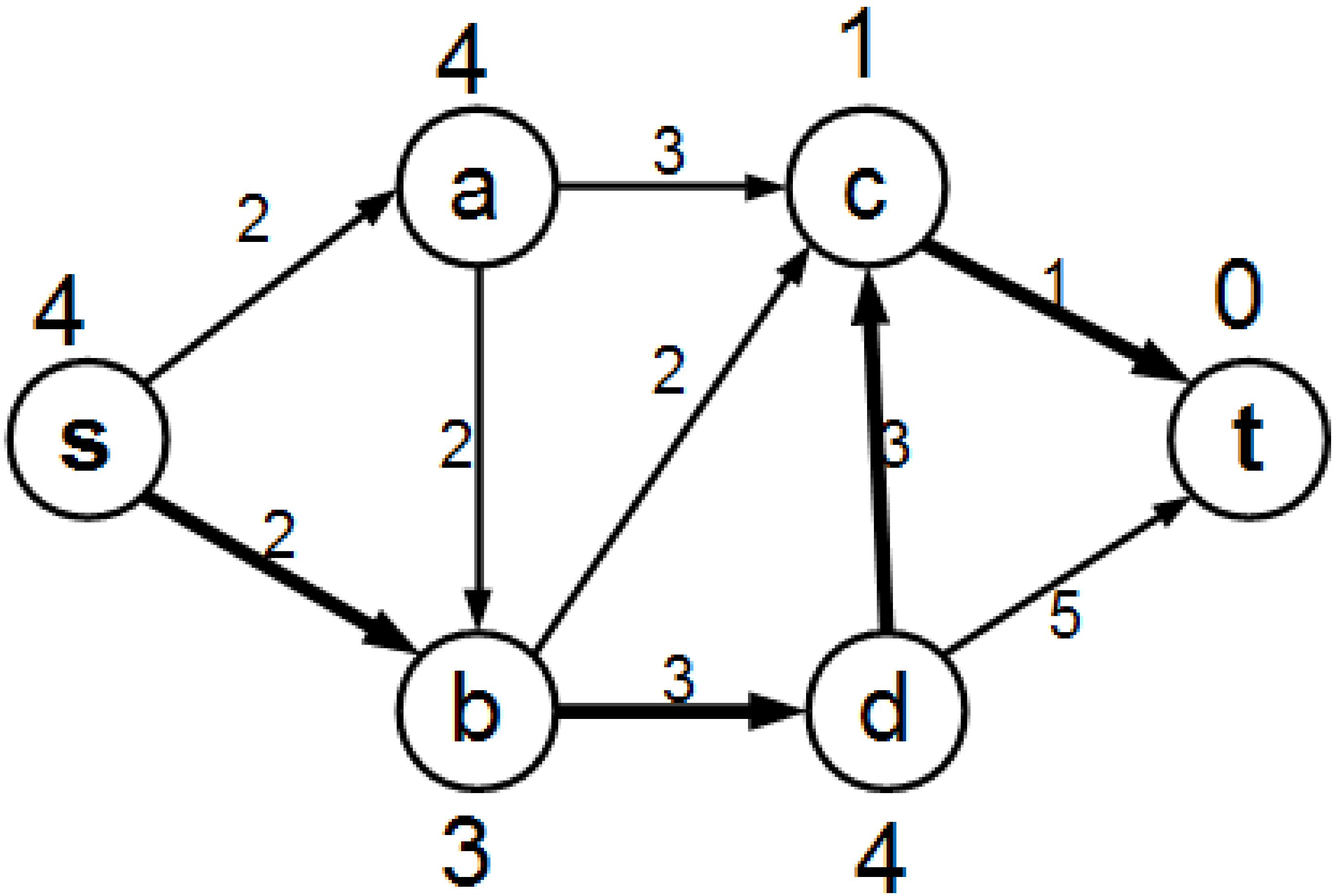
One LRTA* agent 1/4



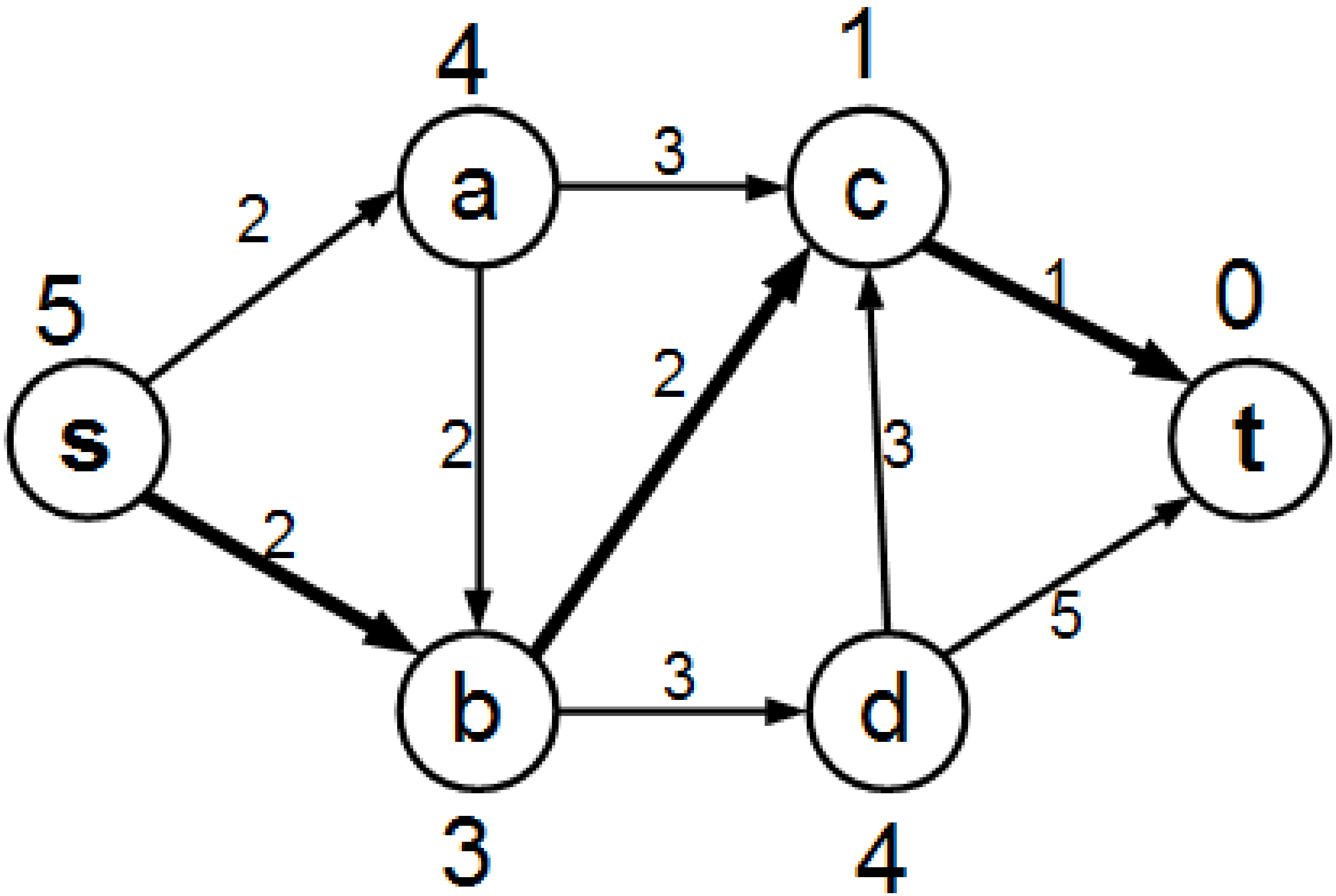
One LRTA* agent 2/4



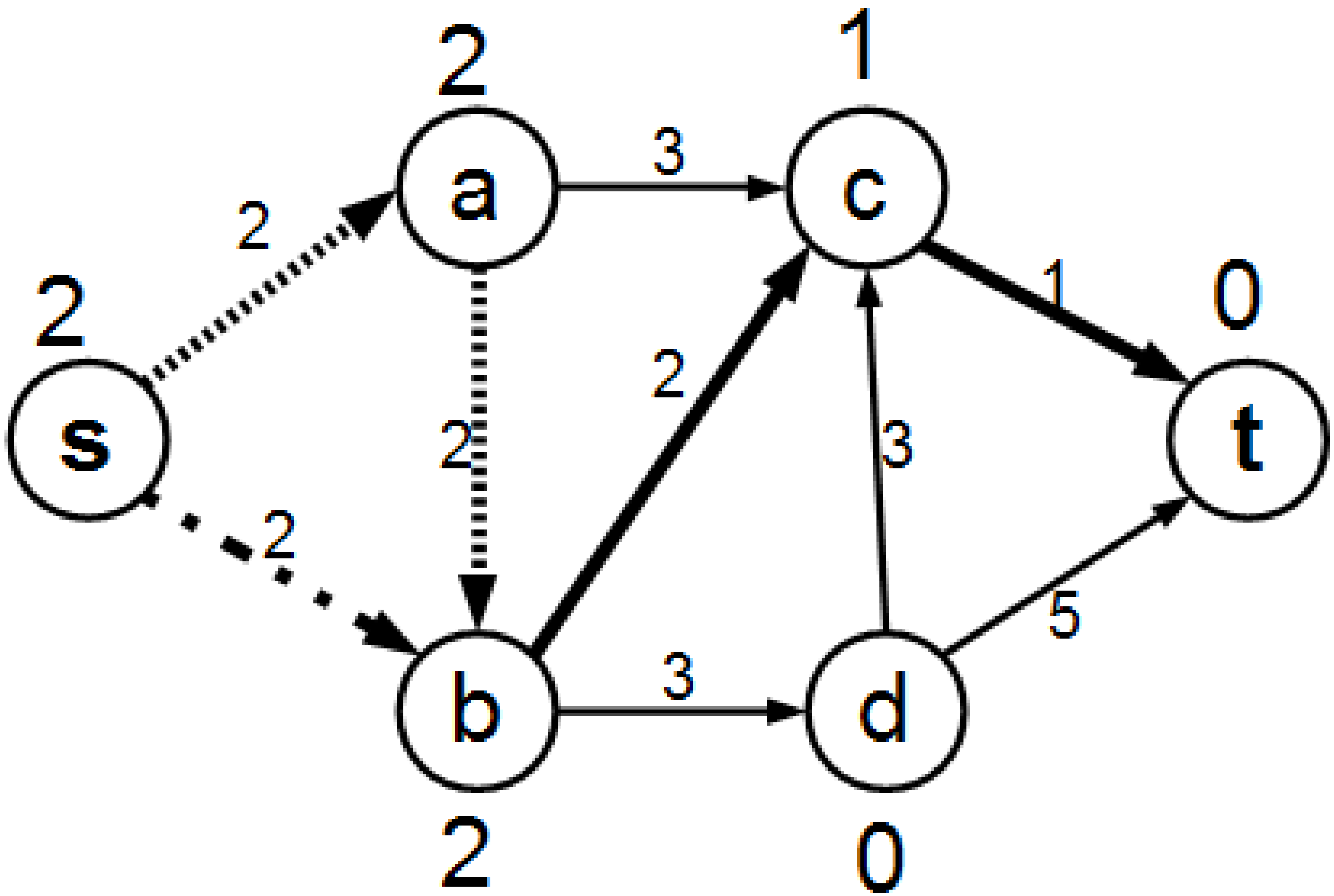
One LRTA* agent 3/4



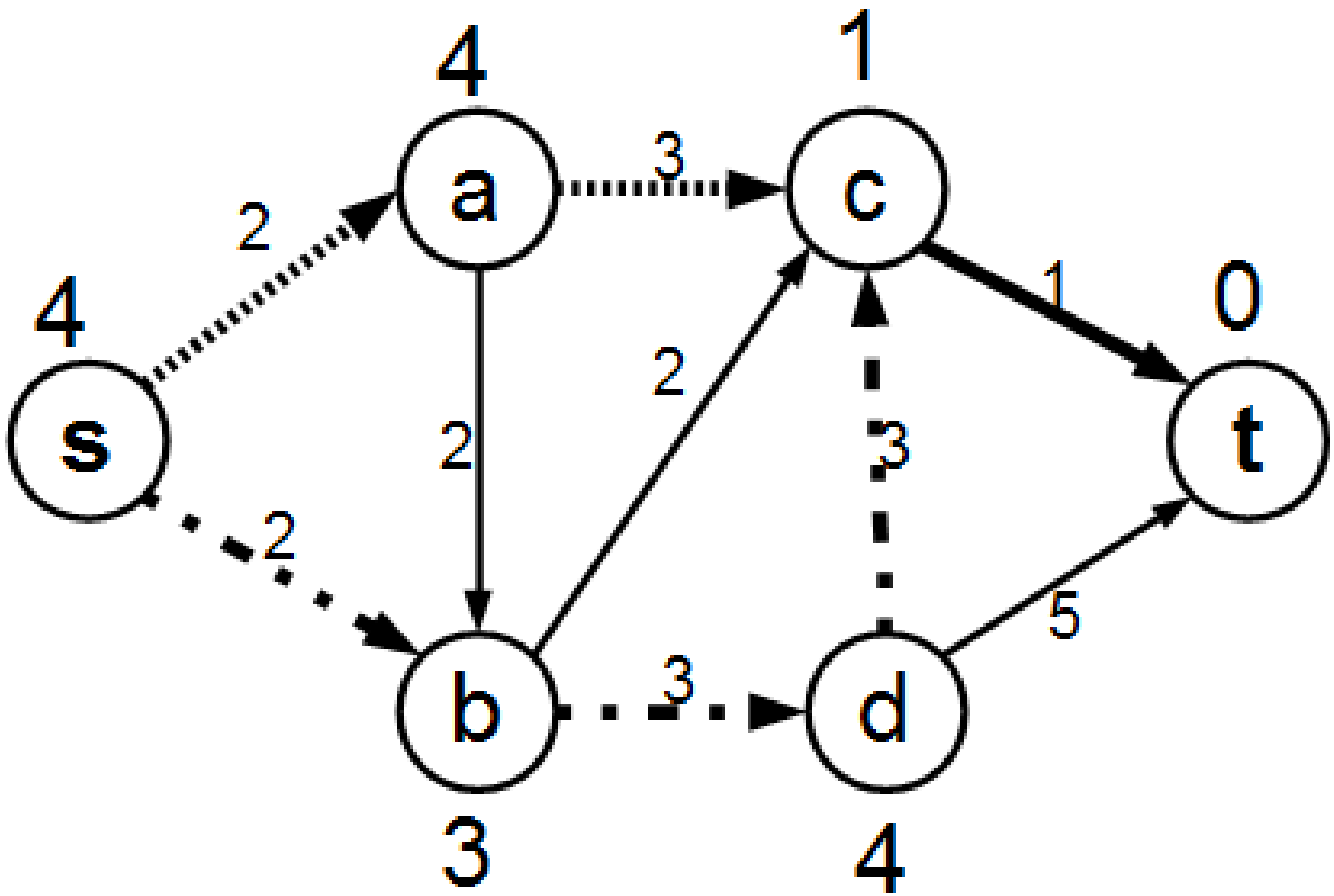
One LRTA* agent 4/4



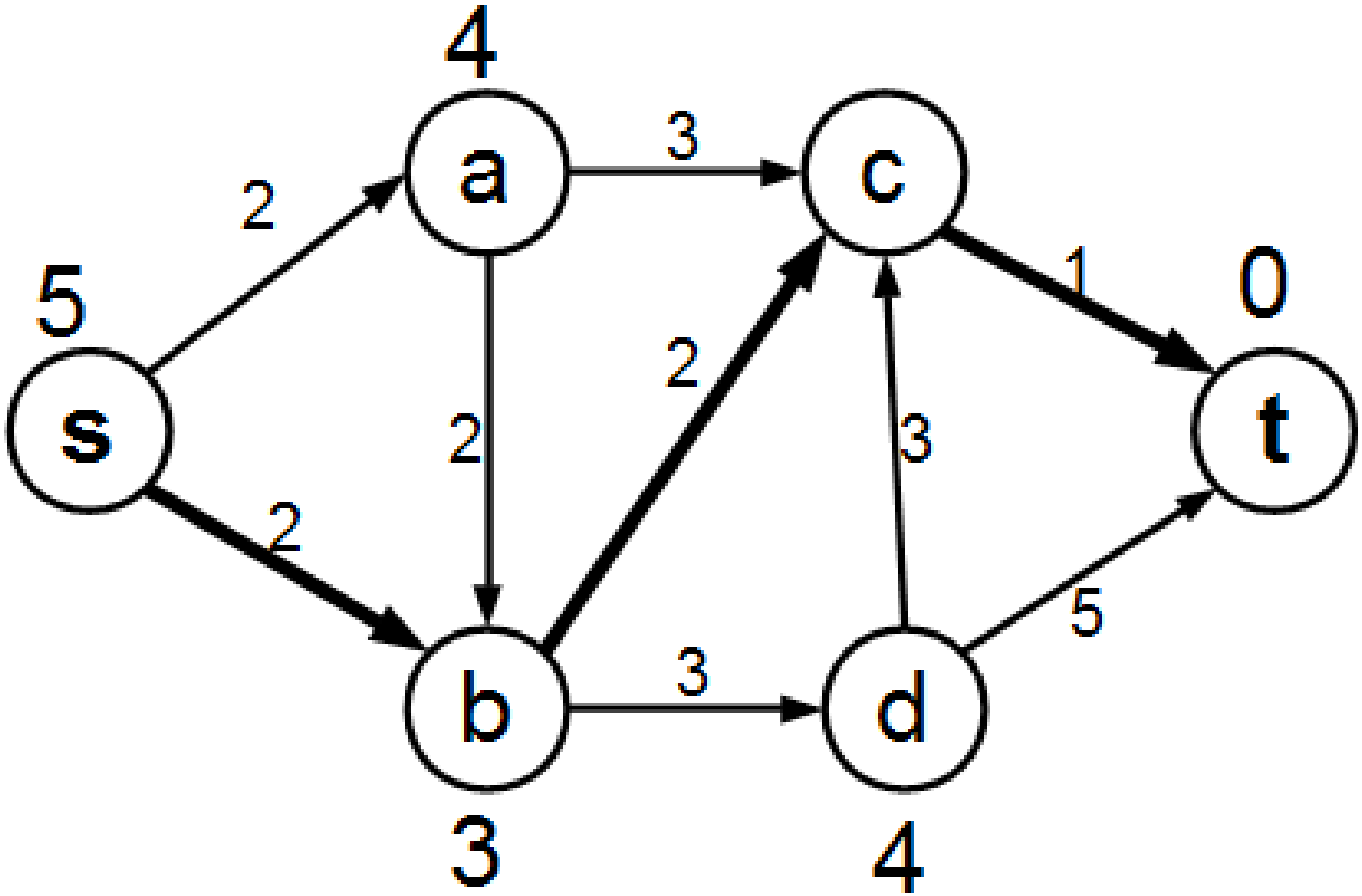
LRTA*(2) (two agents) 1/3



LRTA*(2) (two agents) 2/3



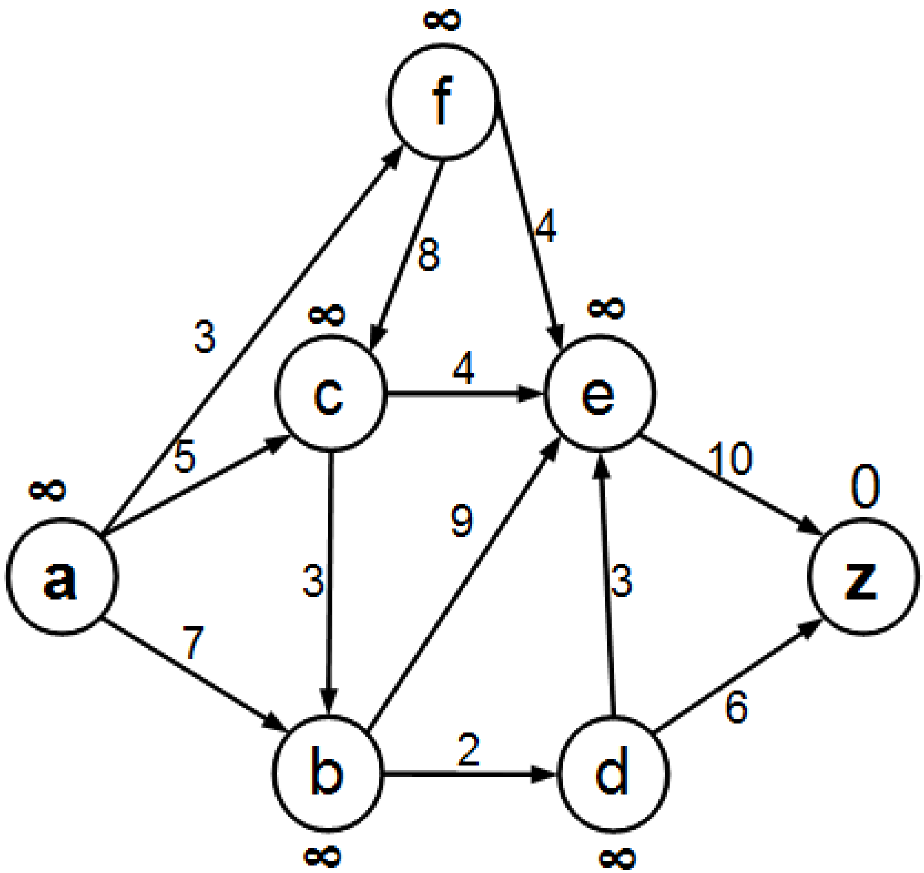
LRTA*(2) (two agents) 3/3



Multiagent technologies

- Extensions to asynchronous backtracking
- Constraint satisfaction optimization
- Learning agents
- Game theory (cooperative and non-cooperative games

An exercise: simulate an execution of ADP and LRTA*



Best reponse dynamics

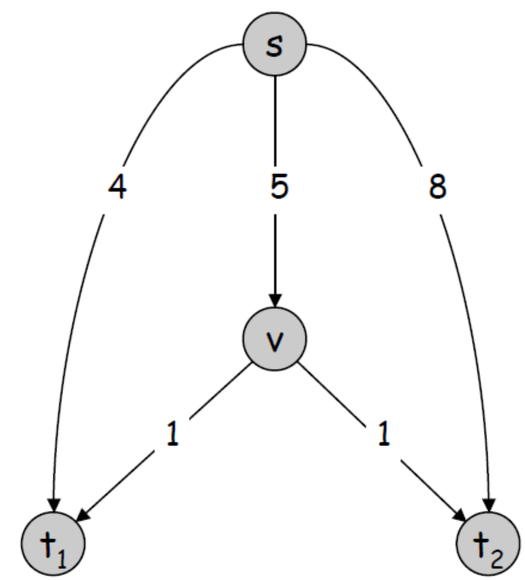
- multicast routing problem
- each agents searches the best solution for himself (selfishness)
- slides from J. Kleinberg, E. Tardos: Algorithm Design. Pearson, 2006 (chapter 12)

Multicast Routing

Multicast routing. Given a directed graph $G = (V, E)$ with edge costs $c_e \geq 0$, a source node s, and k agents located at terminal nodes $t_1, \ldots, t_k$. Agent j must construct a path P_j from node s to its terminal t_j.

Fair share. If x agents use edge e, they each pay c_e / x.

1	2	1 pays	2 pays
outer	outer	4	8
outer	middle	4	5 + 1
middle	outer	5 + 1	8
middle	middle	5/2 + 1	5/2 + 1



Nash Equilibrium

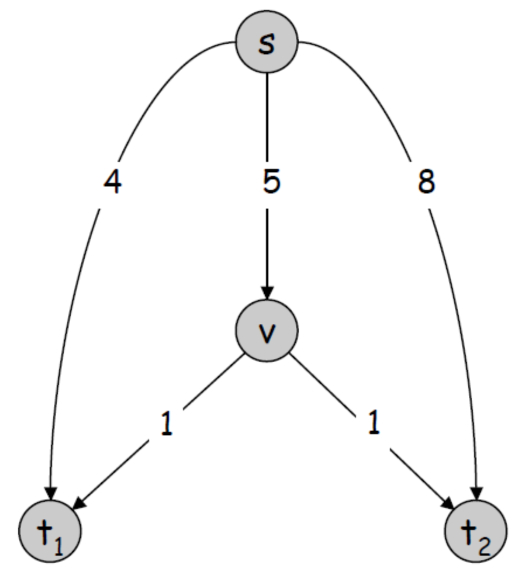
Best response dynamics. Each agent is continually prepared to improve its solution in response to changes made by other agents.

Nash equilibrium. Solution where no agent has an incentive to switch.

Fundamental question. When do Nash equilibria exist?

Ex:

- Two agents start with outer paths.
- Agent 1 has no incentive to switch paths (since $4 < 5 + 1$), but agent 2 does (since $8 > 5 + 1$).
- Once this happens, agent 1 prefers middle path (since $4 > 5/2 + 1$).
- Both agents using middle path is a Nash equilibrium.



Directing multiple agents

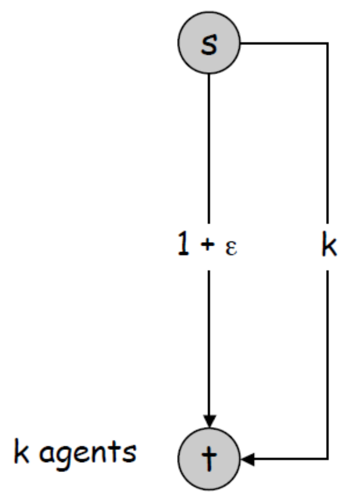
```
Best-Response-Dynamics(G, c) {
   Pick a path for each agent

   while (not a Nash equilibrium) {
      Pick an agent i who can improve by switching paths
      Switch path of agent i
   }
}
```

- provable that the algorithm reaches the Nash equilibrium
- we define a function which strictly decreases in each step

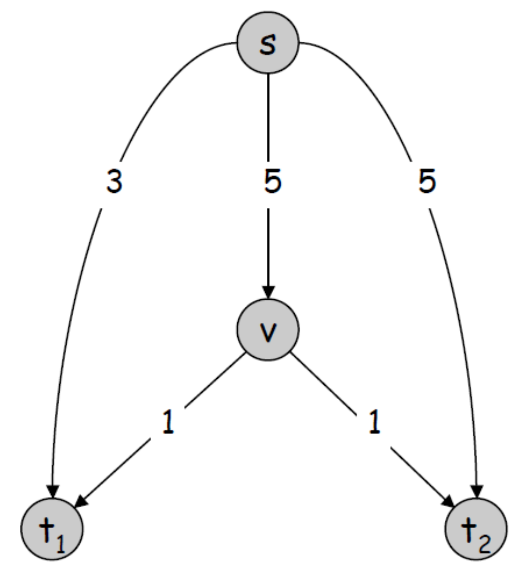
Socially Optimum

Social optimum. Minimizes total cost to all agent.

Observation. In general, there can be many Nash equilibria. Even when its unique, it does not necessarily equal the social optimum.



Social optimum = $1 + \varepsilon$
Nash equilibrium A = $1 + \varepsilon$
Nash equilibrium B = k



Social optimum = 7
Unique Nash equilibrium = 8

Price of Stability

Price of stability. Ratio of best Nash equilibrium to social optimum.

Fundamental question. What is price of stability?

Ex: Price of stability = $\Theta(\log k)$.
Social optimum. Everyone takes bottom paths.
Unique Nash equilibrium. Everyone takes top paths.
Price of stability. $\underbrace{H(k)}_{1 + 1/2 + \ldots + 1/k} / (1 + \varepsilon)$.