MAXIMUM GAIN MESSAGE ALGORITHM

Distributed Algorithm for DCOP: A graphical-game-based approach (Maheswaran et al. 2004).

Outline

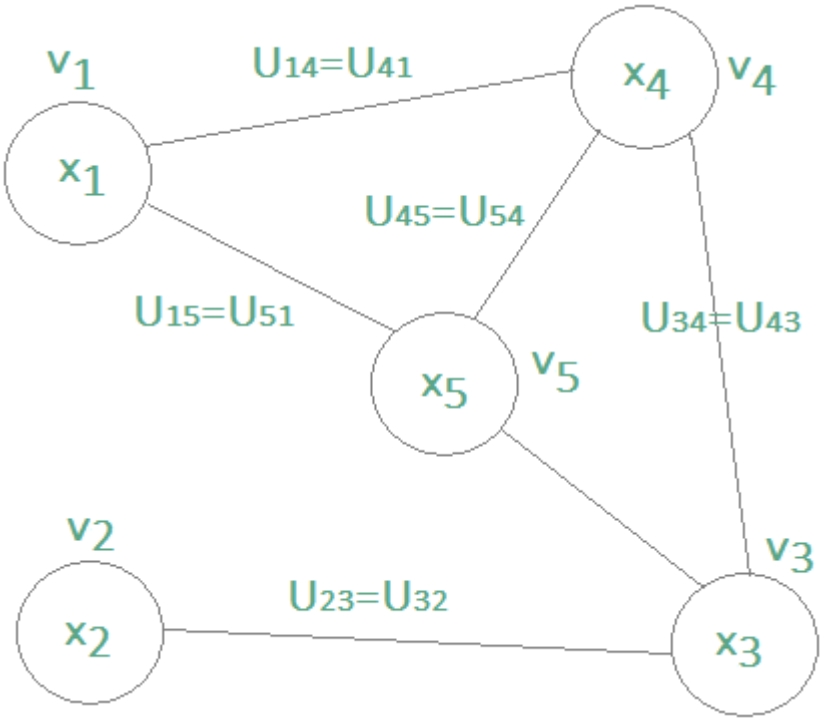
- Brief introduction
- Graphical-game theoretic framework
- MGM and main idea
- DSA and comparison with MGM
- Complexity of MGM and DSA
- MGM-2 and comparison with MGM
- Conclusion
- Software demonstration (maybe)

Local greedy approximate algorithms

- We are dealing with a NP-hard problem: complete methods are often prohibitive for practical applications
- Approximate algorithms are often preferred, as they require very little local computation and communication
- A local greedy search starts with a random assignment for all the variables and then performs a series of local moves trying to greedily optimize the objective function.
- See chapter 12 section 5.1 of the book by G. Weiss, *Multiagent Systems*

Decomposition of a DCOP into an equivalent graphical game

- Players are the variables.
- The actions are the choices of values.
- The information state is the context consisting of neighbor's values.
- Local utility functions are constructed from the constraint utility functions.



Preliminary concepts

- Given two nodes (variables), if there exists an edge between them, then first node is a **neighbor** of the second and vice-versa
- Let us define $x_{-i} \equiv [x_{j1}, \ldots, x_{jK_i}]$, hereby referred to as a **context**, be a tuple which captures the values assigned to the neighboring variables of the $\boldsymbol{i}$-th variable
- We now define a **local utility** for the i-th agent (or equivalently the i-th variable) as: $u_i(x_i; x_{-i}) \equiv \sum_{j \in \mathcal{N}_i} U_{ij}(x_i, x_j)$

Preliminary concepts

- A **Nash equilibrium** assignment is a tuple of values $\hat{x} \in X$ where no agent can improve its local utility by unilaterally changing its value given its current context: $\hat{x} \in \arg\max_{x_i \in X_i} u_i(x_i; \hat{x}_{-i}), \forall i \in \mathcal{N}$
- Given a DCOP game, let $\boldsymbol{X_{NE}} \subseteq X$ be the subset of the assignment space which captures all Nash equilibrium assignments:

$$X_{NE} \equiv \left\{ \hat{x} \in X : \hat{x} \in \arg\max_{x_i \in X_i} u_i(x_i; \hat{x}_{-i}), \forall i \right\}$$

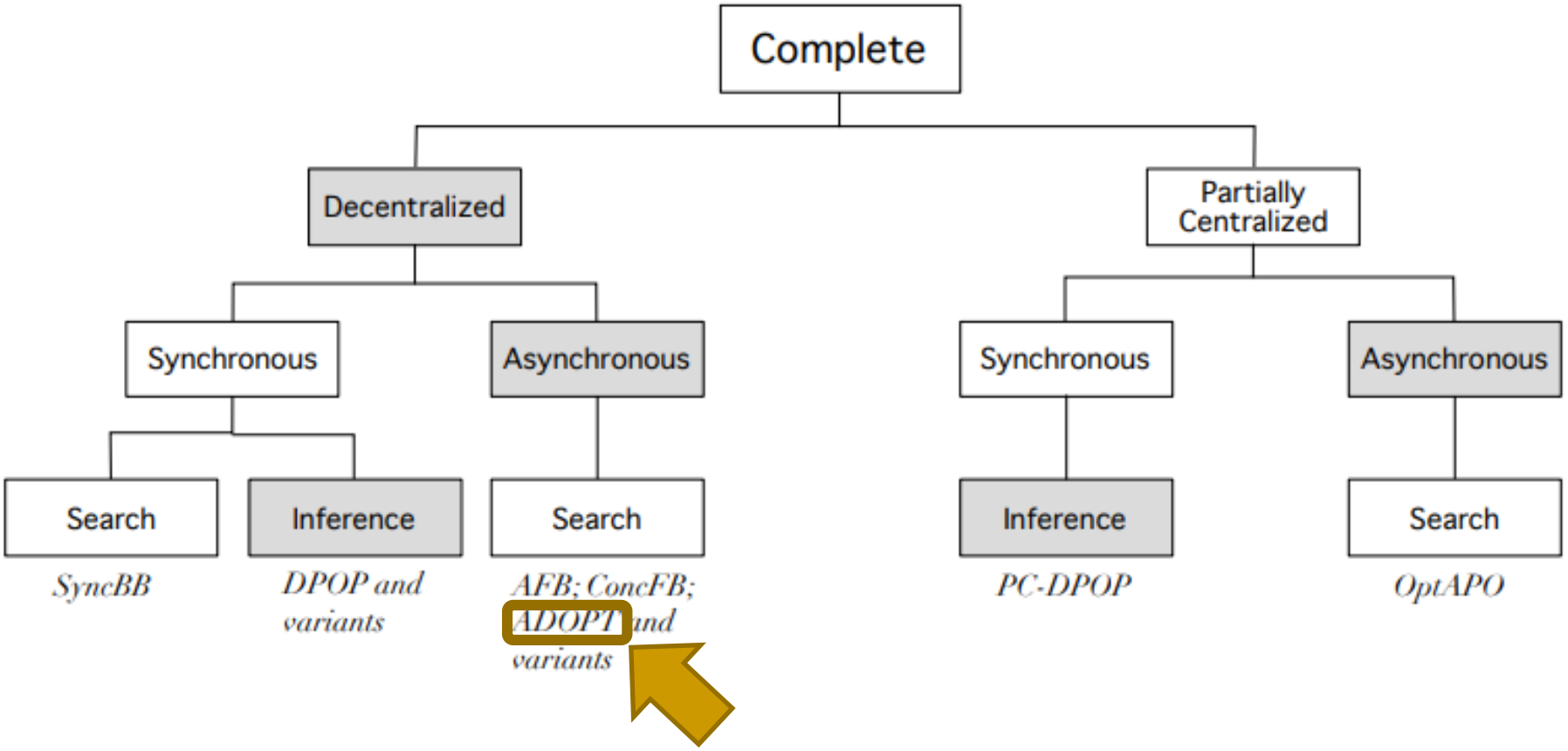
Preliminary concepts

- Proposition: The assignment x^* which optimizes the DCOP characterized is also a Nash equilibrium with respect to the associated graphical game.

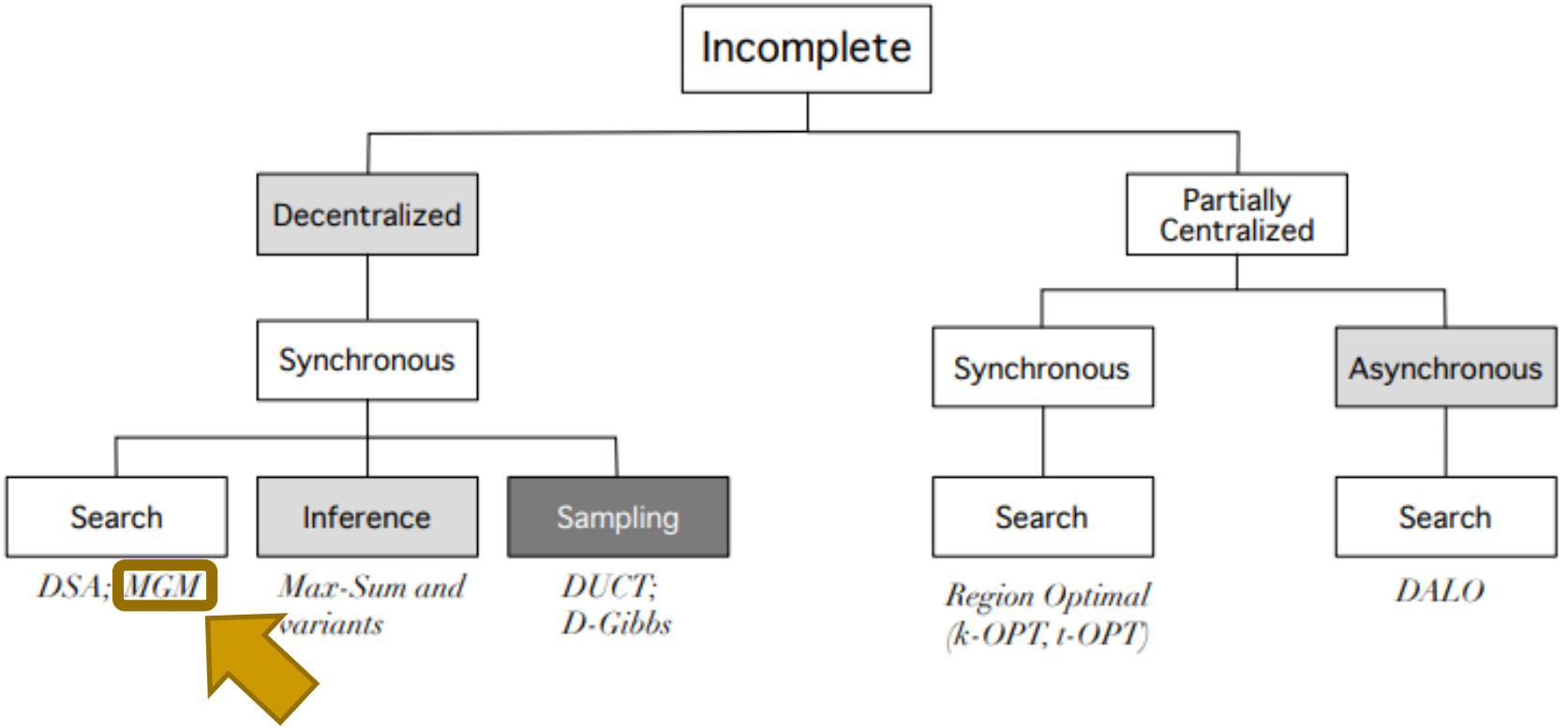
Main idea of MGM

- An approach to address the possible out-of-date knowledge about other agents' variable values, is for neighboring agents to **agree** on who is the agent that can perform a move, while the others wait without changing their values.
- See chapter 12 section 5.1.2 of the book by G. Weiss, *Multiagent Systems*

ADOPT classification



MGM classification



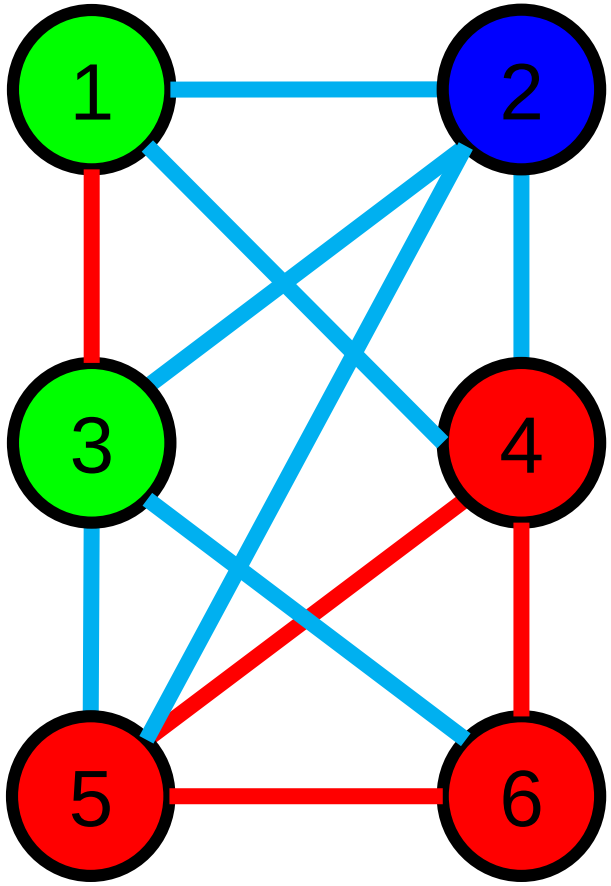
MGM is a synchronous algorithm

- Asynchronous algorithms allow agents to update the assignment for their variables based solely on their local view of the problem, and thus independently from the actual decisions of the other agents. In contrast, synchronous algorithms constrain the agents decisions to follow a particular order, typically enforced by the representation structure adopted.
- Synchronous algorithms tend to delay the actions of some agents guaranteeing that their local view of the problem is always consistent with that of the other agents. In contrast, asynchronous algorithms tend to minimize the idle-time of the agents, which in turn can react quickly to each message being processed; however, they provide no guarantee on the consistency of the state of the local view of each agent.

Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

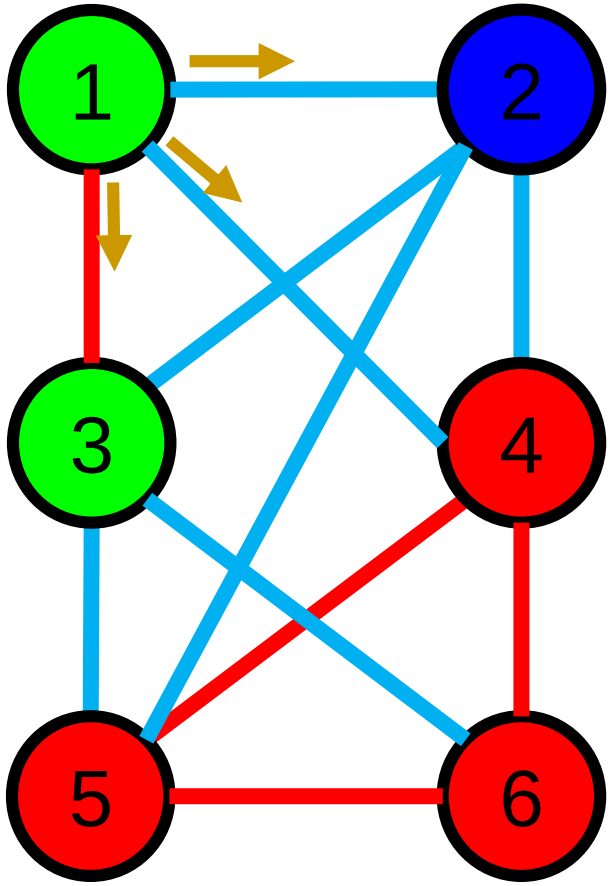
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

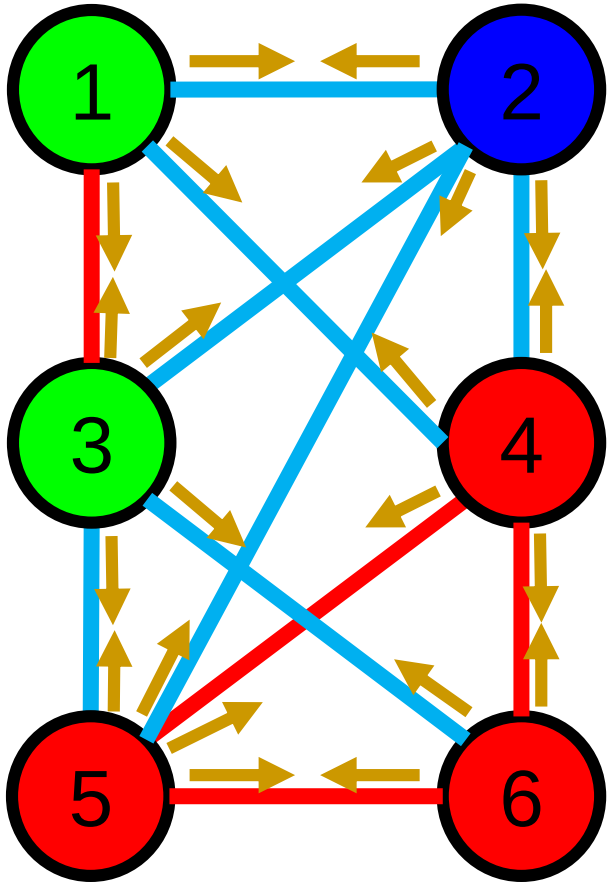
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

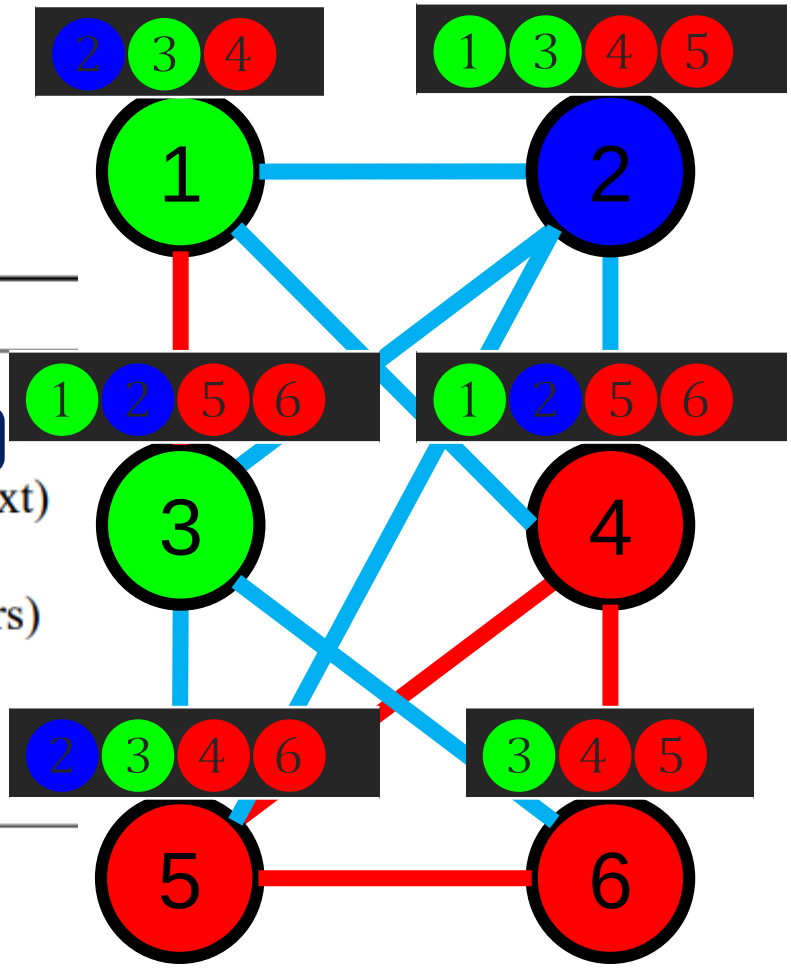
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

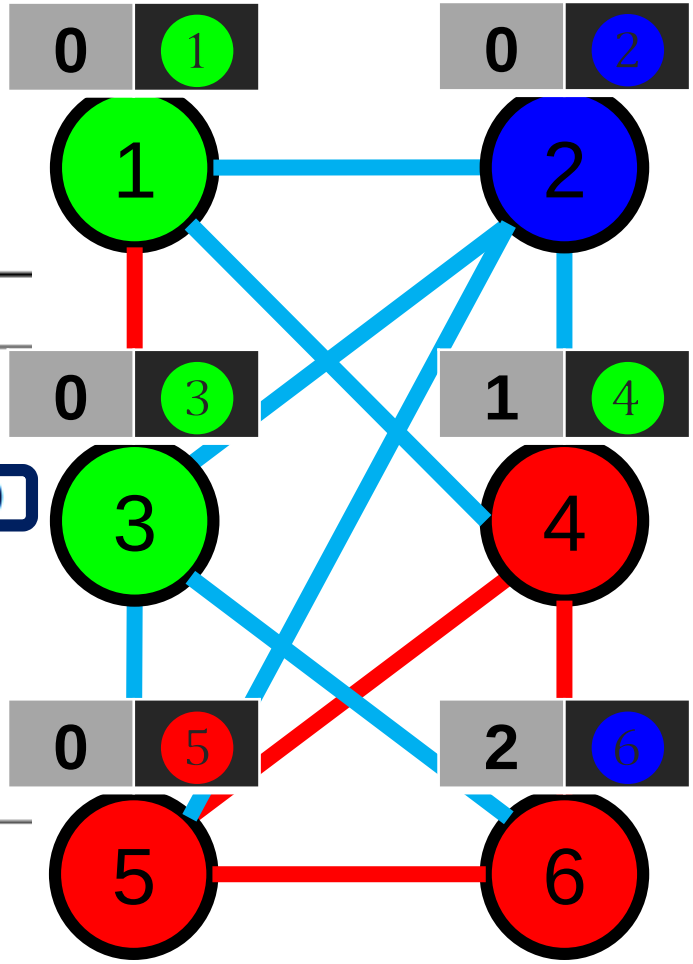
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

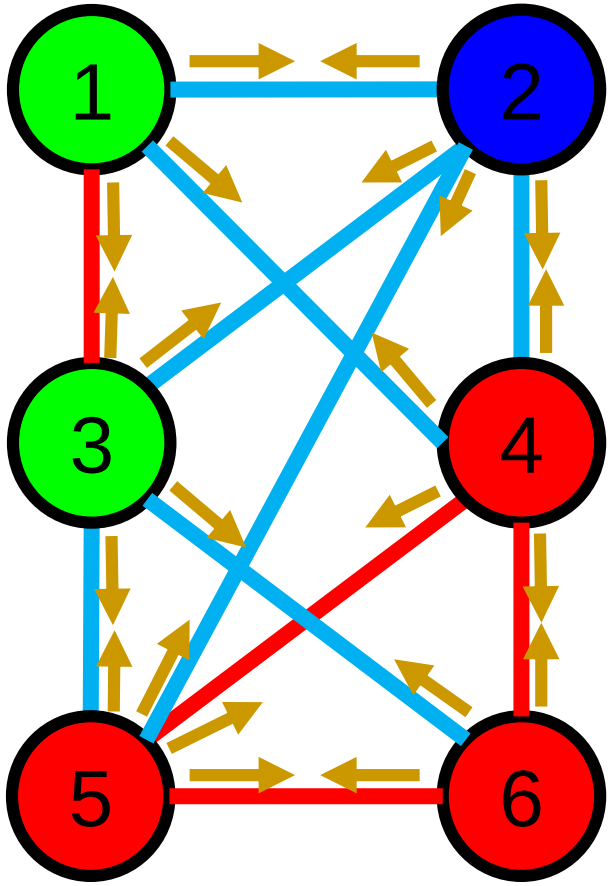
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

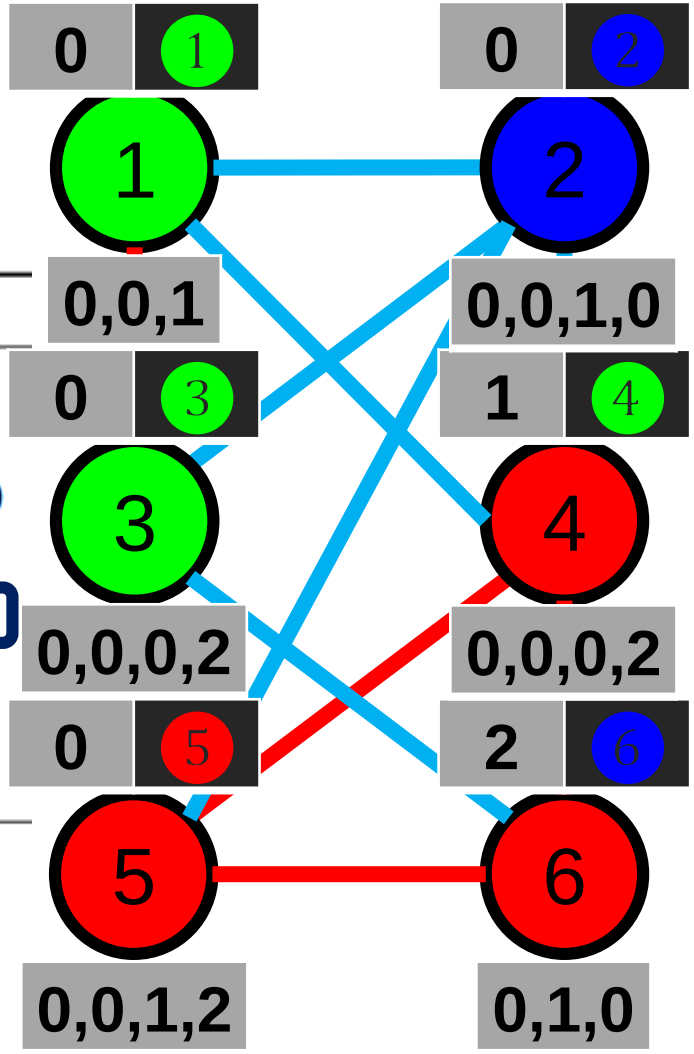
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

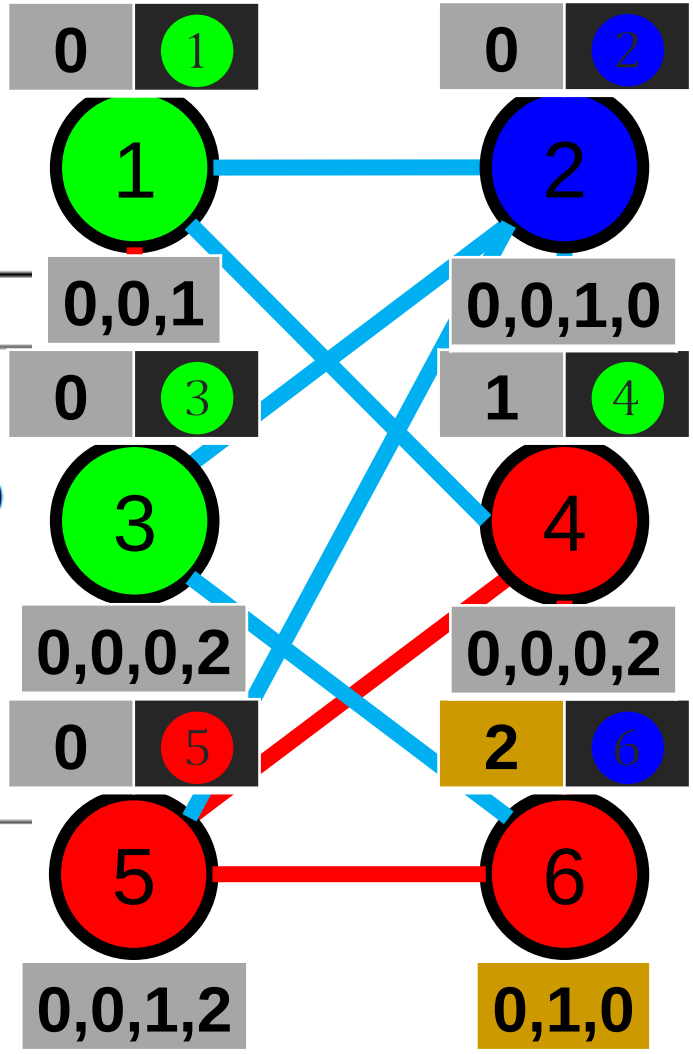
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

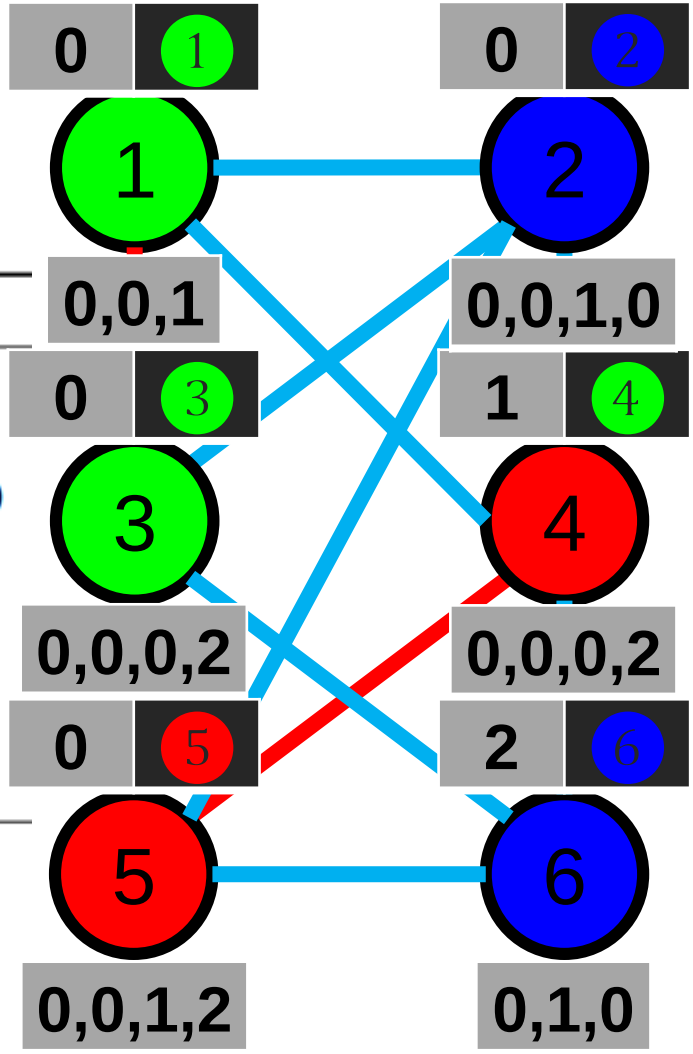
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

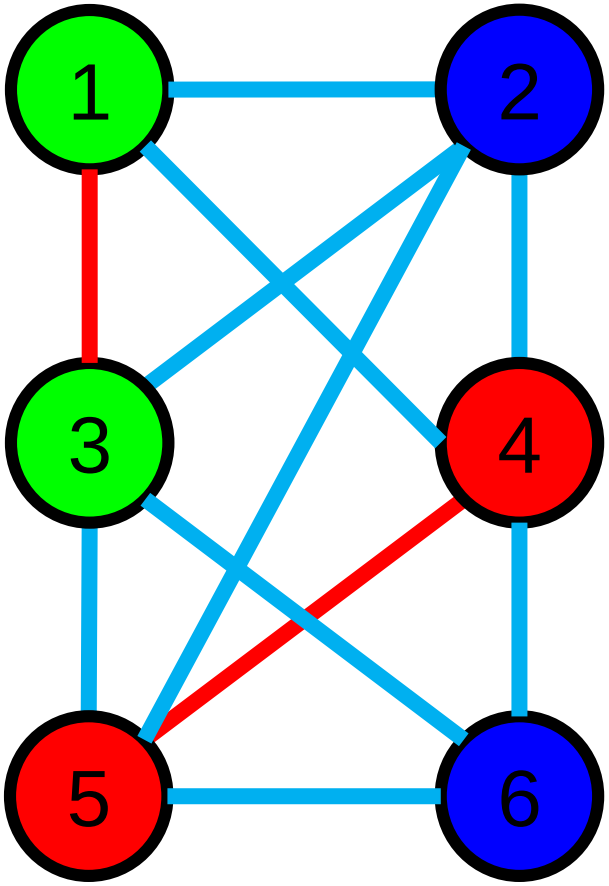
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

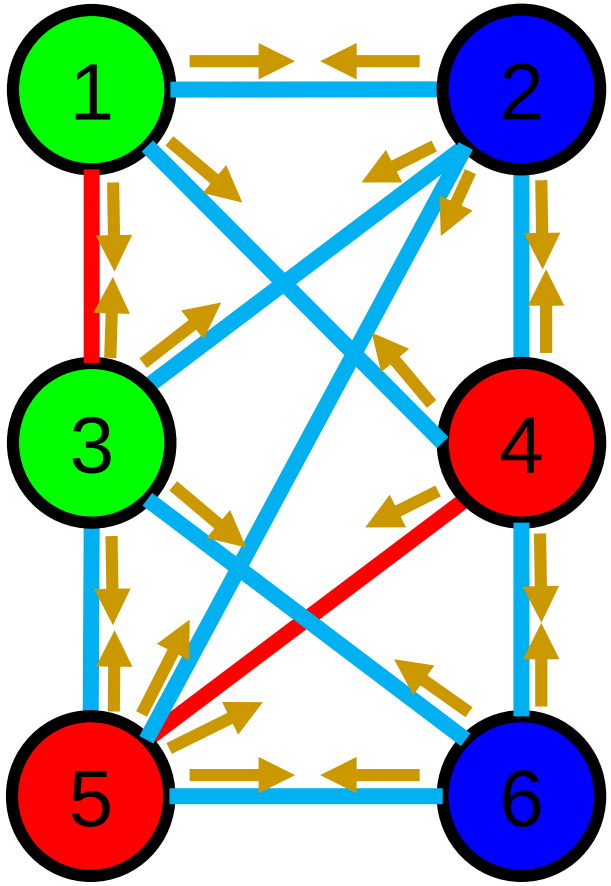
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

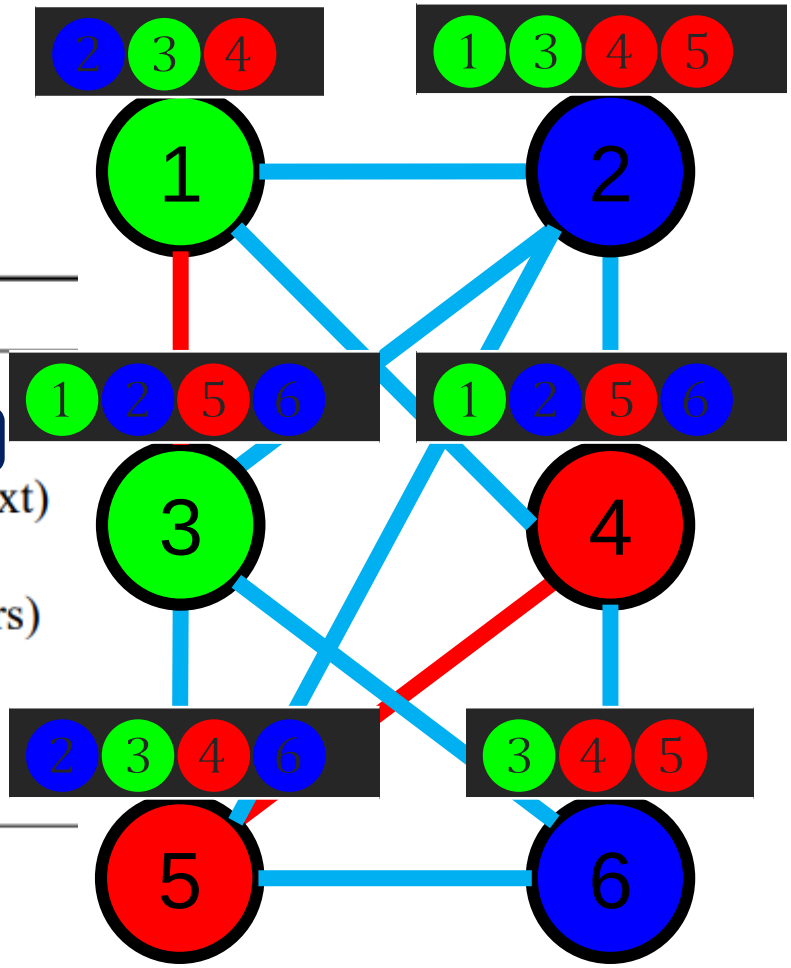
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

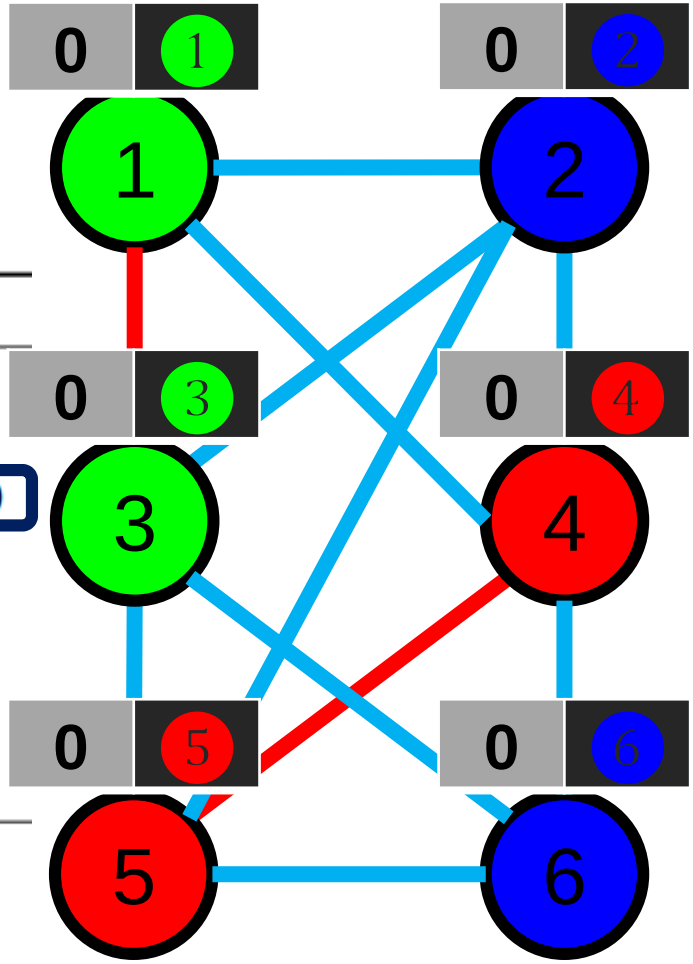
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

Algorithm 1 MGM (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Execution example

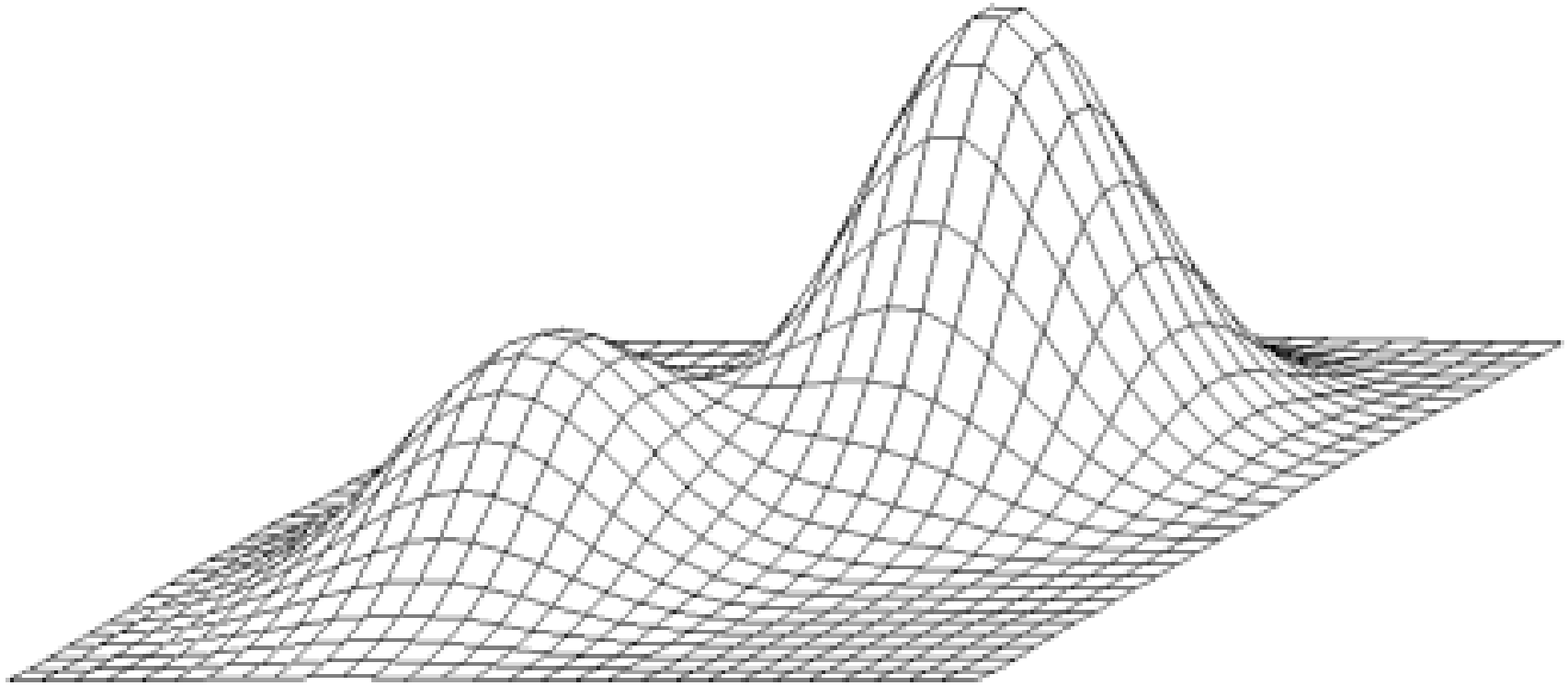
Algorith e)

1: Send
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: **if** gain > max(neighborGains) **then**
7: currentValue = newValue
8: **end if**

MISSION FAILED

What's happened?

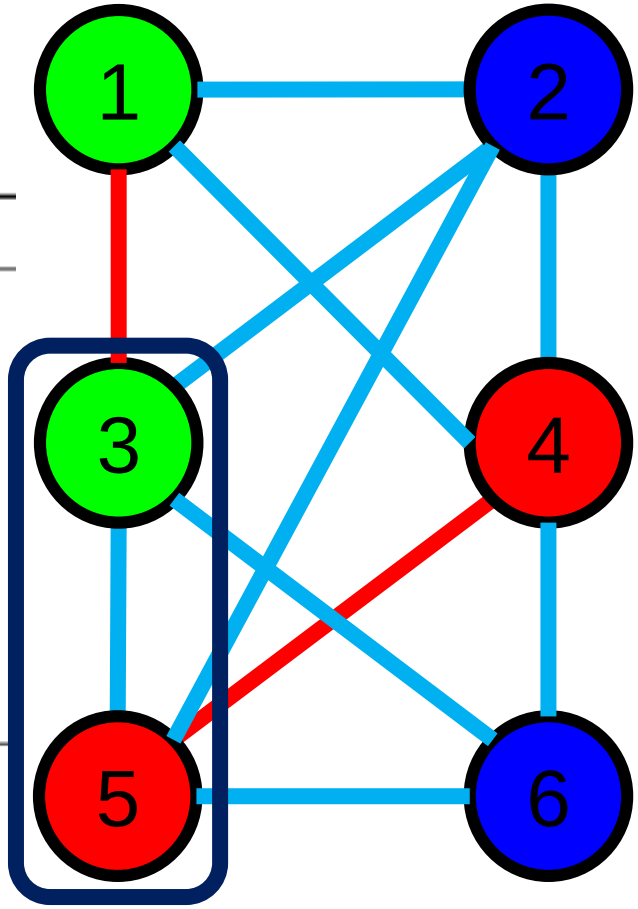
- Local maximum vs global maximum



Local maximum or optimal solution?

Algorithm 1 MGM (allNeighbors, currentValue)

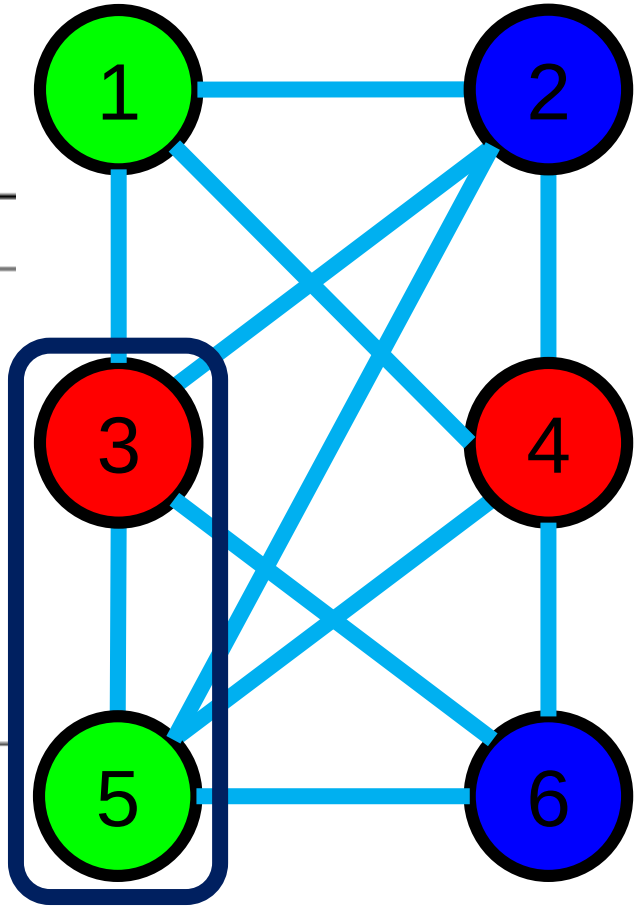
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



Local maximum or optimal solution?

Algorithm 1 MGM (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```



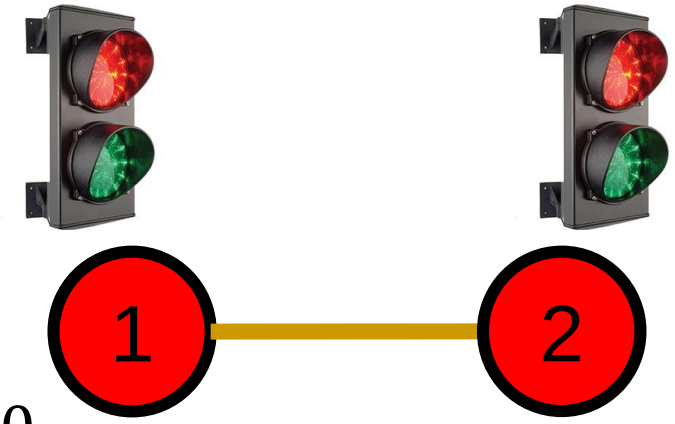
MGM vs DSA (Distributed Stochastic Algorithm)

- DSA: Instead of using gain messages, all agents generate a random value and act only if the number thus generated is lower than a fixed threshold

Algorithm 1 MGM (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```

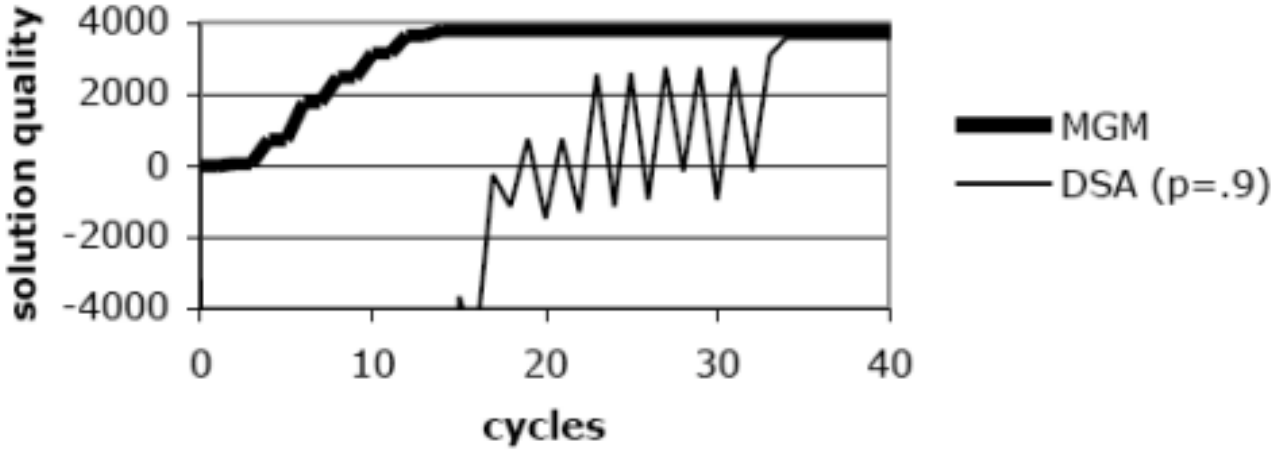
MGM vs DSA: the traffic light game



- $U(red, red) = 0$
- $U(red, green) = U(green, red) = 1$
- $U(green, green) = -1000$
- If both agents are allowed to alter their value in the same round, we would end up in the adverse state (green, green)
- When using DSA, there is always a positive probability for any time horizon that (green, green) will be the resulting assignment

MGM vs DSA: no risk with MGM

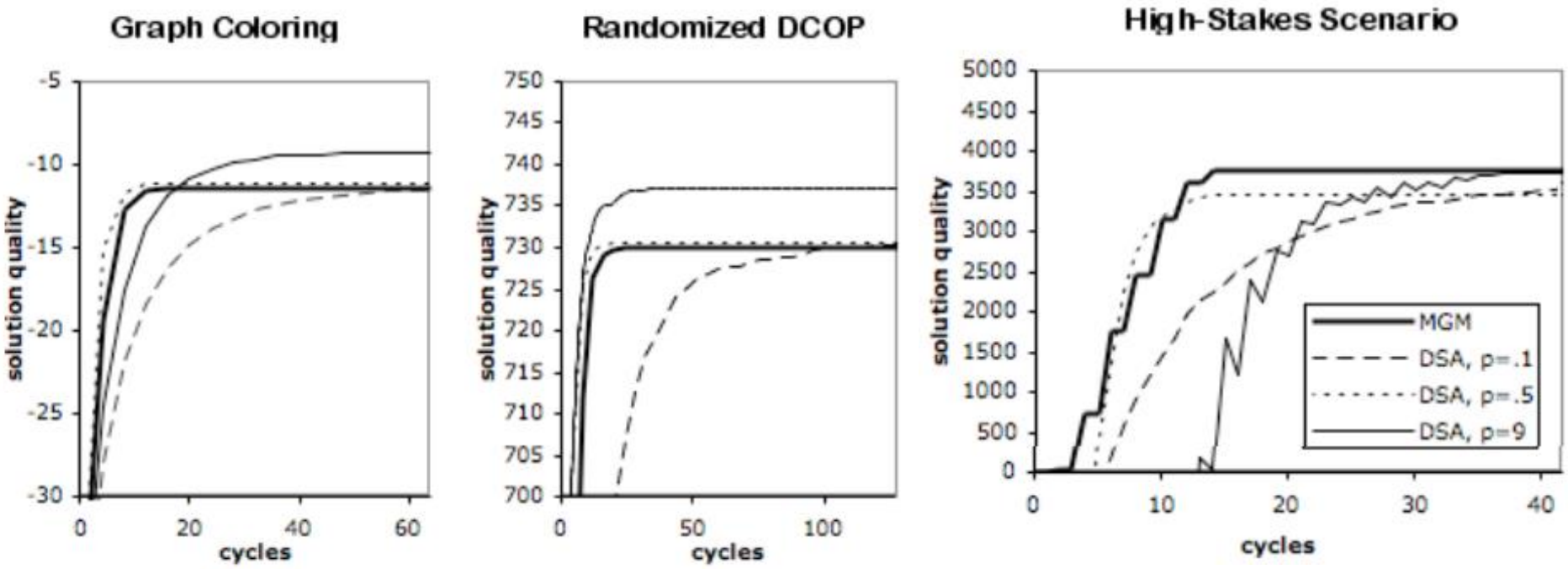
- Proposition: When applying MGM, the global utility is strictly increasing at each round until convergence
- In this high-stakes environment, agents negotiate over the use of resources, and if two agents decide to use the same resource, the result could be catastrophic



Sample Trajectories of MGM and DSA for a High-Stakes Scenario

MGM vs DSA: performance comparison

- Nature of the constraint utility function makes a fundamental difference in which algorithm dominates



Sample Trajectories of MGM and DSA (with three different thresholds)

Complexity of the MGM algorithm

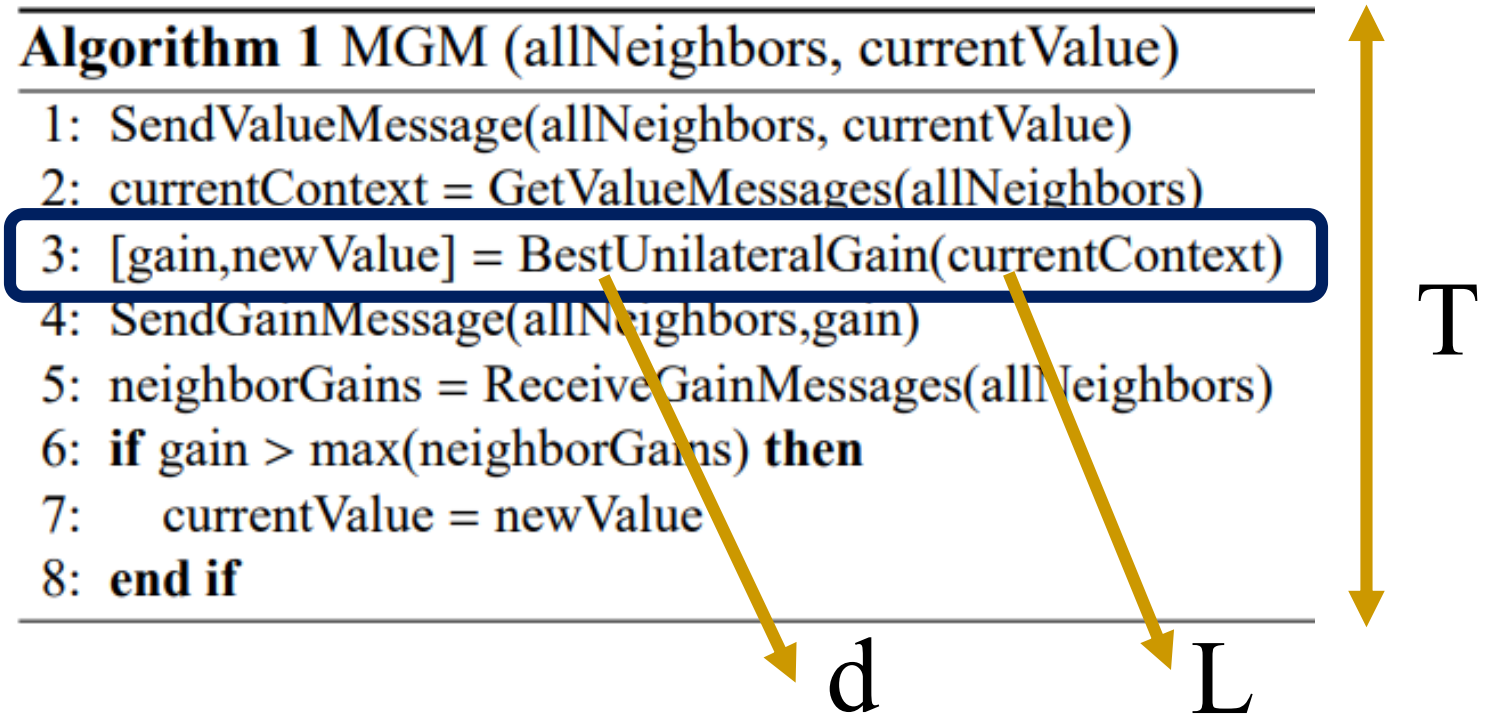
- Runtime complexity: O(TLd), where:
  - T refers to the number of iterations
  - L refers to the largest number of neighboring agents
  - d refers to the size of the largest domain;
- Space complexity per agent: O(L)
- Number of messages: O(TLn)
- Message size: O(1)

Complexity of the MGM algorithm

- Runtime complexity: O(TLd), where:
  - T refers to the number of iterations
  - L refers to the largest number of neighboring agents
  - d refers to the size of the largest domain;
  - **In ADOPT this is $O(d^n)$**
- Space complexity per agent: O(L)
- Number of messages: O(TLn)
- Message size: O(1)

Runtime complexity of the MGM algorithm

- Runtime complexity: O(TLd)



Space complexity per agent of the MGM algorithm

- Space complexity per agent: O(L)

Algorithm 1 MGM (allNeighbors, currentValue)

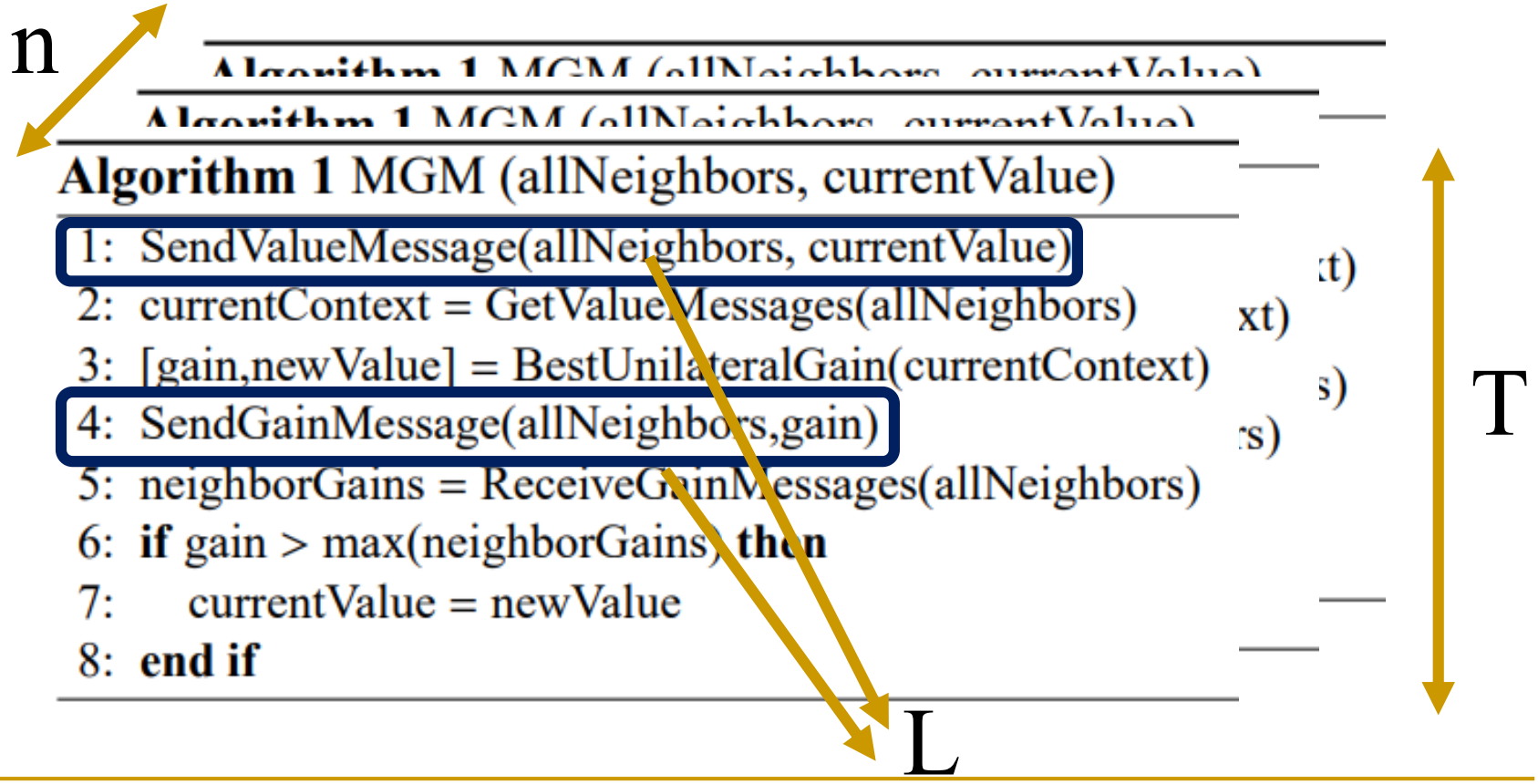
```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```

Message size: O(1)

L

Number of exchanged messages in the MGM algorithm

- Number of messages: O(TLn)



Message size in the MGM algorithm

- Message size: O(1)

VALUE MESSAGE

- Sender
- Color (3 integers)

GAIN MESSAGE

- Sender
- Gain (real number)

Quality of the MGM algorithm

- MGM is incomplete, as just seen
- MGM provides no quality guarantees on the returned solution
  - No error bounds
- MGM is anytime since agents only change their values when they have a non-negative gain

MGM-2: a new 2-coordinated algorithm

- DSA
  - DSA has a cost of one cycle per round
- MGM
  - MGM has a cost of two cycles per round
- MGM-2
  - MGM-2 requires five cycles (value, offer, accept/reject, gain, go/no-go) per round.

Algorithm 2 MGM2 (allNeighbors, currentValue)

```
 1: SendValueMessage(allNeighbors, currentValue)
 2: currentContext = GetValueMessages(allNeighbors); committed = no
 3: if Random(0,1) < offererThreshold then
 4:    committed = yes; partner = RandomNeighbor(allNeighbors)
 5:    SendOfferMessage(partner,allCoordinatedMoves(partner))
 6: end if
 7: [gain,newValue] = BestUnilateralGain(currentContext)
 8: offers = ReceiveOffers(allNeighbors); offerReplySet = ∪ {offers.neighbor}
 9: if committed = no then
10:    bestOffer = FindBestOffer(offers)
11:    if bestOffer.gain > gain then
12:       offerReplySet = offerReplySet \{ bestOffer.neighbor}
13:       committed = yes; partner = bestOffer.neighbor
14:       newValue = bestOffer.myNewValue; gain = bestOffer.gain
15:       SendOfferReplyMessage(partner, commit, bestOffer.partnerNewValue, gain)
16:    end if
17:    for all neighbor ∈ offerReplySet do
18:       SendOfferReplyMessage(neighbor, noCommit)
19:    end for
20: end if
21: if committed = yes then
22:    reply = ReceiveOfferReplyMessage(partner)
23:    if reply = commit then
24:       newValue = reply.myNewValue; gain = reply.gain
25:    else
26:       committed = no
27:    end if
28: end if
29: SendGainMessage(allNeighbors,gain)
30: neighborGains = ReceiveGainMessages(allNeighbors); changeValue=no
31: if committed=yes then
32:    if gain > max(neighborGains) then
33:       SendConfirmMessage(partner, go)
34:    else
35:       SendConfirmMessege(partner, noGo)
36:    end if
37:    confirmed = ReceiveConfirmMessage(partner)
38:    if confirmed=yes then
39:       changeValue=yes
40:    end if
41: else
42:    if gain > max(neighborGains) then
43:       changeValue=yes
44:    end if
45: end if
46: if changeValue=yes then
47:    currentValue = newValue
48: end if
```

MGM-2

- Value cycle
  - Like MGM
- Offer cycle
  - We decide which subset of agents are allowed to make offers. If an agent is an offerer, it cannot accept offers from other agents. All agents who are not offerers are *receivers*
  - Each offerer will choose a neighbor at random (uniformly) and send it an offer message consisting of all coordinated moves between the offerer and receiver that will yield a gain in local utility to the offerer under the current context
  - The offer message will contain both the suggested values for each player and the offerer's local utility gain for each value pair

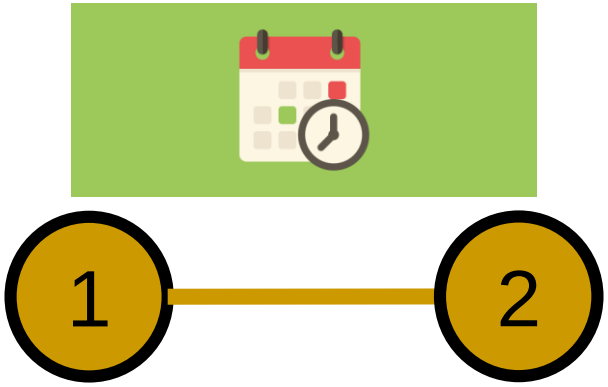
MGM-2

- Accept/Reject cycle
  - Each receiver will then calculate the global utility gain for each value pair in the offer message by adding the offerer's local utility gain to its own utility change under the new context and subtracting the difference in the link between the two so it is not counted twice
  - If the maximum global gain over all offered value pairs is positive, the receiver will send an accept message to the offerer with the appropriate value pair, causing both offerer and receiver to be committed. Otherwise, it sends a reject message to the offerer, and neither one is committed
- Gain cycle
  - Committed agents send the global gain for their coordinated move

MGM-2

- Go/No-go cycle
  - Committed agents send their partners a go message if all the gain messages they received were less than the calculated global gain for the coordinated move and send a no-go message, otherwise. A committed agent will only modify its value if it receives a go message from its partner

MGM vs MGM-2: meeting scheduling

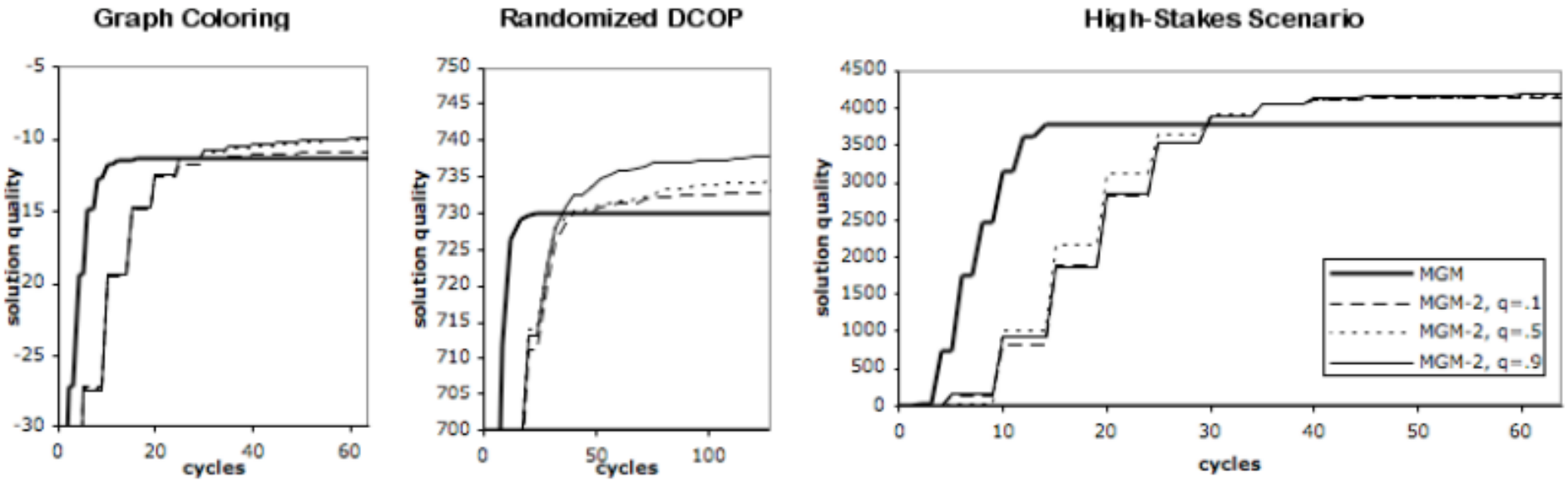


- $U(7,7) = 1$
- $U(7,1) = U(1,7) = -100$
- $U(1,1) = 10$
- Consider two agents trying to schedule a meeting at either 7am or 1pm. If the agents started at $(7,7)$, no 1-coordinated algorithm would be able to reach a global optimum, while 2-coordinated algorithms would.

MGM vs MGM-2: monotony and convergence

- As for MGM, we can show that MGM-2 is monotonic in global utility.
- Let us refer to the set of terminal states of the class of 2-coordinated algorithms as X_{2E}, i.e. neither a unilateral nor a bilateral modification of values will increase sum of all constraint utilities connected to the acting agent(s) if $x \in X_{2E}$.
- Proposition: For a given DCOP and its equivalent game, we have $X_{2E} \subseteq X_{NE}$

MGM vs MGM-2: performance comparison

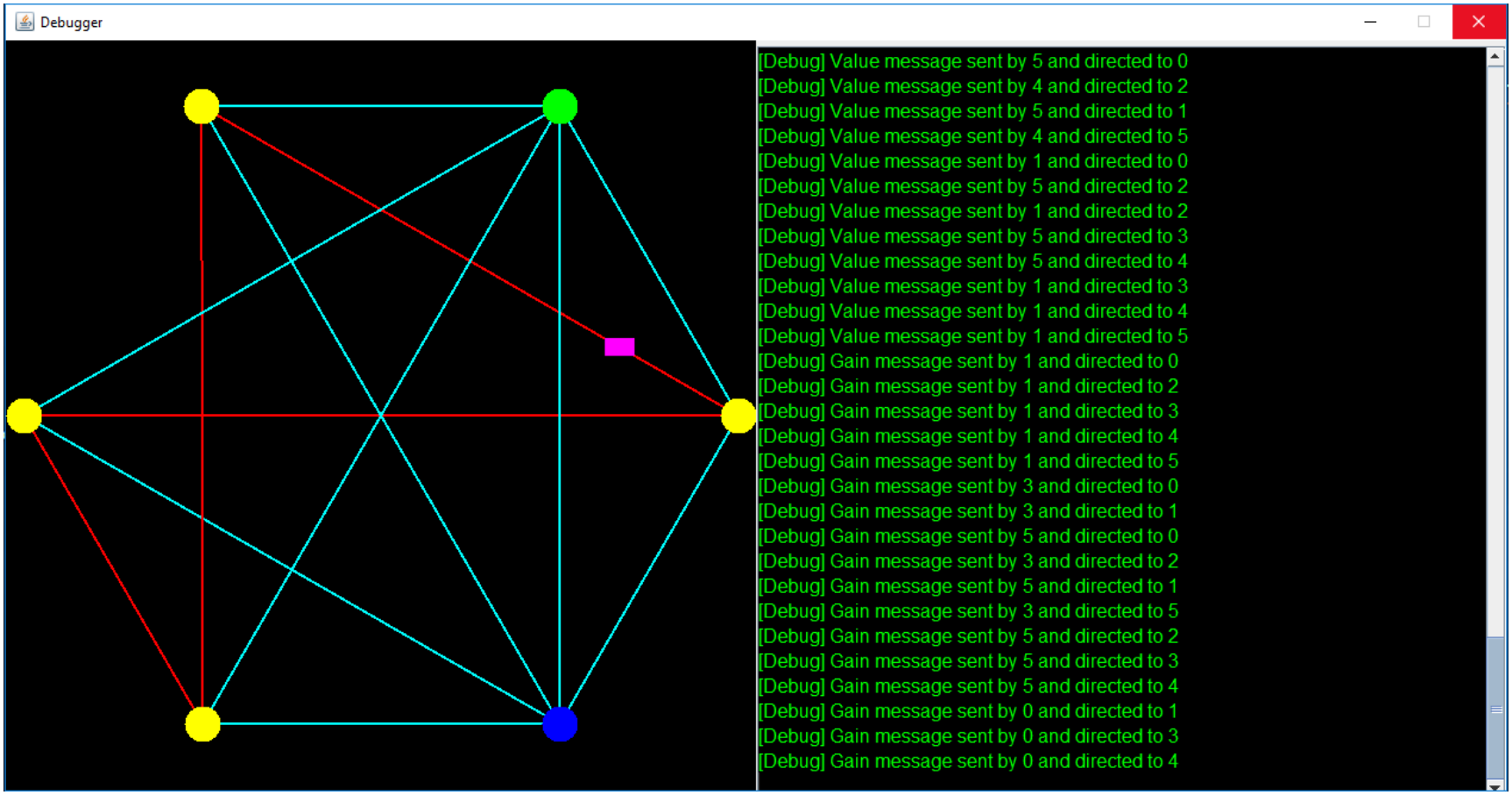


Sample Trajectories of MGM and MGM-2 (with three different thresholds)

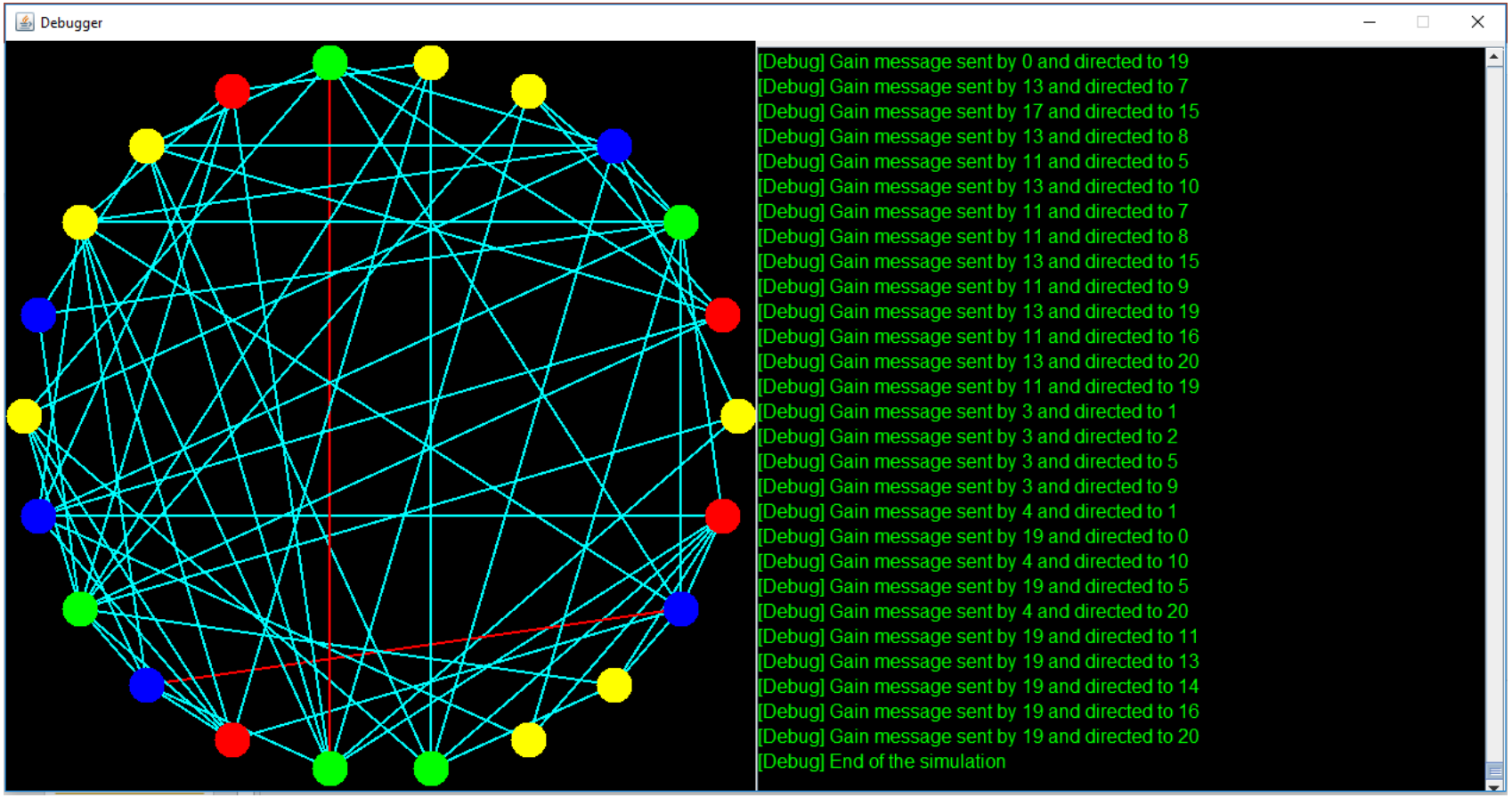
Conclusion

- Apply MGM requires less resources (time included) compared to ADOPT
- Both MGM and MGM-2 are incomplete but monotonic: the global utility is strictly increasing at each round until convergence
- In general we can't say that MGM is better than DSA or vice-versa
- MGM-2 converge slowly compared to MGM but provides, on average, solutions with a better quality

Software demonstration



Software demonstration



Thank you

- Questions?